OPTIMIZED RECALCULATION FOR SPREADSHEETS WITH THE USE OF SUPPORT GRAPH

Morten Poulsen – ykok@itu.dk
Poul Peter Serek – ppserek@itu.dk

Supervisor:
Peter Sestoft

IT-University 2007

Abstract

The goal of this project is to implement and further improve the idea of a support graph as an extension to a spreadsheet application as presented by Peter Sestoft [CORE]. A support graph is used to perform as few recalculations as possible when data in a spreadsheet is altered. This is done by making a support graph that contains the dependencies between cells in a spreadsheet. When changing the content of a cell the support graph is used to find the cells depending on the changed cell, so these and only these can be recalculated, thereby performing a minimum recalculation.

The project presents ideas and implementations with focus on the speed of recalculation, compactness of the support graph and precision of the support graph. Also benchmarking is performed to compare performance to that of Excel 2007 and OpenOffice Calc 2.1.

The project concludes that although the extension of a support graph does improve the spreadsheet application it does not reach the performance seen by Excel 2007 or OpenOffice Calc 2.1. However ideas to further improvement of the support graph are identified.

Foreword

This Master's thesis is written by Poul Peter Serek and Morten Poulsen in the period from September 2006 to March 2007 at the IT University of Copenhagen.

We would like to thank Peter Sestoft, our supervisor, for a very educational and entertaining supervising and Lasse Poulsen for proofreading and suggestions. A special thanks goes to Marianne Lindhart for enduring our bad jokes, mess and for all the cakes and food she made us during the project.

Index:

1 Introduction

When changing the value of one or more cells in a spreadsheet, a number of the spreadsheet formula cells will most likely need to be recalculated. A simple recalculation can be performed by recalculating all cells whenever any change occurs, but this involves recalculating cells which are not influenced by the changed values, which will often introduce a major, unnecessary overhead to the spreadsheet application. Another approach is to perform only the minimally required recalculations, i.e. only recalculating cells that are either directly or indirectly influenced by the changed content. This dependency information can be stored in a so-called support graph. The motivation for such a minimal recalculation is mainly faster recalculation times than recalculating the whole spreadsheet. This gives a much more satisfying and interactive user experience with fast response times, especially when changing values in large and/or complicated spreadsheets.

There is generally a lack of public accessible information regarding spreadsheets and their implementation, including the use and implementation of support graphs. It is know from several independent sources [CORE, page 52-53] that Excel, Microsoft's implementation of a spreadsheet application, uses a support graph. The usage and structure is, however, unknown to the general public. Excel is interesting as its performance in regards to speed of recalculation and size in terms of memory is far greater than competing spreadsheet applications.

The article "A Spreadsheet Core Implementation in C#" by Peter Sestoft (2006, ITU-TR-2006-91) [CORE] describes effective methods and structures to build and maintain a support graph. Our ambition is to design and implement an effective support graph based on this paper.

1.1 Problem formulation

The goal of this paper is to design and implement a prototype support graph as an extension to an existing spreadsheet application. Three main evaluation criteria for the support graph are used:

- Speed of recalculation
- Compactness of the support graph
- Precision of the support graph

The speed of recalculation is the time it takes to perform an evaluation of all changed cells and its dependencies, that is the time it takes from altering a cell until the values of the spreadsheet are updated.

The compactness of the support graph is important so it can fit in memory and not affect the recalculation speed negatively. The precision describes the relationship between the actual number of cells that are recalculated compared to the minimum number that actually need recalculation.

Furthermore the performance of the prototype is to be benchmarked and compared against known spreadsheet applications, namely Excel 2007 and OpenOffice Calc 2.1 and against the spreadsheet application it extends according to the evaluation criteria stated above.

In short the problem formulation is:

"Design and develop a support graph as an extension and thereby improvement of a spreadsheet application, with focus on the speed of recalculation, compactness and precision. The goal is to achieve similar performance to that of Excel 2007 and OpenOffice Calc 2.1"

1.2 Report structure

This paper first of all presents the necessary background information in order to understand the design and implementation of the support graph implementation. This includes theoretical background knowledge regarding finite arithmetic progressions (FAP), which can be used to represent multiple cells in a spreadsheet compactly. This is followed by an analysis of how a support graph can be created and maintained efficiently. An extension to this part is specific implementation considerations. A description of relevant and interesting parts of the complete program is presented next. Our problem formulation is tested in the next section where we present a series of benchmarks, which show various aspects of our program and compare it to other similar spreadsheet applications. Finally, a discussion of the benchmark results follows, ending with a conclusion of whether our support graph implementation achieves the goals stated in the problem formulation.

1.3 Notes

Our support graph implementation will use and extend the existing CoreCalc application, which is a spreadsheet core implementation described in the technical paper [CORE]. The extension of CoreCalc with recalculation based upon support graphs is named SupportCalc.

SupportCalc is implemented simply by extending/modifying the source code of CoreCalc version 0.5. Both programs are made in C# 2.0. SupportCalc is intended as a prototype and not a fully operational spreadsheet editor.

Attached to this report is a CD-ROM that contains the implemented program, in its final version, the real-life spreadsheets benchmarked in this project as well as this report.

1.3.1 Versions

After developing the first version of SupportCalc, both memory and performance profiling was done. As the profiling results shoved various performance issues, ideas of how to solve them have been implemented in a number of newer versions.

There are currently 3 working versions of SupportCalc, called Mark I, Mark II and Mark III, where Mark I is the first version.

2 Background knowledge

This chapter introduces terms and background knowledge that will be used throughout the thesis. Spreadsheets in general are introduced first, followed by a section regarding support graphs in spreadsheets. Finally, there is a short introduction to the spreadsheet application CoreCalc used in this project.

2.1 Spreadsheets

Spreadsheets have many applications, typically within statistics and accounting. Spreadsheets consist of a rectangular table, consisting of columns and rows. Information can be contained in cells as shown in Figure 2.1.

B4	A	B	C
1	2	2	2
2	2,4	4,4	6,4
3	2,88	7,28	13,68
▶ 4	3,456	=SUM(A$1:A4)	24,416
5	4,1472	14,8832	39,2992

Figure 2.1: Example of a spreadsheet

The address of a cell can be defined by the column and row at which it resides. Typically columns values are assigned using letters (e.g. A, B… Z, AA, AB… AZ etc.) and rows with numbers. The address of a cell located at column B, row 5 would be B5. Several spreadsheets can be grouped together in a workbook and a cell in a specific sheet can be referred to by attaching the sheet name to the addressing of the cell. This is typically done by appending the sheet name in front of the cell address, e.g. “NameOfSheet!A1” [CORE, page 10].

A cell can contain simple data types like strings and numbers and complex data types like formulas. A formula can be composed of the following [CORE, page 7]:

- Simple data types (e.g. the number 42 or the string “Accounting”)
- Mathematical operators (e.g. division, multiplication)
- References to other cells and areas located in an arbitrary sheet within the same workbook (e.g. Sheet1!A1 refers to the single cell A1 in the sheet “Sheet1” while Sheet2!A1:B10 refers to the rectangular area defined by the opposing corners A1 and B10 in “Sheet2”)
- Functions (e.g. AVG() which returns the average of the input. Many functions accept simple data types, mathematical operators and references. AVG() can be called with the parameter A1:B10 to give an average over the given area)
- Error types, since spreadsheets do not have exceptions [CORE, page 28].

A cell is typically considered a formula if the cell starts with the equal operator, e.g. “=AVG(A1:B5)”. If the cell does not start with the equal operator, the cell is considered a

simple data type (if you want a string to start with "=" you can normally contain your string within single quotes which will force the cell to be considered a string).

The above format of addressing cells is called the A1 format. In addition, columns and rows can be relative or absolute. Absolute columns and rows are prefixed with the dollar sign character "$", which gives the following combinations:

- B5, both column and row are relative
- $B5, column is absolute, row is relative
- B$5, column is relative, row is absolute
- $B$5, both column and row are absolute

Absolute and relative references are only relevant when changing the location of the cell containing the references. With an absolute reference the absolute part of the address will always point to the same location. Relative references will have their reference adjusted when the cell containing them is moved. E.g. when copying a formula from cell A1 to B3, B3 will contain a copy of A1's formula but with its relative references adjusted. The adjustment is relative to the movement of the cell. E.g. if the cell A1 contains a reference to the cell C5 and the cell A1 is copied to B3, then the relative reference must be adjusted to D7 since the cell moved a single column to the right and two rows down. A cell can also be sheet relative or absolute.

Other addressing formats exist, e.g. R1C1 and C0R0. R1C1 is known from the spreadsheet applications Microsoft Excel [EXCEL] and OpenOffice [CALC] and is defined by the row first, followed by the column [CORE, page 9]. Absolute and relative columns and rows are defined by which numerical position the cell is located in the sheet. Absolute columns and rows are defined by using their numerical position (e.g. R1C1). Relative columns and rows use a numerical value that represents the relative distance from the cell itself to the reference. Relative columns and rows are differentiated from absolute ones by surrounding the numerical difference in square brackets.

The C0R0 format is used by CoreCalc and is very similar to R1C1, the only difference being that indexing of rows and columns begin at 0, not 1, and column numbers come before row numbers [CORE, page 29]. Concrete examples of all previous mentioned formats are summarized in Figure 2.2 where B2 (in A1 format) is the current cell.

A1 format	R1C1 format	C0R0 format	Interpretation
B2	RC	C[0]R[0]	Relative; current cell
A2	RC[-1]	C[-1]R[0]	Relative; previous column, this row
A1	R[-1]C[-1]	C[-1]R[-1]	Relative; previous column and row
C3	R[+1]C[+1]	C[1]R[1]	Relative; next column and row
$A$2	R2C1	C0R1	Absolute; column 1 (A), 2. row
$A$1	R1C1	C0R0	Absolute; column 1 (A), 1. row
$B$2	R2C2	C1R1	Absolute; column 2 (B), 2. row
$C$3	R3C3	C2R2	Absolute; column 3 (C), 3. row
$A1	R[-1]C1	C0R[-1]	Absolute column 1 (A), relative previous row

Figure 2.2: Cell addressing format description [CORE, page 9, modified]

In the examples above only cells within the same sheet can be addressed. Each of the formats can prefix a sheet name separated by an exclamation mark as shown:

Sheet2!A$1

This allows cross sheet references. A notion of dependencies arises from the use of references. If A1 refers to B1, then B1 is said to support A1. It is also said the A1 depends on B1. A dependency between two cells can be direct, as is the case with the previous example, or indirect. An indirect dependency is defined by a non-empty chain of direct dependencies through other cells that connect the two cells.

The presence of references makes it possible to have cyclic dependencies, e.g. if a cell A1 refers to B1 and B1 refers to A1. Generally speaking, if a cell refers to itself, direct or indirect, then a cyclic dependency is present. Cyclic dependencies can be difficult to detect, since a cell may not always evaluate to the same reference, and hence only create cyclic dependencies when some condition is fulfilled. E.g. “IF(RAND()>0.5; B1; 2)” only refers to B1 half the time and if B1 refers to the cell with this formula, then we have a cyclic dependencies half the time. A cyclic dependency can be static or dynamic. The difference is illustrated in Figure 2.3:

	A	B
1	=IF(A2<1; B1; B2)	=A1
2	5	42

	A	B
1	=IF(A2<5; B1; B2)	=A1
2	0	42

Figure 2.3: Examples of static and dynamic dependencies

A cyclic dependency is said to be static if any of the two possible references B1 and B2 can cause a cyclic dependency. In the example to the left the IF statement evaluates to B2 which causes a static cyclic dependency according to the definition since B1 refers back to A1. This static cycle is harmless recalculation wise since it does not create an actual cyclic dependency, called a dynamic dependency. A dynamic cyclic dependency occurs when the evaluated IF statement causes an actual cycle. In the first example we do not have a dynamic cyclic dependency since B2 gets chosen. In the example to the right we do have a dynamic cyclic dependency since A1 evaluates to B1 and B1 refers back to A1.

To sum up, static cyclic dependencies are harmless as long as they do not contain dynamic cycles.

2.2 CoreCalc

CoreCalc is a spreadsheet application implementation developed by Peter Sestoft [CORE]. The lack of literature surrounding spreadsheets and their implementation has been the primary motive for CoreCalc. The purpose of CoreCalc is to act as a catalyst to other spreadsheet related research projects which can use CoreCalc as the core implementation. Alternatives to CoreCalc exist, like Gnumeric and OpenOffice, but the source code is substantially larger and more complex than CoreCalc's.

CoreCalc can use the A1, R1C1 and C0R0 formats mentioned in section 2.1, but the C0R0 format is used as an internal representation for all formulas. In C0R0 any reference from one cell to other cells will be relative to its own position, making it very easy to move cells and share cell formulas with other cells. The space saving achieved with formula sharing can be substantial, since formulas can be copied to many cells.

Currently CoreCalc has the following built-in functions ready for use:

- Mathematical functions (e.g. SIN and COS)
- Functions which takes multiple arguments (e.g. SUM and AVG)
- Functions which returns a matrix (e.g. TRANSPOSE and MMULT)
- Conditional functions (e.g. IF)
- Volatile functions (e.g. RAND and NOW)

Furthermore it has support for basic mathematical operators like multiplication, division, addition and subtraction [Core, page 32]. The specific functions are not interesting with regards to recalculation of the spreadsheet, but the different types are. E.g. CoreCalc does not allow insertion or removal of rows and columns which would break any matrix. Volatile functions in particular are troublesome in recalculations, since they can evaluate to a new value each time any modification is made to the spreadsheet, even if the modification does not directly or indirectly depend on the cell containing the volatile functions [CORE, page 35].

CoreCalc contains a simple recalculation mechanism, which recalculates all cells contained in a workbook when any cell is changed. The recalculation makes use of two boolean fields, visited and up-to-date, both false when a new recalculation is initiated. Visited is used to discover cyclic dependencies and CoreCalc throws an exception if this is the case since these are not allowed by CoreCalc. Up-to-date is used to avoid recalculation of cells which have already been recalculated (thereby enforcing recalculation in dependency order).

We refer to [CORE, page 31] for a detailed description. The main purpose of this paper is to replace CoreCalc's recalculation mechanism with one that uses a support graph.

CoreCalc contains a simple graphical user interface shown in Figure 2.4.

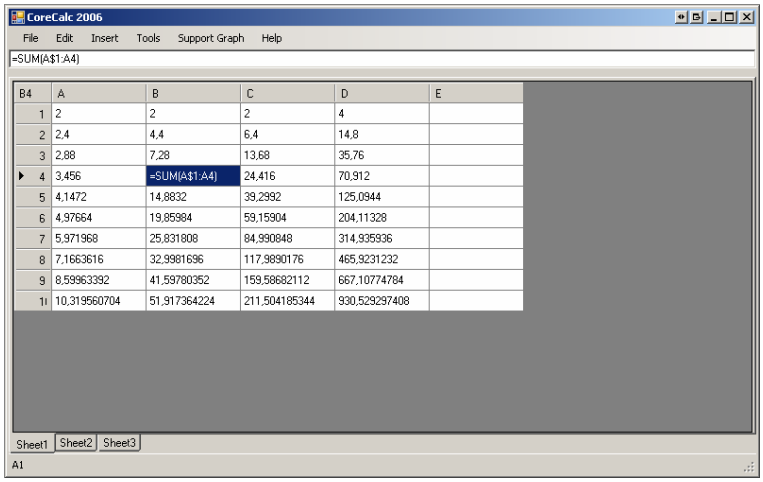


Figure 2.4: Screenshot of CoreCalc's user interface

It contains basic functionality to manipulate and display spreadsheets and can import spreadsheets saved in Excel's or Gnumeric's [GNU] XML format.

3 Managing a support graph

This section presents an analysis of how to build and maintain a support graph in CoreCalc. It is divided into four parts:

- The general idea of a support graph
- The structure of the Support graph
- The initial building of a support graph from an existing spreadsheet
- The maintenance of the support graph when editing the spreadsheet

The reader is encouraged to have read the previous chapter, Background knowledge, or at least have a basic understanding of spreadsheets before proceeding with this section.

The main part of this section is based on the first version, Mark I, whereas version changes to Mark II are described independently in section 3.7. The changes to Mark III are not described in this chapter as this is more in relation to the implementation, but is described in chapter 4, "Development – different versions".

3.1 General idea of a support graph

When a spreadsheet is edited, e.g. by changing the content of a cell or deleting columns and rows, a recalculation may be needed. However, not all cells need to be recalculated - only those that directly or indirectly depend on the altered cells. In order to find these cells we need to know for each cell *c*, which other cells refer to it, or rather which other cells *c* supports. If a cell *c* is changed we can find the cells that need to be recalculated by finding the set *T* of cells that *c* support recursively[1] until we find cells that do not support anything. However the order of recalculating is important, which is illustrated in Figure 3.1. If B2 is altered we need to update C2 and B7, but since B7 also supports C2 we need to recalculate B7 before C2.

	A	B	C
1		Expenses	% of sum
2	Food	100	=B2/$B$7
3	Car	2000	=B3/$B$7
4	Hardware	500	=B4/$B$7
5	Software	1000	=B5/$B$7
6			
7	Sum	=SUM(B2:B5)	
8			

Figure 3.1: The dotted lines illustrates supported cells

To archive correct recalculation order a topological sorting is used, which is described in section 3.3.

[1] Finding supported cells for each cell in *T* and for each of those find supported cells etc.

3.1.1 Concept of a support graph

A support graph is an explicit representation of cell dependencies. Graphs like these can be used to recursively recalculate a cells dependencies as illustrated above. A recalculation is said to be minimal if only cells that directly or indirectly depend on the changed cells are recalculated. The opposite is to recalculate the whole workbook, which usually takes more time. This we call a full recalculation. The challenge is to have a compact representation of the support graph, since the graph in a worst case scenario will have $O(N_A^2)$ edges, where N_A is the amount of active cells[2]. To prevent the support graph from using $O(N_A^2)$ memory we use a representation called FAP set [CORE]. Figure 3.2 shows a worst case example.

	A
1	=SUM(A2:A8)
2	=SUM(A3:A8)
3	=SUM(A4:A8)
4	=SUM(A5:A8)
5	=SUM(A6:A8)
6	=SUM(A7:A8)
7	=SUM(A8:A8)
8	

Figure 3.2: The maximum number of references in a sheet without any static cycles

3.1.2 Introduction to FAP sets

FAP stands for finite *arithmetic progression*[3] and describes a finite sequence of numbers, e.g. {1,3,5,7} or {203, 209, 215, 221}. A FAP set is defined by three parameters:

- *a*, defines the starting point of the sequence
- *b*, defines the periodic distance between consecutive numbers in the sequence
- *k*, defines how many numbers are present in the finite sequence

A FAP set is described by the triple: *(a, b, k)*

Some examples of FAP sets and their corresponding sequence can be seen in Figure 3.3.

FAP set	Sequence of numbers representing the FAP set
(203, 6, 4)	{203, 209, 215, 221}
(12, 12, 12)	{12, 24, 36, 48, 60, 72, 84, 96, 108, 110, 112, 120}

Figure 3.3

3.1.3 Usage of FAP sets

Often when a cell supports large quantities of cells these will be either in direct continuation of each other or with a constant distance (periodic)[4]. This enables us to use

[2] Active cells being the number of cells with content or which at least one cell has a reference to
[3] [CORE, page 55]

FAP sets to represent the edges of the support graph. In Figure 3.4 the cell B2 support cells E2 to E7, and here the FAP set (2, 1, 6)[5] can be used to describe the row component of the cells supported by B2.

	A	B	C	D	E
1	Person	Grade		Grade	Number of people with grade
2	Bob	B		A	=COUNTIF(D2;B$2:B$8)
3	Alice	C		B	=COUNTIF(D3;B$2:B$8)
4	Frank	A		C	=COUNTIF(D4;B$2:B$8)
5	Sam	F		D	=COUNTIF(D5;B$2:B$8)
6	Simon	A		E	=COUNTIF(D6;B$2:B$8)
7	Tom	D		F	=COUNTIF(D7;B$2:B$8)
8	John	B			

Figure 3.4: The COUNTIF function counts the number of cells in an area that are equal to a value (or the value of a cell). E2 will evaluate to the value 2 since there are two cells in B2:B8 that are equal to 'A'

The idea and hope is that with FAP sets we can decrease the memory used by the support graph to a constant factor times the number of non-empty cells in a spreadsheet with many edges[6].

3.1.4 Alternative support graph

Alternatives exist to make a support graph, which does not for each cell explicit contain information about which cells are supported. These include keeping a topological sorted list that approximate the support graph, so if a change is made to a cell *c* then only those cells that appear after *c* in the topological list needs to be updated. However this still has a worst case running time of $O(N)$ where *N* is the number of active cells [CORE, page 51].

However these alternatives are not used nor examined in this report.

3.2 Structure of the support graph

3.2.1 General idea of the support graph

The support graph should represent the opposite of cell dependencies; that is it should represent which cells support other cells. So if cell C3 contains the following expression: "=A1+A2", then the support graph should contain an edge to C3 from both A1 and A2. This means that C3 depends on both A1 and A2 while both A1 and A2 support C3. When a cell is updated the support graph can then be used to determine which cells need updating by travelling recursively through all cells that depends on the changed cell.

[4] No proof is given for this, but in order to have substantial number of supported cells that are not periodic or in continuation, it would require a lot of manual work from the user or some type of automation, thus the spreadsheet would easily become complex and unstructured.

[5] If the FAP set is based on A1 format. In C0R0 format it would be (1, 1, 6).

[6] A worst case will still be $O\left(N_A^2\right)$. Approaching anywhere near this worst case is unlikely because of the typical structure in spreadsheets according to our own preliminary observations (see section 3.2.3).

3.2.2 Static vs. dynamic support graph

Static references are those references in the expression which represents the whole of the expressions expression area.

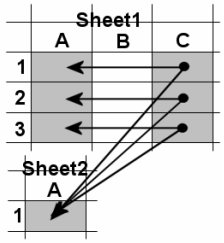
Static references are all references contained in expressions in the cells, whereas dynamic references are those references actively being used during recalculation. E.g. in the expression "IF(1=1, A1, A2)", both A1 and A2 are static references, whereas only A1 is a dynamic reference. Thereby dynamic references might change when changing a condition of a statement, that is even by changing a value of a different cell, for instance in the case of the expression "IF(1=A1, B2, B3)", changing A1 might change the dynamic references, though not changing the static references.

Note that in this example only one of the two possible dynamic references can ever be chosen at one time.

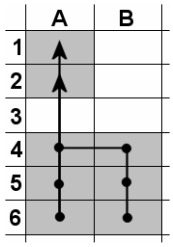
The support graph is said to be static if it contains all possible dynamic paths of every cell. Thus follows that a static support graph has at least an edge for every static reference. This makes a static support graph fairly easy to use since it should only be updated whenever static references are changed and are therefore static during a recalculation. A static support graph is used in this project although possible use of dynamic support graph is shortly discussed in section 3.8.4

3.2.3 Spreadsheet structure

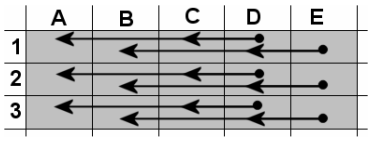
We first look at how the spreadsheet is structured in regards to references to other cells. This is not a thorough or complete analysis, but a preliminary analysis. This is helpful in determining what structure we can expect from spreadsheets and how to effectively build a support graph. In a spreadsheet, cells can be referenced as a single cell (e.g. A1) or as an area defined by two opposite corners (e.g. A1:B2). By looking at several different spreadsheets from the real world, we find the following examples of reference patterns:



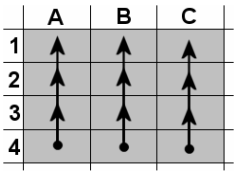
C1 through C3 refer relatively to a single cell, A, on the same row and absolutely to cell A1 in sheet 2. This corresponds to the following C0R0 references: C[-2]R[0] and C0R0



Cells in the area A4:B6 refer to the absolute area A1:A2. This corresponds to the following C0R0 references: C0R0 and C0R1



Cells in column D and E refer relatively to the previous and third previous cell on the same row. This corresponds to the following C0R0 references: C[-1]R[0] and C[-3]R[0]



Cells in row 4 refer relatively to the previous 3 cells in the same column. This corresponds to the following C0R0 references: C[0]R[-1], C[0]R[-2] and C[0]R[-3]

Figure 3.5

In the figures the “dots” represent the cells in which the reference originates and the arrowheads represent the referenced cells. What we se in general from these examples is a recurring pattern which can be described by FAP sets. Using the C0R0 format we can se that the same references are being used by several cells, which could make reference sharing beneficial. Indeed CoreCalc uses this to share not only the reference but the entire expression within a formula (e.g. sharing “SUM(C[0]R[-10]:C[0]R[-1]”).

3.2.4 FAP sets

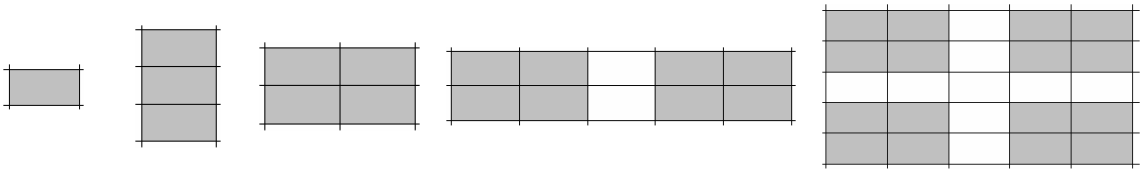
In order to describe which cells a single cell supports, we use FAP sets. Since a FAP set only describes one dimension we need two for the two dimensions of a spreadsheet. We call a two-dimensional description by FAP sets a FAP grid. The simple way would be to always use two FAP set, one in each dimension. However, by a union of several FAP sets, called a FAP list in each dimension we can describe some patterns that would otherwise require several FAP grids and overall more FAP sets. By using FAP lists it is also easier to union several FAP sets into fever sets since only one dimension is handled at a time.

The notation we use to describe these FAP grids is a FAP list with no delimiter for each FAP set with an “x” as a separator between the dimensions. In Figure 3.6 some examples of the notation is shown along with the cells represented by each FAP grid. The implicit delimiter between each FAP set in the FAP list is union. The column dimension is always described first in keeping with the C0R0 and A1 notation and both columns and rows begin from 0.

FAP Grid	**Represented cells (in A1 format)**
(0,1,2) x (4,1,2)	A5, A6, B5, B6
(1,2,3) x (1,3,2)	B2, B5, D2,D5, F2, F5

Figure 3.6

Examples of FAP grids are given below:



a) Single cell, described by the FAP grid (0,1,1) x (0,1,1)

b) Single column, described by the FAP grid (0,1,1) x (0,1,3)

c) Rectangular cell area, described by the FAP grid (0,1,2) x (0,1,2)

c) Two rectangular cell areas, described by the FAP grid (0,1,2)(3,1,2) x (0,1,2)

d) Four rectangular cell areas, described by the FAP grid (0,1,2)(3,1,2) x (0,1,2)(3,1,2)

Figure 3.7: Collection of cells described by FAP sets

Notice how example c and d in Figure 3.7 have dimensions which cannot be described by a single FAP set. It is possible to have several separate rectangular areas which can be described by a single FAP set per dimension:

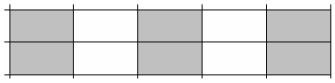


Figure 3.8: Several separate rectangular cell areas that can be described by a single FAP set pr. dimension

The cell area in Figure 3.8 can be described by (0, 2,3) x (0, 1, 2) since the size and gap between areas are identical and can therefore be described by a sequence.

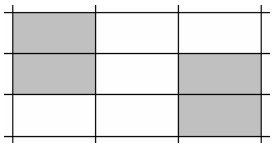


Figure 3.9: One FAP grid is not enough as the rows do not match.

A single FAP grid might not be enough to describe the cells supported by a single cell. Figure 3.9 shows two cell areas that can not be described by a single FAP grid, but needs a union of FAP grids, called a FAP grid list. In the example the cell areas can be described by:

$$(0, 1, 1) \text{ x } (0, 1 ,2) \cup (2, 1, 1) \text{ x } (1, 1, 2)$$

Each cell in the spreadsheet that supports any other cells should therefore have a collection of FAP grid lists corresponding to the supported cells. The support graph for the spreadsheet in Figure 3.10 can be seen in Figure 3.11. The support graph is defined as every FAP grid list contained in the spreadsheet. In this example all of the cells containing a FAP grid list, contains only one FAP grid. Notice how none of the cells in column *C* and *D* have a FAP grid list, since there are no cells referring to them.

	A	B	C	D
1	Pi	Radius	Circumference	Area
2	3.14	1.23	=2 · B2 · $A$2	$=B2^2 \cdot \$A\$2$
3		2.44	=2 · B3 · $A$2	$=B3^2 \cdot \$A\$2$
4		34.12	=2 · B4 · $A$2	$=B4^2 \cdot \$A\$2$
5		23.22	=2 · B5 · $A$2	$=B5^2 \cdot \$A\$2$
6		34.22	=2 · B6 · $A$2	$=B6^2 \cdot \$A\$2$

Figure 3.10: A spreadsheet with data and functions

	A	B	C	D
1				
2	(2, 1, 2) x (1, 1, 5)	(2,1,2) x (1,1,1)		
3		(2,1,2) x (2,1,1)		
4		(2,1,2) x (3,1,1)		
5		(2,1,2) x (4,1,1)		
6		(2,1,2) x (5,1,1)		

Figure 3.11: The support graph of the spreadsheet from Figure 3.10

3.2.5 Volatile functions

Volatile functions are expressions that need to be re-evaluated on every change to the spreadsheet even though there is none of their references that have been either directly or indirectly updated. These include functions such as: “Now”, which returns the current time and “Rand” which returns a random number.

These functions are put into a separate list of cells that have to be evaluated on every change. Note that the cells supported by volatile functions also needs re-evaluation. This list is also represented as a FAP grid list.

3.3 Recalculation order

When a change occurs in a spreadsheet a recalculation takes place. In this recalculation the cells that are updated are said to be re-evaluated.

The support graph and its list of volatile functions contain all information needed to re-evaluate a cell and all its dependencies. A cell can only be re-evaluated if all the cells it references are up to date, that is either they are re-evaluated or they do not need re-evaluation as neither they nor their references are updated. To avoid checks of which cells have been re-evaluated before proceeding we use a topological sorting as suggested in [CORE, page 71]. A topological sorting here means a linear ordering of cells that needs to be re-evaluated, where a directly supported cell always appear after the cell that supports it [C5, page 197]. For the topological sorting to work we need to assume that no static cycles are present in the support graph [CORE, page 71]. The sorting will save some evaluation time since each cell can safely assume that its dependencies have been updated. Figure 3.13 gives an example that arises when B2 in Figure 3.12 is changed. Both B7 and C2 is

	A	B	C
1		Expenses	% of sum
2	Food	100	=B2/$B$7
3	Car	2000	=B3/$B$7
4	Hardware	500	=B4/$B$7
5	Software	1000	=B5/$B$7
6			
7	Sum	=SUM(B2:B5)	
8			

Figure 3.12 – The dotted lines illustrates supported cells

supported by B2 and C2 is also supported by B7. The arrows on Figure 3.13 display the supported cells and thereby also the order of re-evaluation. So a cell should be re-evaluated before any cells that it contains arrows to. This is exactly the achievement of doing the re-evaluation in topological order as seen on Figure 3.14.

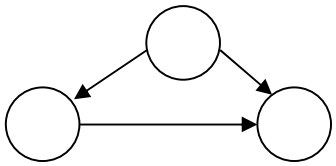


Figure 3.13: Cell dependency

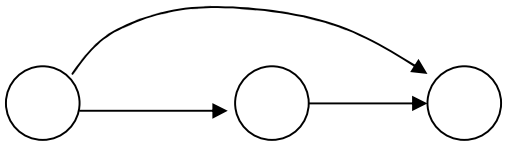


Figure 3.14: Cell dependency topological sorted

Topological sort can be done in linear time. An algorithm to do this is presented below and is taken from [C5], page 197, where L_{sort} is the result of the topological sort:

- For each initial changed cell c_i in the list of all changed cells C_i (including cells containing volatile functions)
  - If c_i is not present in the list L_{sort}:
    - AddCell(c_i)
- Revert the list

where AddCell() is a recursive algorithm:

- For each directly supported cell c_{child} from c_i children
  - If c_{child} is not present in the list L_{sort}:
    - AddCell(c_{child})
- Insert c_{child} as the last item in the list L_{sort}

Since the initial cells c_i and all descendants c_{child} do the same amount of work, the running time of the algorithm can be described as $O(C_i + C_{descendants})$. $C_{descendants}$ is simply all cells which C_i supports directly or indirectly.

For better understanding we give an example of the topological sort algorithm in Figure 3.15

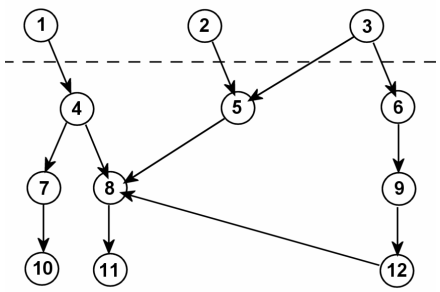


Figure 3.15: Example of re-evaluation order, where cells 1-3 are the initial changed cells and all cells below are their dependencies.

The cells numbered 1 through 3 are the initial cells c_i that are changed, which includes possible volatile functions. Cell 1 is not in the empty list L_{sort} so we call AddCell(c_i) which would recursively add all cells all the way down to cell 10 using the AddCell function (if we handle cell 7 before cell 8, ordering does not matter). We then go back to cell 4 to handle the rest of its children. The result is that 10 and 7 are recursively added as the last item to L_{sort} which now is {10, 7}. From cell 4 we traverse down to cell 11 and then back to cell 4, which is added as no more children of cell 4 exists. L_{sort} is now {10, 7, 11, 8, 4, 1} and we are done with cell 1 and its children. We traverse cell 2 which only adds cell 5 to the list since the cell 8 is already present in the list. L_{sort} is now simply {10, 7, 11, 8, 4, 1, 5, 2}. Finally we traverse cell 3 which adds cells 12, 9, 6 and 3 to the list, but not 5 or 8 since they are already present. The final list L_{sort} is {10, 7, 11, 8, 4, 1, 5, 2, 12, 9, 6, 3} which now just need to be reverted to {3, 6, 9, 12, 2, 5, 1, 4, 8, 11, 7, 10} to give one possible valid re-evaluation order out of many.

Late in the project we encountered static cycles in some sheets, therefore we use a slightly modified version of the topological sorting above as described in [C5, page 199]. The main difference is that the previous topological sort adds a cell after adding its dependencies while this modified topological sorting adds a cell before adding its dependencies. This results in that the topological sorting can terminate when a static cycle is encountered as opposed to entering an infinite loop, but there will be at least one cell in the topological sorting which does not appear strictly after all its dependencies [C5, page 199].

Notice that with this implementation static cycles are not allowed in the spreadsheet as they can result in an incorrect recalculation order. This is normally not the case with spreadsheet applications as Excel or OpenOffice Calc, which does allow static cycles but not dynamic cycles. During the project only few sheets where encountered with static cycles, and of those none had many, so this is generally not considered a large problem, although it would be relevant to check for static cycles and inform the user of them.

3.3.1 Possible extensions to the recalculation order

The running time of the topological sort is $O(C_i + C_{descendants})$. When $C_i + C_{descendants}$ is relatively large compared to the total number of cells containing formulas C_f the running time is close to $O(C_f)$. In this time it is possible to re-evaluate all cells in the sheet, even though they do not depend on the changed cells C_i. One way is to reuse the topological sorting between separate recalculations. This is problematic since a slight change in the

support graph can result in a large change in the topological sort. Furthermore existing algorithms for changing topological sorts are slow [CORE, page 51].

Another way is simply to force a full recalculation, where all cells are re-evaluated, without the use of the support graph if $C_i + C_{descendants}$ is large. An estimate of how many cells would be included in the recalculation would be necessary.

3.4 Modifying FAP grid lists

A number of operations on both FAP grid list and FAP lists are required in order to maintain and build the support graph.

3.4.1 Structure of FAP list

When adding or changing FAP grids in a FAP grid list it might necessary to compare FAP lists to see if they are equal. This can either be done naively by checking that all the numbers one FAP list represents are equal to those of another. However it is much easier if one FAP list *FL* contains the same FAP sets as another FAP list *FL'*, only when they represent exactly the same numbers. To maintain this condition two rules for the FAP sets contained in a FAP list is made. A number $n \in FL$, when *n* is represented by the FAP list *FL*:

1. All numbers in *FL* must be unique. That is for all $n, m \in FL$ it must be true that $n \neq m$.
2. A number $m \in FL$ should always be in a FAP set with it closest neighbour *n* (distance *d*), where *n* has no possibility of being in a FAP set with $b < d$ (and more than one element):

   An example would be if the FAP list contains FAP sets: (0, 3, 2)(4, 3, 3), then this should be altered to: (0, 1, 1)(3, 1, 2)(7, 3, 2)

From the above rules a few things are worth noticing:

- A number *n* whose nearest neighbour is *m* with distance *d*, will always be in the same FAP set as *m* if the other neighbour *p* of *m* has distance *e* to *m* where $e > d$.
- Two numbers with distance of 1 will always be in the same FAP set since no smaller distance is possible.
- Two FAP sets may not overlap. If two FAP sets did overlap this would mean that two numbers represented by the FAP list are not together in a FAP set with their nearest neighbour although it would be possible, which is a clear violation of the second rule. An example is that the FAP set (0,2,3) may not coexist in the same FAP list as (3,2,2), but should be altered to sets that do have lower *b* value namely: (0, 1, 1) and (2, 1, 4)
- If a number $m \in FL$, then if it has two neighbours *k* and *l* of equal distance *s*, then if neither *k* nor *l* have any neighbours of distance *d*<*s*, then they should all be in the same FAP set.

The outcome of the rules is that it is deterministic for each number $n \in FL$, which FAP set it should belong to. Therefore FAP lists will have identical FAP sets only when the numbers they represent are the same.

Example 3.1:
If the following numbers are to be represented by a FAP list:

$$\{1,2,4,6,8,10,11,12,14\}$$

Then we know for sure that the numbers with distance of one to the nearest neighbour must be in a FAP set with it, since no closer neighbour can exist. Therefore the following numbers are converted to FAP sets:

$$\{1,2\} \rightarrow (1,1,2)$$

$$\{10,11,12\} \rightarrow (10,1,3)$$

This leaves:

$$\{4,6,8,14\}$$

Clearly the lowest distance to a neighbour is now two so the conversion will be:

$$\{4,6,8\} \rightarrow (4,2,3)$$

Leaving $\{14\}$, which will be in a FAP set for itself. The resulting FAP list will therefore be:

$$\{1,2,4,6,8,10,11,12,14\} \rightarrow (0,1,2)(4,2,3)(10,1,3)(14,1,1)$$

Since FAP sets may not overlap, they can be sorted by their start position, given by their *a* value. Indeed this is used to define that a FAP set *fs* within a FAP list *FL* have neighbours given by the FAP sets with nearest *a* value before and after the *a* value of *fs*.

3.4.2 Modifying FAP list

A FAP list *FL* consists of FAP sets where each FAP set $fs \in FL$. When referring to the numbers *a*,*b* and *k* it is implicit the three parameters of the current FAP set. The last number represented in a FAP set l_{fs} is calculated as $l_{fs} = a + b \cdot (k-1)$, the first number represented in a FAP set is of course *a*.

Remove interval:
When removing an interval *I* from a starting number *s* to an ending number *e* (both inclusive), all numbers in the FAP list in between *s* and *e* should be removed.
The procedure is:

- For each FAP set $fs \in FL$
  - Check if *fs* cross the interval. If not no changes are needed to *fs*
  - Else if $a \geq s$ and $l_{fs} \leq e$, then *I* completely covers *fs*, so *fs* should be completely removed from *FL*.
  - Else the removing of *I* will result in one or two new FAP sets, one set fs_b before *I* and one set fs_a after.
    - If $s > a$ then fs_b will be: $(a, b, \frac{s-a}{b})$.
    - If $l_{fs} > e$ then fs_a will be: $(ha, b, \frac{(l_{fs} - ha)}{b} + 1)$, where $ha = (((e+1) - a) \bmod b) + (e+1)$

The general idea when calculating fs_a is finding the first valid number in *fs* after *e*.

When removing an interval it is sometimes needed to find the FAP set fs_r that is the part of *fs* that was removed. This is calculated as $fs_r : (ra, b, rk)$, where

$ra = \max(a, a + b \cdot (\frac{s-a}{b} - 1))$ and $rk = \frac{\min(ha, k) - ra}{b}$.

Adding a single number:
When adding a single number *n* to the FAP list *FL*, the procedure is:

- If *n* is already represented by *FL* do nothing.
- If *n* does not lie within the start and end of any of the FAP sets in *FL*: (1.1)
    - If there exist a FAP set *fsb* before *n* and a FAP set *fsa* after *n*, then the neighbours of *n* is l_{fsb} and the *a* value of *fsa* hereby referenced as *c*.
    - According to the rules for FAP lists *n* should if possible be combined with its nearest neighbour that is not in a FAP set with a *b* value lower than the distance to it (thus more than one item). This might mean taking l_{fsb} from *fsb*, *c* from *fsa* or both, and combining them to a new FAP set. If the distance to the nearest possible neighbour *nn* is equal to the *b* value of the FAP set *nn* belongs to, then *n* should just be added to this set. If both neighbours is at same distance *d* and both belongs to FAP sets that have same *b* value namely *d* then both FAP sets and *n* should be combined to a new FAP set.
- Else if *n* does lie within the start and end of a FAP set *fs*, split *fs* into two sets around *n* and insert *n* as in (1.1)

Adding a FAP set:
Adding a FAP set *fs* to the FAP list *FL* is rather simple if *fs* does not overlap with any of the existing FAP sets in *FL*. The only issue might be with the FAP sets that are future neighbours to *fs*. This is handled in the same way as adding a subset of *fs* to *FL* first, namely that without the start and end value of *fs* (without *a* and l_{fs}), and then adding *a* and l_{fs} as single numbers as described above.

If *fs* do overlap with other FAP sets the numbers that *fs* represent is simply added one by one. Since this obviously results in a lot of calculation whenever overlap happens, investigations have been made to check if this would cause any decrease in performance. This has not been the case, so therefore no further effort was put into improving the operation.

Add lines:
Adding lines is used when new rows or columns is inserted into a sheet. This is defined as *N* lines being added before line *R*. For all numbers $n \in FL$ and $n \geq R$ this means that they have to be moved:

$$n \rightarrow n + R$$

This is done for a FAP set *fs* follows:

- If $l_{fs} < R$ no changes are required
- If $a \geq R$ a new *a* value must be assigned: *N* added to *a*
- If $a \leq R$ and $l_{fs} > R$ the FAP set has to be split in two:

1. One before *R*: (a,b,nk), where $nk = (R-a-b-1)/b$
2. One after *R*: $(a+nk\cdot b+N,b,k-nk)$

These two sets must be added like in "Adding a FAP set" above.

Remove lines:
Deleting *N* lines from and including line *R*. Is the opposite operation to "Add lines"

This is done for a FAP set *fs* follows:

- If $l_{fs} < R$ no changes are required
- If $a \geq R+N$ a new *a* value must be assigned: *a* minus *N*
- Else calculate the two numbers:

  $k_1 = (R-a+b-1)/b$ and $k_2 = (R+N-a+b-1)/b$

  - If $1 \leq k_1$ then a part of *fs* is before the deleted lines, namely: (a,b,k_1)
  - If $k_2 < k_1$ then a part of *fs* is after the deleted lines, namely: $(a+b\cdot k_2-N,b,k-k_2)$

  That or these set(s) if any must be added like in "Adding a FAP set" above.

3.4.3 Modifying FAP grid list

The required operations are:

- Add a FAP grid:

  We want a compact representation, so when adding a FAP grid to the existing FAP grid list, we need to check if it could be combined with one or several other FAP grid(s) to decrease the number of FAP grids in the FAP grid list.

- Delete area:

  Deletes an area from the cell addresses represented by the FAP grids.

- Add rows or columns to a sheet:

  Adjusts the FAP grids to the new row/column numbers that the adding has resulted in, thus breaking those FAP grids up that lies around the new lines

- Delete rows or columns from the sheet:

  Adjusts the FAP grids to the new row/column numbers that the removing has resulted in, thus altering those FAP grids up that represented cell addresses that where removed.

In the following assume that the current FAP grid list *F* consist of a list of FAP grids, each grid $f \in F$. Each FAP grid *f* consists of two FAP lists, one for the column part lc_f and one for the row part lr_f. A FAP list is considered equal to another FAP list only when the numbers they represent are the same.

The operations are carried out as follows:
Add a FAP Grid:
When adding a FAP grid *f*':

- Go through all the existing FAP Grids.
  - If a FAP grid *f* consist of a column and row FAP list equal to those of *f'*, *f'* is discarded as *F* already represent the cell addresses of *f'*.
  - If no FAP grids have a equal column or row FAP list to those of *f'*, f_n is added to *F*.
  - If a FAP grid *f* consist of a column FAP list equal to $lc_{f'}$, then the FAP sets of $lc_{f'}$ is added to lc_f[7], and the resulting in a change to *f* resulting in a new FAP grid *f''*. Therefore *f* is removed from *F* and *f''* is added in the same way as *f'* was (using this method recursively).
  - Handling a FAP grid *f* consist of a row FAP list equal to $lr_{f'}$, is completely similar to when the columns are equal.

Delete an area

When deleting an area from a FAP grid it might result in the FAP grid being broken up into several pieces.

Example 3.2:

In Figure 3.16 two figures are shown. The first represent the FAP grid list: (0, 1, 4) x (1, 1, 5) with the area from C1R3 to C2R4 about to be deleted. The next figure represents the FAP grid list after the deletion. Here several FAP grids are needed. The FAP grid list after deletion is[8]:

$$((0,1,1)(3,1,1)\times(1,1,5))\cup((1,1,2)\times(1,1,2)(5,1,1))$$

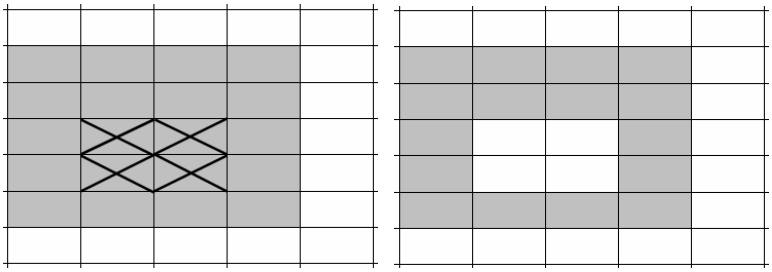


Figure 3.16: The grey areas represent the FAP grid list, and the black X's the cells about to be deleted

The procedure for deleting the area A is:

- For each FAP Grid $f \in F$ where A crosses lc_f and lr_f:
  - If A completely covers lc_f and lr_f then just remove *f* from *F*.
  - Split lc_f into two FAP lists. One FAP list lc_{fn} that describes the columns outside A and one FAP list lc_{fd} that describes the column that is part of A.
  - Also from lr_f calculate the FAP list lr_{fn} that was outside A
  - Create up to two FAP Grids:
    - If lc_{fn} is not empty create f_1 consisting of lc_{fn} and lr_f
    - If lr_{fn} is not empty create f_2 consisting of lc_{fd} and lr_{fn}
  - Add the created FAP grids to *F* using the add operation.

[7] Done recursively using the technique from section 3.4.2, under "Adding a FAP set"

[8] Other FAP grid lists are possible, this is the one used in the algorithm

Add rows or column to a sheet:
When inserting N new rows before a row R:

- For each FAP Grid $f \in F$ where lc_f represent at least one number $n > R$
  - Insert range into lc_f[9]

Inserting columns is similar.
Delete rows or columns from a sheet:
When deleting N rows after a row R:

- For each FAP Grid $f \in F$ where lc_f represent at least one number $n > R$
  - Remove range from lc_f[10]

Deleting columns is similar.

3.5 Building the support graph

Building the support graph is necessary when we have an existing spreadsheet without one. This could be when we create a spreadsheet within another application and load it into our spreadsheet program. We build the support graph in three stages, as proposed by [CORE, page 61]:

1. Build an expression occurrence map. Expressions are the only entity that can refer to other cells. They can be shared between cells in the applications as virtual instances as shown in Figure 3.5.
2. Find area of cells that are referred by the expressions using the occurrence map.
3. The last stage builds the support graph using the referred cells.

The end result of the algorithm is that each cell *ca* has a list of FAP grids which represents the cells it directly supports.

3.5.1 Building the expression occurrence map

This method of building the occurrence map should be reasonably compact when using typical spreadsheets like the one shown in section 3.2.3 [CORE, page 61]. The result is that each column has a mapping of expressions *e* to a sequence m(*e*) of FAP sets representing the occurrences of the expressions in that column, where m is a map. We use the C0R0 reference format when explaining the algorithm below:

- Scan each sheet in the workbook.
- Scan columns in the sheet left to right, starting with the column 0.
- In each column, scan the rows from top to bottom looking for expressions *e*, starting with row 0.
- If an expression e is found, handle it the following way:
  - If f is not present in the map m, then add sequence m(*e*) = (r, 1, 1), where r is the current row.
  - Otherwise assume the last entry in the map m for e is (r', b', k'). We proceed as follows:

[9] Described in section 3.4.2.
[10] Described in section 3.4.2.

- If k' = 1 then the FAP set only represents a single occurrence of e and we can merge the current occurrence by setting k = 2 and b = r – r'. Now the latest entry in m(e) is updated to (r', (r – r'), 2).
- Otherwise if r = r' + b'k' then the occurrence e is the next member of the previous FAP set and the latest entry in m(*e*) is updated to (r', b', k'+1) (only the k value is adjusted)
- Otherwise we add the new last item in m(*e*) which is set to be (r, 1, 1) since we cannot extend the previous FAP set.

One issue with this algorithm is that if the same expression appears with different interval then a lot of FAP sets for the same expression will be made. An example is shown in Figure 3.17, where the same expression is present in cells with alternating interval. The resulting FAP sets in this instance would be:

$$\{(0,1,2),(3,1,2),(6,1,2),(9,1,2)\}$$

However this is not very often seen in spreadsheets as this would require the user to do quite some manual work since it cannot be easily constructed using simple copy operations. However, using macros, external spreadsheet generation or other methods of automatic formula copying, this could happen.

	A
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	

Figure 3.17: The grey cells represent cells with the same expression present

Notice that the algorithm works in one dimension, the next overall step in building the support graph extends this into two dimensions described in section 3.5.2.

The performance of building the expression map and the next step in the algorithm can suffer depending whether the columns or rows are scanned in the first step of the algorithm. This is because of a FAP set explosion as shown in Figure 3.18 and Figure 3.19.

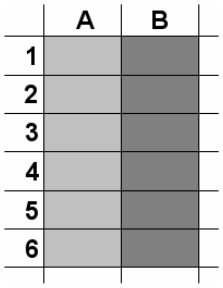


Figure 3.18: C0R0:C0R5 and C1R0:C1R5 both contain shareable expressions, but expressions are not sharable between the two columns A and B.

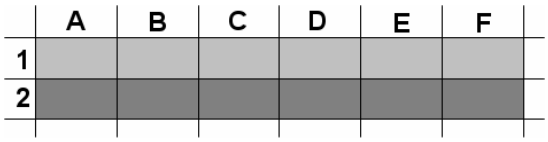


Figure 3.19: C0R0:C5R0 and C0R1:C5R1 both contain shareable expressions, but expressions are not sharable between the two rows 1 and 2.

If we scan columns for expressions, Figure 3.18 ends up with two FAP sets in the occurrence, on for each column. Figure 3.19 will get two FAP sets per columns (since the expressions are only sharable row wise), a total of 2x6 = 12 FAP sets in the occurrence map. The opposite is true when scanning rows for expressions. For this reason we modified the algorithm so it can scan both columns and rows, but for most part scanning columns would gain the fewest FAP sets in the occurrence map. This is because the structure of normal spreadsheets has more shareable expressions vertically than horizontally from our own observations. This observation may be explained by the fact that spreadsheet applications typically imposed a restriction in the number of rows and columns in any given spreadsheet. For instance Excel 2003 imposed a limit of 256 columns and roughly 65.000 rows [EXP]. The number of columns was far more restricted than the number of rows which could explain the tendency to copy vertically rather than horizontally.

Example 3.3: corresponding FAP sets[11]
In column 1 (B) we have the expression "SUM(C2R[0]:C[1]R10[12])" in the following row numbers: (1, 2, 4, 6). This leads to the FAP sets: (1,1,2) and (4, 2, 2).

3.5.2 Finding referred cells from expressions

The cells referred from an expression correspond to the cells that directly support the expression. This part of the algorithm finds these referred cells for each expression occurrence. The algorithm can scan columns or rows just as the previous one. We only show it for column scanning, but it is trivial to scan the rows. A columns position in the spreadsheet is *mc* when representing m(*e*). The algorithm follows:

- Scan each sheet in the workbook.
- Scan columns in the sheet left to right, starting with the column A.
    - Create a FAP set (*mc*, 1, 1) representing the column dimension.
    - Get the FAP set from the expression map that corresponds to the rows.

[11] This example is based on column scanning
[12] This can not be represented in A1 format since the row number is not defined - for cell B1 the corresponding expression would be "SUM($C1:C$10)" and for B2 "SUM ($C2:C$10)".

- Scan each expression e occurrence in that particular column from the occurrence map m(e)
  - Extract the cell area reference *A* (e.g. A1:$B$5 or A1) from the expression
  - Calculate the absolute area CA from the cell area reference. We convert relative references to absolute for the two corner cell references describing a cell area *RA* and all cells within this area are the referenced cells. For each dimension of the area *RA* the start (upper left corner) is the start of area *A* if it is absolute, if it is relative it is the start of area *A* added with the start of the FAP set for that dimension. The end of a dimension of the *RA* area is equivalent just with the end of *A* and the end of the FAP set for the dimension.

Example 3.4: calculating cell area for referenced cells[13]
For column 1 (B) we have the FAP set (4, 2, 2) and the cell reference *cr* (C2R[0]:C[1]R10) and wish to calculate the cell area *ca* that are referenced. The start column of *cr* is absolute so this is equal to the start column of *ca*. The end column of *cr* is relative so the start column of *ca* must be the end column of *cr* added to the current column, which is 1. The start row of *cr* is relative so this needs to be added to the start of the FAP set, which is 4 (corresponding to the *a* value of the FAP set). The end row of *cr* is absolute and therefore equal to the end row of *ca*.

The end result for *ca* is: $ca = (C2R4 : C2R10)$

Notice how we use a simpler approach to finding all cells in a given cell reference than the one described in [CORE, page 66]. We convert relative references to absolute for the two corner cell references describing an area and all cells within this area are the referenced cells. This seems to save us much calculation since we only perform simple calculation on the two opposite corners rather than explicitly calculate each cell in the area. The only disadvantage is that some area references with relative end and start point, might have unreferenced cells in the area. In order to deal with this there is made a check in last step of building the support graph from referred cells, so these cells will not be included anyway.

The column FAP set is created so we can work in two dimensions in the next step of the overall algorithm. The row FAP sets is stored within m(*e*) which was calculated when creating the expression occurrence map. We now have the cell references which support expression instances on a per column basis, but with enough information to work in two dimensions in the next step.

3.5.3 Building the support graph from referred cells

This last step calculates for each referred cell *c* in the cell area CA[14] the FAP grid S_{ca} added as a result of the current expression *e*. The row part of S_{ca} is a subset (a', b', k') of the FAP

[13] This example is based on column scanning
[14] That was found in step 2

sets (a, b, k) from the expression map for the specific column m(*e*)[15] numbered *mc* representing the occurrence of the expression in this column. The main issue is to calculate the correct *a* and *k* value for the row part of S_{ca} when the cell CA has either a relative row start, relative row end or both.

Example 3.5: A simple example to explain the general idea[16]
A entry in the expression map for column 2 has the FAP set *rf* (2,2,5) and an expression containing the reference (C3R0:C3R10). Since this reference is absolute, the cell area CA is equivalent to it and also the FAP set *rf* represents exactly the rows of the supported cells for all of the referenced cells. Since *rf* is from the 2nd column we construct the FAP set *cf* (2, 1, 1) representing that column and use it together with *rf* to construct a FAP grid. This FAP grid will then represent the supported cells for all the cells in CA for this expression.

This step is an inner loop of the previous algorithm, provided separately for simplicity. The algorithm always works with cell area references, where single cell references are represented as cell area references (e.g. C0R0:C0R0).

- Scan each cell area reference from before
- Scan each FAP set (a, b, k) in m(e) (represents the occurrences of the expression e row wise in this column and therefore the occurrences of the cell area references)
- Overall three different cases are applicable depending on the four different combinations of absolute and relative rows from the opposite corners of the cell area reference. The current row is represented as r'.

    1. If the cell area reference contains both row absolute corners ($C0Ri_1$:$C0Ri_2$)
        - Since both corners are absolute the current expression *e* always references all cells within the cell area, therefore S_{ca} = (a, b, k) x (*mc*, 1, 1).

    2. If the cell area reference contains one row relative and one row absolute corner ($C0Ri_1$:$C0R[i_2]$ or $C0R[i_1]$:$C0Ri_2$) :
        - This case can be subdivided into three cases. We will only show it for when the first row i_1 is absolute. When i_1 is the relative row simply switch i_1 and i_2 in the description below
            - If the current row r' = i_1
                - S_{ca} = (a, b, k) x (*mc*, 1, 1)
            - If the current row r' < i_1
                - Find the highest k_1 value so that $i_2 + a + b(k_1 - 1) \leq r'$.

                  If $k_1 \neq 0$ then S_{ca} = (a, b, k_1) x (*mc*, 1, 1)

[15] That was found in step 1
[16] This example is based on column scanning

- If the current row r' > i_1
  - Find the lowest k_1 value so that $i_2 + a + bk_1 \geq r'$. If $k_1 \neq k$ then $S_{ca} = (a + bk_1, b, k - k_1)$ x (*mc*, 1, 1)

3. If both corners of a cell area reference are absolute (C0R[i_1]:C0R[i_2]) and $i_1 \leq i_2$
   - Find the highest k_1 so that $i_2 + a + b(k_1 - 1) \leq r'$ and lowest k_2 so that $i_1 + a + bk_1 \geq r'$.
   - Define $k_1' = \max(0, k_1)$ and $k_2' = \max(k, k_2)$ then S_{ca} can be described as:
     - $S_{ca} = (a + bk_1', b, k_2' - k_1')$ x (*mc*, 1, 1)

- The adding of for S_{ca} to the FAP grid list S_c of the current cell address, where $S_{ca} = (a_r, b_r, k_r)$ x (a_r, 1, 1) is rather simple:
  - If $S_c = \varnothing$
    - Add S_{ca} to the empty set S_c.
  - Else if (a_r, b_r, k_r) matches any row dimension in the set S_c then union (a_r, 1, 1) with the corresponding column dimensions in S_c according to the rules described in chapter 3.4.3.

If in step 2, when finding the referred cells from expressions, several cell areas are found for the same expression, some preventive measures have to be made to avoid overlaps of FAP grids for the cell *c* where the FAP grid S_{ca} is added in the last step. Since the column part of S_{ca} is just a number this can be compared to the other FAP grids to be added. Thereby the cell address of cell *c* and the current column number of the FAP grid S_{ca} to be added distinguish those row FAP sets, described as a FAP list *fl* that may overlap. Therefore a map *m* from the column number and the current cell to the added row FAP sets for the current cell is made. So before S_{ca} is added to S_c a lookup in *m* is performed and the relevant FAP list *fl* found. The row FAP set *fs* of S_{ca} is then added to *fl* and during that process duplicates are removed from *fs*.

Volatile functions are found during the first step of finding referred cells from expressions and put into the separate list of cells that have to be evaluated on every change.

3.5.4 Summary of building the support graph

For better understanding the three steps of building the support graph is given here again, but with details from the last chapters. It is assumed that column scanning is used:

1. For each column C build an expression map that from a expression *e* refers to a FAP set *fs* representing the rows where the expression is present.
2. Go through the expression map, column by column. For each column go through each expression *e*, and calculate the area CA wherein *e* has references to.

3. For each cell *c* in CA calculate the FAP grid that represent those cells containing the expression *e* in the column C that have a reference to *c*. This is calculated on basis on the FAP set *f*s.

If a expression contains several reference areas, e.g. "SUM(A1:B2)+SUM(G2:H2)", several areas are found in step two. Step 3 will be done once for each such area.

3.6 Maintaining the support graph

This section will explain how to modify the support graph to reflect changes in the spreadsheet. The modifications we want to be able to do are:

- Delete the content of an area (including deleting the content of a cell)
- Create formula in a single cell
- Copy formula to an area (including copying to a single cell)
- Move formula area to another area (including moving a single cell)

3.6.1 Modifying the support graph

We define a rectangular area A of cells, starting at cell address *ca*, having *r* rows and *c* columns. This area A directly references a set *T* of cells[17]. Also S_a is the set of absolute cell addresses directly supported by a single cell at address *a*. Finally a cell address *ca*, consist of a column component ca_c and a row component ca_r. Notice that S_a is another representation for the FAP grid list introduced in section 3.2.4.

This section follows [CORE], page 59.

- Delete the content of all cells in cell area A
  - Find the set *T* of absolute cell addresses that the formulas in A have references to
  - Remove the area A from S_a for each cell address *a* in *T*.
  - If a formula f in a cell at address ca in A contains at least a single volatile function, remove ca from the global list of cells that contains volatile functions.

- Create formula *f* at cell address *ca*
  - If *ca* contains an existing formula *f'*, delete it from the support graph (see "Delete formula" section).
  - For each unique cell reference *a* in *f* add the FAP grid (c_{ca}, 1, 1) x (r_{ca}, 1, 1) (representing *ca*) to S_a.
  - If the formula *f* contains at least a single volatile function, add *ca* to the global list of cells that contains volatile functions.

- Copy formula *f* at cell address *b* to a rectangular area A
  - Delete all the cells in A (as described above)
  - The area A can be described by the FAP grid (ca_c, 1, *c*) x (ca_r, 1, *r*). For each cell address *a* in *T* a subset of this FAP grid must be added to S_a. If *f*

[17] And since *T* is a set no duplicates can be present

contains multiple references to the same cell, e.g. "=SUM(C0R0:C5R5)+C0R0", several subsets, one per reference should be added. This or these subsets are calculated as when building the support graph, which is described in chapter 3.1.
    - If the formula f contains at least a single volatile function, add the cell addresses defined by the FAP grid (c_{ul}, 1, columns) x (r_{ul}, 1, rows) to the global list of cells that contains volatile functions.

- Move formula *f* at cell address ca_1 to ca_2
    - Create formula *f* at cell address ca_2
    - Delete formula *f* at cell address ca_1

- Insert rows (insertion of columns is a trivial change)
    - When inserting N new rows before a row R the following applies
        - Each S_a that contains a row r > R must be adjusted. (so all S_a must be checked)
        - Each formula that contains an area reference *ar* that did cover the row R and R-1 before the insertion of rows will now also cover all of the N inserted rows for all the columns in between the start and end column of *ar*. To handle this, elements of building the support graph which is described in chapter 3.1 is used. Firstly an expression map is made for each expression in a row that has reference areas covering at least the row R and R-1[18]. This expression map[19] is then used in the same way as in building the support graph to calculate FAP grid lists for the new rows.

- Deleting rows (deleting columns is a trivial change)
    - When deleting N rows after a row R the following applies:
    - Each S_a that contain a row r > R must be adjusted (so all S_a must be checked)

With these operations it is now possible to make standard changes to a spreadsheet and still maintain a correct support graph.

During these operations no overlap of FAP grids are introduced in the FAP grid lists. The potential operations where an overlap could be introduced would be the creation or copy of a formula. In creation of formula only unique addresses are used in the creation of a formula are used and the deletion removed any previous support to the cell. During copy the same operation as when building the support graph is used, where measurements to avoid overlaps are made.

[18] All formulas that contain the expression will cover the inserted rows even if they are relative since the expression map is made on row basis
[19] See chapter 3.4.1 for an explanation of expression map

3.7 Changes from Mark I to Mark II

The change from mark I to mark II is the reuse of FAP grid lists between cells and relative FAP lists.

A lot of references in a spreadsheet are often either to the same absolute cell or to the same relative area, e.g. a sum function that for each column takes the four previous values and sums them. Therefore sharing of FAP grid lists between cells could be an advantage. To enable this fully however a cell must be able to have means of making relative dependencies. This is done on the FAP list level, so that both dimensions can be either absolute or relative independently thus still enabling the FAP sets in the FAP list to still be sorted[20], since they relatively still would have the same position. In Figure 3.20 an example of how the support graph of Figure 3.10 from chapter 3.2.4 would look with relative references. A FAP list that is relative starts with a *R*. Clearly the FAP grid list of B2 to B6 can be shared, thereby only using two FAP grid lists instead of six.

	A	B	C	D
1				
2	(2, 1, 2)x(1, 1, 5)	(3,1,2)x *R* (0,1,1)		
3		(3,1,2)x *R* (0,1,1)		
4		(3,1,2)x *R* (0,1,1)		
5		(3,1,2)x *R* (0,1,1)		
6		(3,1,2)x *R* (0,1,1)		

Figure 3.20: The support graph of Figure 3.10 with relative support graph

By allowing FAP list to be relative however also allows for overlapping FAP grids. However this is limited to a factor of four, since there can be only two overlapping FAP lists for each dimension (2 dimensions x 2 states (relative or absolute)). Furthermore it might sometimes be an advantage to have overlapping FAP grids. An example of this is shown in Figure 3.21. Although the FAP grids in B1 have two references to A1 it allows B1 to share the FAP grid list with the rest of the column. Since the row component of the first FAP grid is relative the FAP grid list of B2 only references A1 once.

[20] FAP sets can be sorted on *a* values, see chapter 3.4.1 for further information

	A	B
1		(0,1,1)x R (0,1,3) ∪ (0,1,1)x(0,1,2)
2		(0,1,1)x R (0,1,3) ∪ (0,1,1)x(0,1,2)
3		(0,1,1)x R (0,1,3) ∪ (0,1,1)x(0,1,2)
4		(0,1,1)x R (0,1,3) ∪ (0,1,1)x(0,1,2)
5		(0,1,1)x R (0,1,3) ∪ (0,1,1)x(0,1,2)
6		(0,1,1)x R (0,1,3) ∪ (0,1,1)x(0,1,2)

Figure 3.21: Allowing overlapping FAP grids may allow for reuse of FAP grid lists

Some changes are of course needed in maintaining and building the support graph in order for the sharing to work. These changes are however fairly simple.

3.7.1 Changes to building the support graph

When building the support graph the only place a change is needed is when building the support graph from referred cells as described in chapter 3.5.3. The general idea is to keep the relativeness of the address references that the support graph is build upon over to the FAP grid to be added to those referenced cells.

If the cell are reference contains relative end and start for the column, the column part of S_{column} should be relative, which is done by subtracting the column number of the cell to be inserted in from *mc* of the *a* value of the FAP set.

Regarding the row part of S_{column} the absoluteness depends on the case of the cell area reference. When dealing with case 1 the row part of S_{column} should be absolute and when dealing with case 3 the row part should be relative, just as the address is. In case 2 the row part is set as absolute.

Notice that area references can have a relative start and a absolute end on each dimension, which FAP sets can not. Therefore FAP grids that are based on such area references are rarely the same, which causes the owning FAP grid list differs from others, and is thereby un-shareable.

The union of S_{ca} and S_{column} in the last step of building the support graph can often be avoided. This is done simply by holding a map from all S_{ca} and S_{column} that are to be unioned to the resulting S_{ca}'. Then instead of redoing the calculation if both S_{ca} and S_{column} are the same for two cells, then the result can just be reused, also resulting in reuse of FAP grid lists.

3.7.2 Changes to modifying the support graph

When copying a formula the method from building the support graph is used, therefore this method can be altered in same way as described in the chapter above.

Creating a formula can simply be changed to act as if a formula was copied.

When deleing the content of all cells in a area and when deleting rows or columns, the method in the FAP lists simply need to change the absolute references to a relative ones, which is simply done by subtracting the current row / column number from the absolute reference. This is also the case with insertion of rows. Though insertion of rows might add some new FAP grid lists, this is done in the same way as when building the support graph, so this can also just be changed as in the above chapter.

The downside of changing the FAP grid list of a cell is that a copy has to be made before modifying it.

3.7.3 Future extension to Mark II

The shortage of Mark II is mainly when dealing with area references that have an absolute start and a relative end or opposite (relative start, absolute end) in one dimension, e.g. A$1:A5 or B1:$X15. When copying a cell or building the support graph the FAP grid list can be reused, but only if the area reference it was built on had either both absolute start and end, or both relative start and end in the dimension the expression map is built on, hence if columns are scanned then row start and end should have equal relativeness. This is because FAP sets can only be either absolute or relative for both start and end, as a result of their construction.

So if FAP sets where changed, so instead of having a *k* value that determines how many numbers are present in the sequence another value *e* could be used that determine the end of the FAP set. This enables us to make the start value *a* and the end value *e* of a FAP set independent in terms of relativeness.

This would not only make it possible to reuse more FAP grid lists between cells but would actually make it easier to build the support graph since no calculation for *k* value for FAP sets has to be done. Instead the *e* value is just as easy to calculate as the *a* value. Note that when the start of a reference is relative this means that the end of a FAP set needs to be relative and vice versa, e.g. if the reference C0R0:C0R[0] is present in B1 to B10, then A1 trough A10 would all support B10 but only A1 would support B1. When building the support graph, only the last step of building the support graph from referred cells would be substantially altered in the case where the start and end of the cell area CA have opposite relativeness. Let the resulting FAP set be *fs* and have values *a, b* and *e* for the row part, where *a* and *e* are both absolute and *r* is the current row. Assume column scanning is applied. Then *fs* is calculated as follows, where an *R* implies that the number is relative:

If the cell area is C0Ri_1:C0R[i_2]:

Then if r>i_1:

$$fs : (R\min(-i_2, i_1 - r), b, e)$$

If r<i_1:

$$fs : (a, b, R\min(-i_2, i_1 - r))$$

And if r=i_1:

$$fs : (a, b, e)$$

In case of C0R[i_1]:C0Ri_2 then use the above with i_1 and i_2 exchanged.

In Example 3.6 a usage of the above is shown.

Example 3.6
If there exist the expression "SUM(C0R[-5]:C0R15)" in B10 to B20, this would result in the representing FAP set *fs* : (10, 1, 20). Notice that neither *a* nor *e* is relative at this point. The area A the expression affects would be C0R5 to C0R15. For each row *r* the row FAP set fs_r would be:

$$(10, 1, R \min(15-r, 5))$$

, where R means that the number is relative. As for all rows in this example min(15-*r*, 5) = 5, hence fs_r would be: (10, 1, 5) for all rows.

3.8 Alternatives

This chapter will shortly describe alternative ways or ideas when dealing with a support graph.

3.8.1 Possible extensions to maintenance of the support graph

Simple checks could be made to prevent unnecessary maintenance to the support graph when editing a cell. The support graph only needs to be updated if the change to a cell modifies any references to other cells. This can be done by changing the references directly, e.g. SUM(A1:A2) to SUM(B1:B2). Notice that a change of IF(0.2>0.5; B1; B2) to IF(0.7>0.5; B1; B2) would not result a change to the support graph since we do not evaluate the cells when building the support graph. In particular, editing a data cell (one not containing any formula before or after editing the cell) may cause a recalculation, but can never affect the support graph.

A further step would be to cache the evaluated value of a cell *c* and compare it to the newly calculated value. As long as the value does not change we do not need to recalculate the cells supported by *c*. Functions like FLOOR(), ABS(), MONTH(), DAY() are examples of either volatile functions that rarely change value or functions that might not change value even though cells they reference to does so.

3.8.2 Possible extensions to FAP grids

By not allowing FAP grids to represent cells that a cell does not support we can quickly get an explosion of FAP sets if the cell area structure is or becomes irregular:

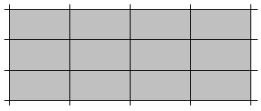


Figure 3.22, described by the two FAP sets (0,1,4) x (0,1,3)

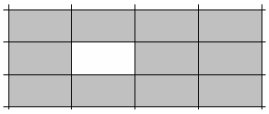


Figure 3.23, middle cell removed. A non overlapping description takes 6 FAP sets. One possible description is (0,1,4) x (0,2,2), (0,1,1) x (1,1,1), (2,1,2) x (1,1,1)

One possible solution to this problem is to explicitly define which cells are excluded from a cell area description. E.g. the above example could be described as (01,4) x (0,1,3) excluding (1,1,1) x (1,1,1) which only takes 4 FAP sets versus 6. It would save space and

time since it is faster to add two FAP sets, rather than splitting 2 existing FAP sets into 6 FAP sets. But we must take into account the overhead cost and added complexity of working with two separate sets of FAP sets when taking this approach. The solution would require the program to clean up the list of included and excluded FAP grids, e.g. in Figure 3.23 it might be an advantage to get rid of the exclude list and update the include list if more of the cells are removed. Another approach is to use set theory to construct sets using unions, intersections and complements in various combinations. Extending our current representation of FAP grids (the product of unions of FAP sets) we could describe Figure 3.23 by ((0,1,4) x (0,1,3)) \ ((1,1,1) x (1,1,1)). "\" is the relative complement and A\B is the set of all members in A but no in B.

From our own observations this type of irregular structure rarely occurs in typical and realistic spreadsheets. When it occurs it would probably be an error (e.g. accidental deletion).

Allowing overlaps between FAP grids could save space in certain cases. One of the more simple ones is described in Figure 3.24 and Figure 3.25.

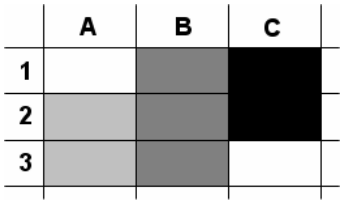


Figure 3.24: FAP grids without any overlap described by the FAP grids (0,1,1) x (1,1,2) (1,1,1) x (0,1,3) (2,1,1) x (0,1,2)

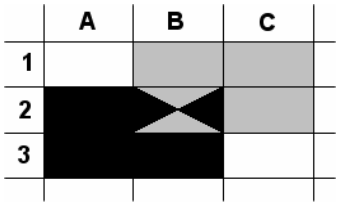


Figure 3.25: FAP grids with overlap occurring at cell B2: (0,1,2) x (1,1,2) (1,1,2) x (0,1,2)

In Figure 3.24 the 7 cells can minimally be described by three FAP grids. By allowing overlapping FAP grids we can reduce it to two FAP grids as shown in Figure 3.25. The overlap would result in unnecessary recalculation unless special action would be taken to prevent it. However this unnecessary recalculation would not lead to wrong recalculations, only longer recalculation time.

Another way to save space is to reduce the usage of FAP sets when appropriate. A FAP set (0,1,1) x (1,1,1) could simply be described by the coordinates (0,1). This would save space in an implementation of the representation. In more general terms any FAP set which has a k value of 1 can be described by a single integer instead of the 3 integers needed by a FAP set. This would however require two structures instead of one thus possibly converting between them when cells are added or removed.

3.8.3 Allowing FAP sets to overlap

In this project FAP list may not contain overlapping FAP sets, which is an implication of the 2. rule for FAP list in chapter 3.4.1. This rule was made to ensure that FAP lists always contain exactly the same FAP sets when the FAP lists represent the same numbers.

However there might be other ways to ensure this, without confining to the above mentioned rule.

The advantage of overlapping FAP sets is clearly that in some cases this might decrease the size of a FAP list considerable. E.g. the FAP list:

(0,1,2)(3,1,2)(6,1,2)(9,1,2)

Could instead be represented as:

(0,3,4)(1,3,4)

However as mentioned in chapter 3.5.1, this already becomes a problem when building the support graph so some kind of conversion of FAP sets is needed to obtain the more compact overlapping form. Also it is not always obvious when it is an advantage to try to convert FAP sets to overlapping sets or when to try to convert overlapping FAP sets into a single FAP set as seen below.

Rules on how to join overlapping FAP sets into a single set and union of overlapping FAP sets is given below.

Join overlapping FAP sets into a single set

Zip multiple

Generalization	Example
$(a,b,k) = (a,nb,k_0) \cup (a+b,nb,k_1) \cup$ $(a+2b,nb,k_2) \ldots \cup (a+(n-1)b,nb,k_{n-1})$ Where $k_i = (k-1-i+n)/n$	$(0,3,7) =$ $(0,9,3) \cup (3,9,2) \cup (6,9,2)$

Union of two overlapping FAP set:

If one needs to eliminate duplicate entries in the resulting union of the two overlapping FAP sets when the *b* values are not the same, then the zip multiple rule can be used. The basic idea is to divide the two FAP sets into smaller FAP sets with the same *b* value. Duplicate entries can now be removed and overlapping sets can be handled by using the overlap rule from the previous section.

The steps for performing a union of two overlapping FAP set is as follows:

Step	Example
Take the union of the two FAP sets	$(2,3,13) \cup (0,4,7)$
Find the lowest common divider for the two values of *b*	$lcd(3,4) = 12$
Find *n* value for each of the two FAP sets by dividing the common divider with *b*	12/3 = 4 and 12/4 = 3
Use the zip multiple rule on the two FAP sets with their *n* values	$(2,3,8) = (2,12,4) \cup (5,12,3) \cup (8,12,3) \cup (11,12,3)$ $(0,4,7) = (0,12,3) \cup (4,12,2) \cup (8,12,2)$
Union the two FAP sets	$(2,3,13) \cup (0,4,7) = (2,12,4) \cup (5,12,3) \cup (8,12,3) \cup (11,12,3) \cup (0,12,3) \cup (4,12,2) \cup (8,12,2)$
Delete duplicates and check for overlapping sets	The union of the two sets with the *a* value 8 can be represented with the set: $(8,12,3) \cup (8,12,2) = (8,12,3)$

3.8.4 Avoid unnecessary supporting

When recalculating the current support graph some over-approximation in the number of cells to be re-evaluated is made. To be in the minimum set of cells to be re-evaluated a cell *c* must obey the following two minimum criteria:

1. Dynamically[21] depend directly or indirectly on the changed cells and
2. At least one of the cells *c* directly depends on must have changed value.

An example of a violation of the first minimum criteria with the current support graph is a cell *d* containing the following: "IF(1=1;A1;B2)" where cell B2 is changed. Although the value of *d* will not change it will still be re-evaluated. The second minimum criterion would be violated if another cell *b* depended only on *d* in the above example. Then *b* would be recalculated although it would not be needed.

The "IF" formula in the above example is a classic example of a so-called non-strict function. Non-strict functions are functions which do not require all their arguments evaluated before evaluating the function itself. IF(1=1; A1; B1) only ever needs one of the two arguments A1 and B1, since both can never be chosen, in this case only A1 needs to be evaluated beforehand. Only non-strict functions lead to a difference between static and dynamic references.

A relatively easy way to make the current over-approximation smaller would be to avoid re-evaluating cells whose dependency cells have not changed, thereby obeying the second minimum criterion. This could be done by adding a number *r* in each cell that represents the number of references in that cell. During the re-evaluation cells that do not change value during an update should decrease this value by one for all cells they support. Whenever the *r* value for a cell *c* reaches zero, this cell could be ignored in the recalculation since none of its dependencies have changed value. Furthermore the *r* value for the cells *c* support should be decreased by one. However this would also introduce a

[21] See section 3.2.2 for a definition of dynamic

certain overhead making it possibly slower than re-evaluating the cell. For instance if the cell only contains a reference it would possibly be much faster just to re-evaluate it.

Another more complicated method would be to make a dynamic support graph, thereby obeying the first minimum criterion. For instance in the above example with the formula "IF(1=1; A1; B1)" in the cell *d*, B1 would not even support the cell *d*. However a dynamic support graph would also need dynamic updates, which could occur during the topological sorting. Consider the example of A1 having the value 1 and the cell *d* containing the following formula "IF(A1=1;B1;B2)". In a dynamic support graph only B1 and A1 would support *d*. If A1 is changed to the value 2 then the support graph would have to change to reflect that B1 no longer supports *d* but B2 does, which would not happen before the topological sorting is in progress. A way to solve this would be to presume that any change to any dynamic dependencies *d* would have, would require a recalculation of *d* after the last static dependency of *d*. This would avoid re-evaluating any cells to calculate new dependencies before the topological sorting is finished, although this would slightly disobey the first minimum criterion[22]. A problem still exists however, if *d* suddenly during the topological sorting depends on B2, B2 might already be in the list so *d* would have to be added before B2[23]. More precisely *d* would have to be added before the first static dependency it had. Which structure to choose for the topological list during the topological sorting in order to find the first of a number of elements is not clear.

Both methods could be combined as they each improve on different aspects, namely each of the two minimum criteria above. They would however still make a over-approximation on the number of cells to be re-evaluated since the proposed dynamical support graph does not obey the first minimum criteria.

3.8.5 Other ideas

Some other ideas to improve the support graph was thought of but not exploited. These include:

- Sharing FAP lists or/and FAP grids in addition to, or as an alternative to share FAP grid lists. Obviously there will be more FAP lists and FAP grids that can be shared than FAP grid lists, since a FAP grid list is just a collection of the two, either directly or indirectly. The main back draw is that maintenance might be more complicated.
- Trading memory for performance or opposite. Trading performance for memory can be done by over-approximation of the number of cells in the support graph like in section 3.8.2, where FAP grids are allowed to overlap, resulting in fewer FAP grids. Trading memory for performance can be done by saving the topological sorted list of a cell after recalculation, so it can be reused as long as no relevant changes are made to the support graph.
- The current construction of a FAP list is that is it just a number of unions of FAP sets. Instead other constructs could be used:

[22] As the cell *d* would have to be re-evaluated to be certain which dynamic references are in effect.
[23] Actually *d* would have to be after B2 in the final list of the topological sorting but since the last operation of the topological sort is to invert the entire list, *d* should be before B2 at the time of insertion. See section 3.3 for more information on the topological sorting.

- Products of FAP sets. A product of two FAP sets f_1 : (a_1, b_1, k_1) and f_2 : (a_2, b_2, k_2) would be as follows:

  Apply f_1 for a total of k_2 times starting from a_2 with the interval b_2+1 in between each usage.

Example:

(2, 1, 2) X (4, 6, 3): {6, 7, 16, 17, 22, 23}

- Difference of FAP sets, for instance:

  (0, 1, 10)/(3, 5, 2) : {0, 1, 2, 4, 5, 6, 7, 9, 10}

3.9 Additional details

Additional details and examples regarding building and maintaining the support graph can be found in [CORE, page 61-68].

4 Development – different versions

This section describes the different versions and therefore the part of the development process where focus was on performance (contrary to error correcting or extension of functionality). Any results between versions or subversions mentioned in this chapter are not from the benchmarks in chapter 6, but from the profiling results, which are mostly based on the two spreadsheets called ROLE and CONST1. The ROLE sheet is described in section 6.2.4, and the CONST1 is a large constructed spreadsheet for development purpose[24]. The reason for choosing these two spreadsheets is that they are both large, show memory and performance problems when handled in CoreCalc, but not to large to make profiling too time-consuming.

Several versions of a SupportCalc prototype have been developed during the project. The first functional prototype, Mark I, was primarily built on the theory from [CORE], as described in chapter 3. The later versions are made after finding performance issues that where found unacceptable and then profiling the current version to find possible bottlenecks. Solutions to eliminate or decrease the impact of a bottleneck was created and implemented to create newer versions in an iterative manner. The versions are named Mark *x*, where *x* is the version number.

Screen dumps of the profiling results and short explanation of these can be found in the appendix in section 11.3.

4.1 Profiling tool

The tool used for profiling is a beta version of ”dotTrace”. Information about the tool can be found in [JBDT]. The application can profile memory usage as well as processor usage. Memory usage is shown per class basis and processor usage per method basis.

4.2 Mark 0

Mark 0 was the first version of SupportCalc. It is based on the theory described as one of the alternative designs in section 3.8.3 , where FAP sets in a FAP list are allowed to overlap. This version has the same performance problems as Mark I, as well as having problems with union of FAP grids due to FAP lists ability to having different FAP sets although the numbers they represented are the same. Also some calculating errors existed within some of the methods used for union of overlapping FAP sets although these where not discovered until Mark II was fully developed. These errors have never been corrected in this version as the focus was on improving Mark II, therefore this version was never fully tested, though some profiling was done.

4.3 Mark I

This version differs from Mark 0 in the implementation of the rules for FAP lists. These rules are described in section 3.4.1. Although Mark I in difference to Mark 0 in some instances does a lot of calculations when making a FAP set to a FAP list, there seems to be almost no difference in performance between the two.

[24] The spreadsheets are on the attached CD-ROM.

Two performance problems existed with this and the previous version. Firstly the memory usage of SupportCalc was very high, often using at least twice the memory than without the support graph (equivalent to running CoreCalc), also the time of recalculation was high when using the support graph compared to Excel and Calc.

The primary reason and idea behind using FAP sets, hence one of the primary ideas of this project, was to decrease memory usage compared to a naive implementation of a support graph. Therefore the high memory usage was assessed as being the single most important problem.

Already during Mark 0, this was diagnosed and the main contributor to the high memory usage was the FAP lists, although a lot of these did not hold more than one FAP set. In order to avoid using FAP lists, when only one FAP set was present a dynamic internal state change was tried, as described in section 3.4.2. The performance was poor so another solution had to be found. The idea of sharing FAP grid lists between cells was invented and implemented in Mark II.

4.4 Mark II

The difference between Mark I and Mark II is the implementation of relative FAP lists and sharing of FAP grid lists between cells. This is described in section 3.7. This worked very well and even helped one of the other performance issues. The time it took to build the support graph from scratch has been significantly reduced in most cases.

The main problem is now that of recalculation. The recalculation process was profiled, which showed that at least 9/10 of the time was used in the topological sorting. This was addressed in Mark III

4.5 Mark III

In the topological sorting a collection that behaves like a HashSet and a LinkedList is used[25]. However to check if a cell address is in the list a normal Contains() method is run on the collection. Profiling results showed that this method is run a substantially number of times, where the cell address is already there, and is therefore heavily contributing to the running time. After a talk with one of the developers behind the C5 library [C5], Peter Sestoft[26], a possible solution was found. Instead of using the method in the collection it was decided to use a two-dimensional boolean array, where the two dimensions represent the column and the row, for each sheet to represent if a cell was in the collection. This proved to be a major improvement in calculating the topological sorting, and was about a factor 2-3 faster.

Another profiling was done, and now the major contribution to the time consumption was collecting the cell addresses from the FAP grids in each step of the topological sorting. More precisely it was the large amount of insertion in array lists[27] that took time. This issue

[25] HashLinkedList class from the C5 collection [C5]
[26] Who also happens to be our supervisor
[27] Specificly the ArrayList from the C5 collection [C5]

was solved by allowing the FAP grids access to the two-dimensional boolean array, thereby totally avoiding the FAP grids inserting cell addresses to the array lists, if the address was already in the topological list. This contributed to the recalculation being another factor 2 faster, leading to a total factor of about 4-5 faster.

A final profiling was made, but this showed no single point or any easy optimization where speed performance of recalculation could be improved. This fact thus that the recalculation speed now was considered sufficiently improved, no further improvement was done in this version.

As an easy addition to sharing of FAP grid lists a feature to search the entire spreadsheet for equal FAP grids was made, comparing each FAP grid down to each FAP set in each FAP list. This enhances the sharing already introduced in Mark II, since this is only based on checking for equal FAP grid lists based on the same expression and column as described in section 3.7.1. This feature is invoked manually through the user interface[28] and is not used when building and maintaining the support graph otherwise. The results was not profiled but benchmarks[29] shows support graphs which shared from nothing up to 7 times more FAP grid lists than in Mark II.

4.6 Future development

The most obvious improvement would be to implementing the changes described in section 3.7.3, where FAP sets have both relative start and end. This would make sharing of FAP grid lists possible in all cases where copying of a cell is done.

[28] using the “Remove duplicates” button
[29] See section 6.4.3

5 SupportCalc implementation

This chapter shortly introduces our considerations when implementing the support graph. This is followed by an overview of the most important classes in SupportCalc with a short description of their function and relation to other classes. We highlight some interesting classes of SupportCalc afterwards, followed by a short description of various helper classes and applications used during the development of SupportCalc. Finally a short list of limitations and known errors in SupportCalc is presented.

5.1 Considerations when implementing the support graph

This section is not a complete walkthrough of the implemented program, nor does it give an overview of it. An overview can be found in section 5.2. It is mostly important considerations in respect to usage of certain specific data structures.

To understand this chapter an understanding of standard datastructures such as HashMap, Arraylist and Linkedlist, thus a understanding of how a combination of these will work is nessasery. A lot of data structures from the C5 library, hereby referenced as C5, are used. For further information about this libary see [C5].

5.1.1 Using a tree in FAP list

The FAP sets in a FAP list is in a collection, TreeSet from C5 that is a balanced red-black tree, that does not allow duplicates. Each FAP set is then inserted according to its a value. The benefit is that when inserting numbers or FAP sets their neighbours can be found in log(N) time where N is the number of FAP sets in the collection. Also when deleting a area or removing or inserting rows the FAP sets that are affected can be found in log(N) time instead of going through the whole collection (which might be the worst case anyway).

5.1.2 Using dynamic internal storage for FAP list

Many FAP lists only has one FAP set. Therefore it might be beneficial to change the internal representation of a FAP list to simply a FAP set. A FAP list or a FAP set could be encapsulated in another object, that delegate the correct methods, thus replacing the FAP set with a FAP list and opposite when needed.

This was tested and no benefit in memory usage was found, so the idea was dropped.

5.2 Program documentation

5.2.1 Overview

The overall classes and their relations are shown in the class diagram in Figure 5.1 using Unified Modelling Language (UML) as a representation. We will not explain the UML notation, but refer to [CORE, page 21] for a very short introduction or [PIJ, page 7] for a thorough introduction. Only relevant classes of SupportCalc are shown. We do not present CoreCalc's classes, but refer to [CORE, page 19-45] for a detailed description of the implementation of CoreCalc.

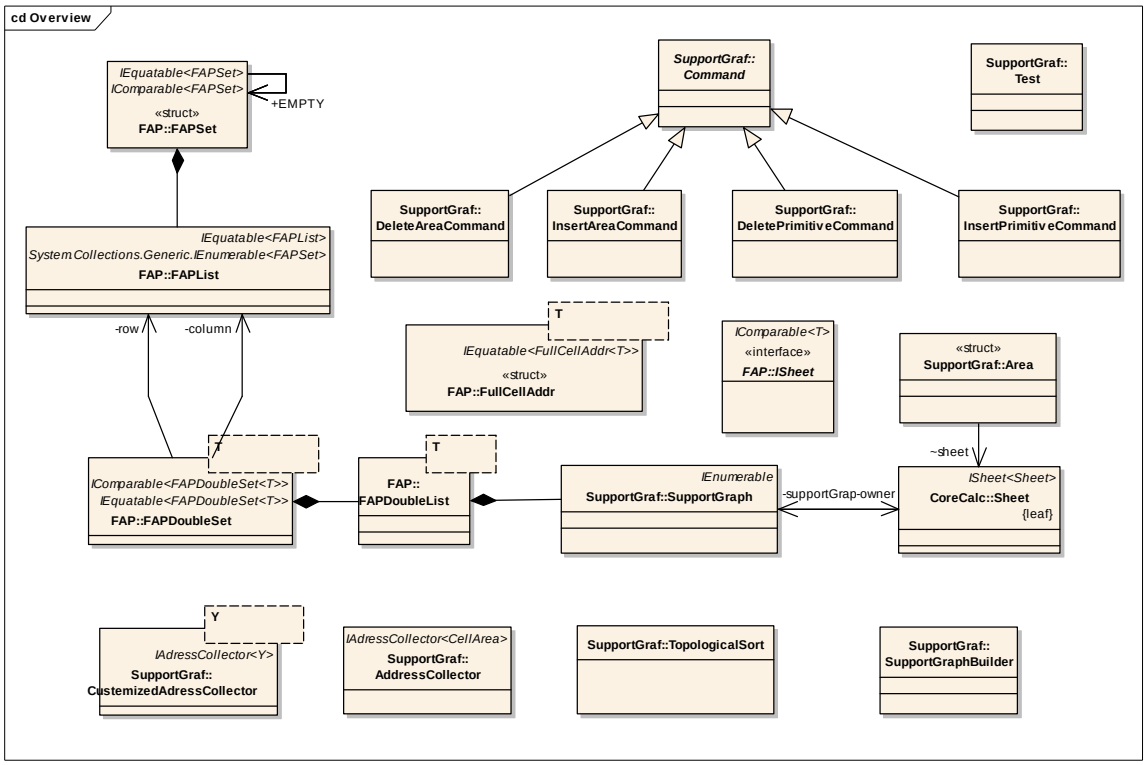


Figure 5.1: Overall classes in SupportCalc and the Sheet class from CoreCalc

The “Sheet” class from CoreCalc contains a “SupportGraph” object, which is a collection of “FAPDoubleList” objects, and methods to handle these. The "FAPDoubleList" class represent what is commonly referred to as FAP grid lists in this thesis. Each cell that supports another cell is represented in the collection. The “FAPDoubleList” contains "FAPDoubleSets" which correspond to FAP grids. These contain two "FAPList" objects, one for each dimension, containing a collection of "FAPSet" structs.

The “SupportGraph” collection mentioned before contains methods to maintain the support graph once it has been built initially. Inserting and deleting cells and areas, inserting columns and rows, executing a topological sorting etc.

“FullCellAddr” is a struct which represents an address as column, row and sheet. “FullCellAddr” adds the sheet belonging to the cell so we can reference cell from different sheets. This struct was added as the classes included in the FAP namespace are contained in a different project than the CoreCalc project, and therefore does not have access to the "CellAddr" struct in CoreCalc.

“TopologicalSort” represents the different types of topological sorting the “SupportGraph” class can perform. It performs sorting on a collection of “FullCellAddr”. The various topological sorting are based on [C5, page 197] and some are modified to use the boolean array described in section 4.5 and various collections can be chosen for the “FullCellAddr” that are sorted.

The “SupportGraphBuilder” class builds the support graph from an existing workbook. It builds the expression map, finds the referred cells from the expressions in the map and builds the support graph from the referred cells as described in section 3.5, Building the support graph.

“AddressCollector” and “CustomizedAddressCollector” are classes for extracting addresses representing the cells that a given expression supports statically.

The “Test” class can be seen as an equivalent to the Excel and Calc scripts that defines benchmarks for SupportCalc. In addition it contains some helper methods that are relevant when benchmarking and debugging the support graph, e.g.:

- Printing compactness related information
- Analysing the correctness of the support graph
- Converting between spreadsheet cell addressing formats etc.

The Command class encapsulates the concept of a spreadsheet altering operation, like insert cell and delete column using the Command design pattern [GOF, page 233]. Subclasses define specific spreadsheet altering logic, while the abstract base class Command defines shared logic like timing methods and an undo operation to revert changes. It is only used in executing benchmarks, but was planned to be used together with SupportCalc so every executed spreadsheet operation could be queued and undone.

5.3 Highlights of SupportCalc

This section shortly introduces the more interesting classes and functions of SupportCalc.

5.3.1 TopologicalSort

This class handles the topological sorting when a cell is changed. Seven different types of topological sorting is implemented, each of these as a method starting with "Toposort", where the rest are helper methods.

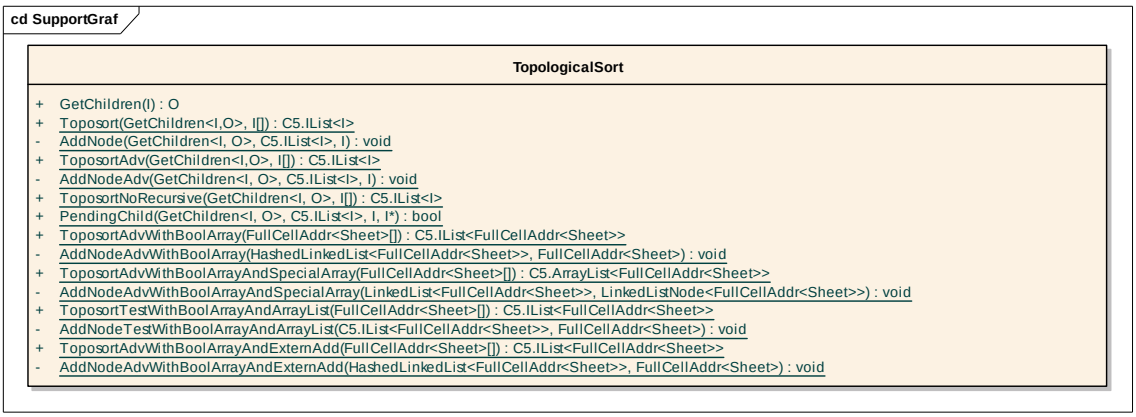


Figure 5.2: Methods of the TopologicalSort class

The topological sorting used is based on [C5, page 199], the three of the methods, "Toposort", "ToposortAdv" and "ToposortNoRecursive" are more or less direct copies of methods described in [C5]. To understand the description of the methods either reading the implementation or [C5] page 199 is necessary.

The rest of the topological sorting methods are:

- ToposortAdvWithBoolArray: Based on "ToposortAdv", but with a bool array to check if cells are in the list instead of using the built-in contains method in the HashLinkedList.

- ToposortAdvWithBoolArrayAndSpecialArray: Same as the method above, but using the built-in "System.Collections.Generic.LinkedList" instead of the C5 HashLinkedList [C5].

- ToposortTestWithBoolArrayAndArrayList: Same idea as the method above, just based on "TopoSort" and using an ArrayList from C5.

- ToposortAdvWithBoolArrayAndExternAdd: Based on "ToposortAdvWithBoolArray", but instead of checking if cell addresses are to be added in this method, this is moved out to where the addresses are collected (getting children).

The method "ToposortAdv" is used in Mark I and Mark II, "ToposortAdvWithBoolArrayAndExternAdd" is used in Mark III.

One basic difference from the implementation of [C5] is the introduction of a delegate method to get the children of node. This removes the need for implementation of nodes and enables dynamic finding of the children.

5.3.2 SupportGraph

The "SupportGraph" class is responsible for the support graph for a sheet. One "SupportGraph" object is therefore tied by double reference to each "Sheet" object (a sheet has a reference to the support graph for that sheet and vice versa). The sheet need to inform the support graph of which operation has occurred, so the support graph can perform necessary maintenance, whereas the support graph only needs access to the cells in the support graph in order to find references from which to build the support graph. The support graph holds a two-dimensional array of "FAPDoubleList" objects, where the position in the array, described as (column, row), corresponds to the location of the cell.

Furthermore the "SupportGraph" class is also responsible for finding the cells needed to be updated when a change occurs. This is does by using the "TopologicalSort" class described above.

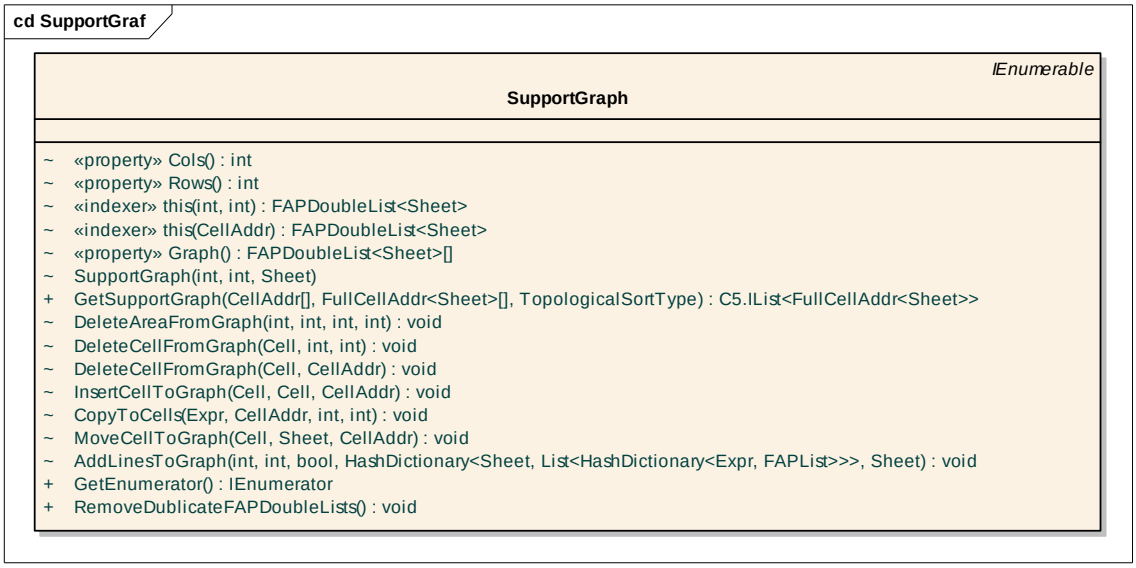


Figure 5.3: Methods of the SupportGraph class

5.4 Helper classes and applications

We use an alternative XML reader to load an existing XML sheet into SupportCalc which is a factor 2-4 faster than the original found in CoreCalc when loading the larger PROT and ROLE spreadsheets. The XML reader is taken from [XML] and resides in the "Excel2003XMLReader" class.

For message logging, we use a logging framework called Log4Net [LOG]. This logs messages from SupportCalc onto the console and in several flat files. This is used for debug, informational, error and benchmark related messages. We have created a static class "Log" which acts as a front-end to the framework.

An A1/C0R0 format converter, "CellFormatConverter", was created as a standalone helper application during debugging.

A FAPViewer application was created in which the effects of added and deleting FAP sets in FAP lists could be seen.

5.5 Limitations and errors

SupportCalc using the support graph has some of the same limitations as in CoreCalc as described below.

- Moving a cell is not implemented
- Delete row/column is not implemented, but insert row/column is.
- Handling of references which points to outside the sheet. For instance when the formula "=A1" in cell B1 is copied one position to the left to A1, then the reference in the formula gets adjusted and point outside the sheet and throws an exception.

Other limitations include:

- We have not implemented support for arrays and matrices using the support graph and using the alternative XML reader to load existing spreadsheets.
- Some debug related code was left in the constructor of the struct “CellAddr” in CoreCalc. This had a negative impact on performance when many “CellAddr” were created.

6 Performance measurements

First a description of which performance evaluation criteria we use to evaluate our benchmarks is introduced. These are derived from the three criteria from the problem formulation. An introduction to the overall benchmark setup follows, in which hardware, software, measuring tools, benchmarks methods, spreadsheet application and sheet used, are presented. The setup of the individual benchmarks is presented next and finally the results from the benchmarks are shown. Comments and notes about the results are presented and discussed further in chapter 7.

6.1 Performance evaluation criteria

Performance can be evaluated in different ways. This section presents the different performance evaluations we will use at various points in our benchmarks. They are derived from the three criteria defined in the problem formulation; Speed of recalculation, Compactness of support graph and Precision of support graph. Each performance evaluation is denoted by a letter in uppercase for easy reverence.

6.1.1 (A) Speed of recalculation

The speed of using the support graph in a recalculation which does not trigger any structural changes to the support graph (like copy/delete/insertion of rows/columns would do).

6.1.2 (B) Speed of maintenance and build of support graph

The performance of the support graph when its structure changes and when it is initially build.

6.1.3 (C) Compactness of the support graph

How compactly the support graph is represented compared to the number of edges it represents. We use the ratio of FAP sets to the number of edges in the support graph, that is edges pr. FAP set. When benchmarking the maintenance of the support graph we also check if the support graph becomes less compact after several maintenance related operations.

6.1.4 (D) Precision of support graph

How accurately does the topological sorting represent the cells that need to be involved in a minimal recalculation? Although a cell *c* depends on cells that have been changed *c* might not change, and therefore the cells that depend on *c* do not need re-evaluation.

6.1.5 (E) Memory usage

The overall memory usage for having the support graph and/or the workbook in memory. Heavy memory swapping in our benchmark environment might explain some reduction in performance. The currently used memory of the applications both in physical and in physical/virtual memory are used as measurements. Memory is always measured after the support graph has been built unless stated otherwise.

6.2 Overall benchmark setup

This section introduces our benchmark setup and the methods used for benchmarking recalculation times in both CoreCalc SE, SupportCalc, Excel 2007 [EXCEL] and OpenOffice Calc [CALC]. CoreCalc SE is a special edition of CoreCalc [CORE] where certain functions, a new and faster loader as well as other elements where added in order to perform benchmarks (basically CoreCalc SE is SupportCalc without support graph).

Excel 2007 was chosen since it is arguably the de facto spreadsheet application, both in terms of usage and performance. OpenOffice's Calc is a popular open source implementation of a spreadsheet application which makes it interesting to see how well it performs against both Excel and SupportCalc. CoreCalc SE was chosen as a reference since this is the basis of SupportCalc, and thus shows the difference between using support graph or not. Not all versions of SupportCalc are used in each benchmark, as the operations performed by the benchmark might not have been affected by the changes between versions.

6.2.1 Hardware

All benchmarks have been performed on a 2600+ AMD Sempron processor with 1.5 gigabyte memory. The processor frequency was forced to be constant to prevent inaccurate time measurements. We do however use virtualization software which implies some overhead in performance. 1024 megabytes of memory are available to the virtual machine.

6.2.2 Software

We use VMware server virtualization (version 1.0.1, build 29996) [VMS] to create a virtual environment to perform our benchmarks. We do this mainly to create a portable benchmark environment which during the testing phase of the project. Both host and guest operating system is Windows Server 2003 Enterprise, including available patches as of 15 January 2007.

We use a trial version of Excel 2007 (version 12.0.4518.1014), OpenOffice (version 2.1) and SupportCalc (Mark I, Mark II and Mark III) which are based on CoreCalc version 0.5. When benchmarking SupportCalc, we invoke the compiled binary *.exe file directly in order to eliminate the overhead of launching SupportCalc from our developing tool.

6.2.3 Measuring tools

Several macros are used for measuring recalculation times in the different spreadsheet applications.

Excel 2007

We use a macro presented in a technical paper from Microsoft [EXP, about 1/3 through the paper]. The macro makes use of the windows API to measure time, which according to the paper is more accurate than using the Time() function in Visual Basic for Applications (VBA). It should measure intervals accurately down to a few milliseconds interval.

SupportCalc

We make use of the Stopwatch class in .NET to measure time, which measures in intervals less than 1 millisecond on our hardware. We have made sure that the class uses a high resolution counter and not the less accurate system counter to obtain the high resolution. Furthermore we have made sure to operate the processor at a fixed frequency since the measurements depend on the frequency. Logging of the measurements is done with the Log4Net logging framework, both to the console and to a flat file.

OpenOffice Calc
We use a VBA macro which measures time using the GetSystemTicks() method, which returns the system ticks provided by the underlying operating system. On our benchmark system, we get 1000 ticks per second, so 1 tick = 1 millisecond. We tried using the Time() function as described in [RCG], but failed to get any useable time measurements. It seems that we get time measurements in intervals of about 15-16 ticks which equals to 15-16 milliseconds accuracy. The consequence is that we do not get very accurate measurements when benchmarking smaller spreadsheets.

Memory usage
For measuring memory usage in all three cases we use Process Explorer [PE] which is a small utility that among other things can measure process resource consumption.

6.2.4 Spreadsheets

We use several real-world spreadsheets in each benchmark we perform. We will shortly describe the overall characteristic and structure of each spreadsheet. Synthetic spreadsheets will be presented on a per case basis. The spreadsheets can be found on the CD-ROM in the "/Spreadsheets/" folder. If more than one file exists in a folder the original file is marked with "[ORG]" in the filename. If the spreadsheet was found on the internet a "source.txt" containing the location of the spreadsheet on the internet is present. Spreadsheets are presented sorted by size on disk with the largest sheet first. The largest are the most interesting spreadsheets for our purposes.

Some spreadsheets have been changed so they would function in SupportCalc. These changes are mostly equivalent in terms of functionality. The changes were correcting formulas that had errors (e.g. trying to add a string with a number) and replacing (reference) names with their actual references. We have also added functions in SupportCalc that were needed, thus making one "dummy"[30] method for a user defined function.

Lastly all spreadsheets were changed from their original format (typically Microsoft's *.xsl or OpenOffice's *.sxc format) to Microsoft's *.xml format which CoreCalc can understand. Some conversions were problematic, specifically *.sxc formats which could not directly be converted to *.xml by OpenOffice 2.1. We solved it by letting OpenOffice convert it to the *.xsl format and then let Excel 2007 convert it to *.xml. OpenOffice seems to have problems opening the *.xml version of the *.sxc spreadsheets, so we use the original format in these cases.

[30] A dummy method being a method that contains the same parameters and return the same type as the real method, for the intent of making the program work, but without the logic of the function.

Each spreadsheet is presented with a short description. Furthermore some statistics are shown which give an idea of how large the spreadsheet/workbook is and how well we represent it with FAP sets and grids. The ratio between the number of FAP sets and FAP grid lists shows the overall complexity of the FAP grids. If there are exactly twice the FAP sets than FAP grid lists, then we know each FAP grid list only consists of two FAP sets, since this is the lowest possible number of FAP sets pr. FAP grid list. The number also gives an indication of how complex the FAP grid lists are. This information is used in conjunction with the benchmark results later on. The edge count is the number of edges in the support graph, and is equal to the number of references in the spreadsheet.

Proteins chains (PROT)
A spreadsheet from "Danmarks Farmaceutiske Universitet" which is an attempt to build a data matrix with the phi-psi angels of proteins chains. This is the largest spreadsheet which was benchmarked.

Type of statistic for the whole workbook	Value
Size on disk (in megabytes)	51,8
Total number of cells	3358172
Total number of non-null cells	594524
Total number of cells with FAP grid list	330923
Edge count	763648242
FAP sets (initial build of support graph)	882773
FAP grids(initial build of support graph)	441332
FAP sets / FAP grids	2,00025
FAP sets / cells with FAP grid list	2,66761

Role-playing (ROLE)
A collection of spreadsheets defining various aspects of a role playing game. Contains many individual table structures, mathematical operations, functions (e.g. "VLOOKUP()", "IF()" and "MAX()") and cross-references.

Type of statistic for the whole workbook	Value
Size on disk (in megabytes)	23,7
Total number of cells	558265
Total number of non-null cells	112754
Total number of cells with FAP grid list	139696
Edge count	239152019
FAP sets (initial build of support graph)	3221850
FAP grids(initial build of support graph)	240435
FAP sets / FAP grids	13,40009
FAP sets / cells with FAP grid list	23,06329

Probability (PROBA)
A workbook containing probability calculations. Uses a user defined function called "ProbabilityRange()"[31] and also using the "SUM()" function to sum many references

[31] This custom function is implemented as a "Dummy" function in SupportCalc

containing probabilities. Especially the “Stack” sheet contains many references. Overall the workbook contains highly structured data where most cells are either number values or "SUM()" functions with references.

Type of statistic for the whole workbook	Value
Size on disk (in megabytes)	1,3
Total number of cells	18646
Total number of non-null cells	13148
Total number of cells with FAP grid list	4584
Edge count	287142
FAP sets (initial build of support graph)	10002
FAP grids(initial build of support graph)	5001
FAP sets / FAP grids	2
FAP sets / cells with FAP grid list	2,18194

Alpha scattering simulation (ALPHA)
A single spreadsheet simulating an encounter between an alpha particle and a nucleus. It is highly structured, containing mostly arithmetic operations between references and few mathematical functions and constants. Structure-wise most cells rely on the result of the previous cell in the same column.

Type of statistic for the whole workbook	Value
Size on disk (in megabytes)	0,32
Total number of cells	4812
Total number of non-null cells	1886
Total number of cells with FAP grid list	1529
Edge count	5643
FAP sets (initial build of support graph)	4657
FAP grids(initial build of support graph)	2328
FAP sets / FAP grids	2,00043
FAP sets / cells with FAP grid list	3,04578

Financial statement (FINAN)
A workbook containing the financial statement of “Dansk fjernvarme”. It almost exclusively consist of number values, arithmetic operations between references and "SUM()" functions.

Type of statistic for the whole workbook	Value
Size on disk (in megabytes)	0,1
Total number of cells	1287
Total number of non-null cells	671
Total number of cells with FAP grid list	576
Edge count	846
FAP sets (initial build of support graph)	1452
FAP grids(initial build of support graph)	726
FAP sets / FAP grids	2
FAP sets / cells with FAP grid list	2,52083

6.2.5 Benchmark methods

To perform our benchmarks in Excel and OpenOffice Calc we need to make modifications to the sheets to ensure that recalculation takes place. When measuring the speed of a recalculation (A) we change cells as little as possible in an effort to prevent structural changes in the support graph, but as we have no knowledge of the inner logic of Excel and OpenOffice Calc this is not guarantied.

In SupportCalc we have methods which can force a specific cell to be recalculated, even if its value and/or formula is unchanged. For Excel and OpenOffice Calc each cell marked for recalculation is handled as follows:

- **Empty cell:** Insert the new number 1
- **Number cell:** Add 1 to the current value of the cell
- **String cell:** Concatenate a string to the current string value of the cell
- **Formula cell:** Surround the formula expression *f* as follows: **"**=IF(1=1; *f*; 1)"

We benchmarked this in both Excel and OpenOffice Calc to see if indeed it forced a recalculation (since one could imagine that if the value of the cell does not change, as is the case with the formula cells, it would not recalculate its dependencies). The measurement of running times confirms that surrounding a formula expression with the above "IF" construct forces a recalculation even though the value does not change. A comparison was made against a benchmark where the value of the cell did change.

We measure memory usage of the spreadsheet applications by two criteria:

- The memory footprint of the process in physical memory which cannot be shared between other processes ("private working set" in Process Explorer [PE])
- The memory usage of the process in physical and virtual memory combined which cannot be shared between other processes ("private bytes" in Process Explorer [PE])

The working set can be larger than the private bytes if enough physical memory is available since unused memory is not removed from the working set. If available physical memory falls below a certain threshold working sets are trimmed so unused memory are released.

6.3 Benchmark definitions

The purpose of this section is to define benchmarks that test the performance of recalculation under simulated real life circumstances. We benchmark each of the spreadsheets mentioned in section 6.2.4 using measurement methods described in section 6.2.5. Here we evaluate the performance in regards to performance evaluation defined in section 6.1. Details of the specific parameters used in the benchmarks can be found in the "Test.cs" class in SupportCalc which can be found on the attached CD-ROM under "/Benchmark results/" in Word 2003 XML format.

Our benchmarks are as follows:

6.3.1 Speed of recalculation (A)

In this benchmark a number of cells are marked or changed so they should be recalculated, and the speed of the recalculation is then measured. Since changes from SupportCalc Mark I to Mark II have no effect on the speed of recalculation, the Mark II version has not been benchmarked.
Furthermore benchmarks are only shown for the larger ROLE and PROT sheets, since the smaller sheets FINAN, ALPHA and PROBA all have a worst running time of less than a second with Mark I, and are therefore not interesting with respect to the support graph. The average speed of 10 runs will be presented. It is expected that Mark III will have a significant improvement in the time for calculating the topological sorting and thereby also the total recalculation speed. As per [EXP], recalculation times below one second should not interrupt the user's concentration.

The following benchmarks where made:

1% dependencies
This benchmark simulates a "best case" scenario where only a few cells need to be recalculated because of very few dependencies from the changed cell(s) (only 1% of the total number of non-null cells in the sheet are included in a recalculation). We expect the performance to be much better than a full recalculation. We do not recalculate cells containing volatile functions, unless we manually add them in our benchmarks, since they influence the total number of cells involved the recalculation. This is also true for the 5% and 10% benchmark.

5% dependencies
5% of the total number of non-null cells are involved in a recalculation. The performance again expected to be faster than a full recalculation.

10% dependencies
10% of the total number of non-null cells are involved in a recalculation. The performance is still expected to be faster than a full recalculation.

All dependencies
This simulates a "worst case" scenario, where all cells are involved in a recalculation (even cells that are null). Because of the overhead of using and maintaining a support graph, the performance should be much worse than a full recalculation. This case differs from a normal "full" recalculation in that we manually change every cell in the sheet and invoke a recalculation of changed cells. In SupportCalc we use the support graph.

Full recalculation
This benchmark invokes the built-in full recalculation method of each spreadsheet application which forces a full recalculation of the whole workbook. In the case of SupportCalc, CoreCalc SE was used instead. We expect is to be faster than the "All dependencies" benchmark, but slower than the 1-5-10% cases.

6.3.2 Speed of support graph maintenance (B)

The purpose of this benchmark is to see how fast operations that alter the support graph are. We measure the time it takes to perform different types of changes on a spreadsheet. This benchmark is not performed with the Mark III version of SupportCalc as the speed of maintenance was not improved between Mark II and Mark III. We do not benchmark Excel or OpenOffice Calc since it is not clear how to measure support graph maintenance separately from the evaluation of cells.

The performance when using Mark II is expected to suffer from the fact that overlapping FAP grids may be present[32] and the same operation on the FAP grids therefore can be performed up to four times. On the other hand, insertion of cells (especially a large number of cells) might benefit from the sharing of FAP grid list and the sharing of temporary FAP grid list operation results.

A series of smaller operations are expected to be slower than a single, larger operation that performs the same structural changes. This is due to the fact that insert and delete operations on FAP grid lists are based on areas, so an area insert or delete will only update the affected FAP grid lists once. We only benchmark spreadsheets ROLE and PROT because the other lack size and complexity. The smaller sheets are simply processed too fast to obtain useful results. In the ROLE sheet we execute benchmarks on the “Skill” sheet since it is the largest. Each result is an average of 10 runs. We benchmark CoreCalc SE, SupportCalc Mark I and SupportCalc Mark II. We do not include the time used for recalculations in the results, since we are only interested in the speed of the actual structural changes. The definition of the benchmark is as follows:

1) Insertion of a single cell which depends on and supports at least one other cell.
2) Deletion of a single cell which depends on and supports at least one other cell.
3) Insertion of an area (size of area is 1% of non-null cells) which depends on and supports at least one other cell. We do not use the optimized area-based methods for this.
4) Deletion of an area (size of area is 1% of non-null cells) which depends on and supports at least one other cell. We do not use the optimized area-based methods for this.
5) Insertion of an area (size of area is 1% of non-null cells) which depends on and supports at least one other cell using area-based methods.
6) Deletion of an area (size of area is 1% of non-null cells) which depends on and supports at least one other cell using area-based methods.

When inserting a new cell, we insert the formula “=$A$1”. We alternate between deleting and inserting cells and timing them separately. Notice, that in the case of benchmarks using the support graph, an insert operation must first delete the existing entry. Therefore we effectively benchmark delete and insert combined in this case.

[32] described in section 3.7

6.3.3 Support graph build time (B)

This benchmark simply measures the time it takes to build the support graph from scratch. Only SupportCalc Mark I and Mark II are used, since this can not be measured in the other programs (except Mark III which does not offer any changes to Mark II in this situation). Excel can forcefully rebuild the support graph, but a full recalculation is invoked afterwards which makes it difficult to measure the actual build time of the support graph.

In the case of Mark II the reuse of calculations and FAP grid lists in the support graph should make a complete build of the support graph faster than Mark I since we replace many additions of FAP grids to FAP grid lists and deletions of FAP sets from FAP lists with a less expensive lookup operation. The lookup operation is used to find a FAP set that can be reused. All 5 spreadsheets are benchmarked.

6.3.4 Compactness (C)

The compactness of the support graph in SupportCalc after an initial build is measured. This is measured as support graph edges pr. FAP set. This is tested in SupportCalc Mark I, Mark II and Mark III.

It is expected that the number of FAP sets is much higher than the number of edges since we can represent many edges per FAP set. Furthermore, Mark II is expected to have a vast improvement over Mark I since the FAP grid lists and indirectly the FAP sets can be reused. Mark III is expected to have a minor improvement over Mark II, since FAP grid lists that are equal but not based on the same expression and column[33] during the build of the support graph[34], can be reused. However, in Mark I there can be no overlaps whereas in Mark II overlaps between relative and non-relative FAP lists can occur which theoretically could make Mark II worse.

6.3.5 Degradation of support graph (C)

An investigation is made into how gracefully the support graph degenerates after introducing structural changes. Ideally the support graph should not degrade. Benchmarking is done by performing support graph altering operations and then undoing the operations in reverse order. We measure the FAP set count before and after the changes. We benchmark in the ROLE and PROT workbooks. In the ROLE workbook we use the "Skills" sheet. In both benchmarks we perform the following operations using area optimized functions:

- Delete area E100:H500
- Insert formula "=Sum(A1:C20)" to the area C90:J200
- Delete area G120:I300
- Insert formula "=$C5" to the area D20:F180

We then reverse the effects of the above operations to get the same sheet.

[33] If column scanning is used, otherwise row
[34] As described in section 3.7.1

6.3.6 Precision of support graph (D)

We benchmark how well our support graph reflects the actual cell dependencies in the workbook. More specifically we see how accurately the topological sorting represents the cells that need to be involved in a minimal recalculation. Lack of time prevented us from benchmarking the real-life so we demonstrate the problem on specially constructed spreadsheets[35]. As described in section 3.8.4 over-approximation is done, and this benchmark should reflect the impact of the added recalculation.

Figure 6.1 shows the specially constructed spreadsheet:

	A	B	C
1	“price of selected product”	“VAT inclusive”	“Total price VAT inclusive”
2	=$C$6	=$C$6*1.25	=$B$5*$C$6*1.25
3			
4	“Selected product #”		
5	45		
6			=VLOOKUP($A$5, A8:C10, 3, false)
7	“product id”	“name”	“price”
8	45	“OpenOffice Calc”	0
9	46	“Excel”	299
10	47	“CoreCalc”	0

Figure 6.1: A specifically designed spreadsheet to show the effects of over approximation of the cells need to be re-evaluated. The dotted lines represent the cell being supported.

In the benchmark we change the value of cell C9 to 199 and note how many cells are recalculated according to the static support graph contra the number of cells that change value. We expect a higher over approximation when non-strict functions are involved, especially if the dependencies of the branches are deep and/or cover a large area (e.g. VLOOKUP()). In this case we expect cell C9, C6 and the area A2:C2 to be recalculated according to the static support graph. Only C9 should change value, since C6 still evaluates the value of cell C8 which has not been changed.

6.3.7 Memory consumption (E)

The memory consumption is measured after loading the sheet and building the support graph (except for CoreCalc SE that only loads but ca not build a support graph).

It is expected that the results follow the results from the "Compactness(C)" benchmark, in chapter 6.3.4. That is that since Mark II share FAP grid lists this should use substantially less memory than Mark I, and since Mark III further improves this by allowing further reuse of FAP grid lists as described in section 4.5, Mark III should use a bit less memory than Mark II.

[35] If we had more time we would execute the benchmark by storing all the current values of cells, executing a “Speed of recalculation (A)” benchmark, then comparing the values in the topological sort with the ones stored previously.

6.4 Benchmark results

This section presents our results for each benchmark presented above. Raw benchmark data can be found on the CD-ROM at "/Benchmark results". Notice however that OpenOffice Calc failed to open the ROLE sheet and thus no results are presented in this case.

6.4.1 Speed of recalculation (A)

As can be seen in Figure 6.2 the recalculation of the PROT sheet in SupportCalc Mark I and Mark III is substantially faster than CoreCalc, in the 1% and 5% cases. However both Excel and OpenOffice Calc are much faster than CoreCalc. The most interesting fact however is that there is no improvement of the recalculation time between SupportCalc Mark I and Mark III, which indicates that the topological sorting is not always the bottleneck of SupportCalc Mark I as otherwise indicated in section 4.5, at least not for all sheets.

The values are all too high for operating a spreadsheet application normally, maybe with the exception of Excel.

The All benchmark was heavily influenced by the lack of physical memory, which could be seen by almost no processor activity during the benchmark. Excel and OpenOffice Calc could not execute this benchmark at all. Excel loops in the first recalculation attempt, getting to around 70% of the total recalculation done before looping. An investigation showed that the way we change the cells in Excel did not create a cyclic dependency, which could have explained the behavior. Calc failed to complete even after 6 hours (processor usage was high during this period and the recalculation had not begun). This might be due to the VBA script that changes the content of each cell in the workbook.

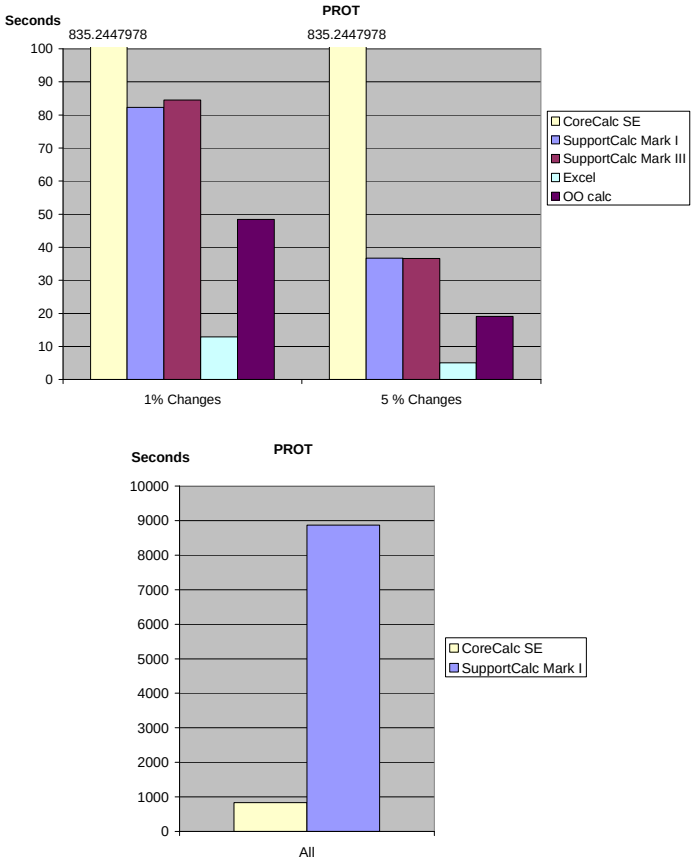


Figure 6.2: Recalculation speed benchmark (A) of PROT

Notice how recalculating five times the amount of cells in the PROT sheet is still about three times faster than the in the "1%" benchmark using SupportCalc. We confirmed that the amount of recalculated cells were in fact correct. It could be that the complexity of formula recalculation is greater in this benchmark than the former.

Figure 6.3 shows the recalculation speed of the ROLE sheet. In both the 1% and 5% case, SupportCalc has a clearly advantage compared to CoreCalc SE. The over 2 seconds it takes for CoreCalc SE to recalculate would normally be too much if this happened every time a cell was altered. In the case of 10% the advantage of optimizing the topological sort can clearly be seen in the difference of SupportCalc Mark I and Mark III. Excel has no problem in recalculation and is superior compared to SupportCalc. OpenOffice Calc could not open the sheet.

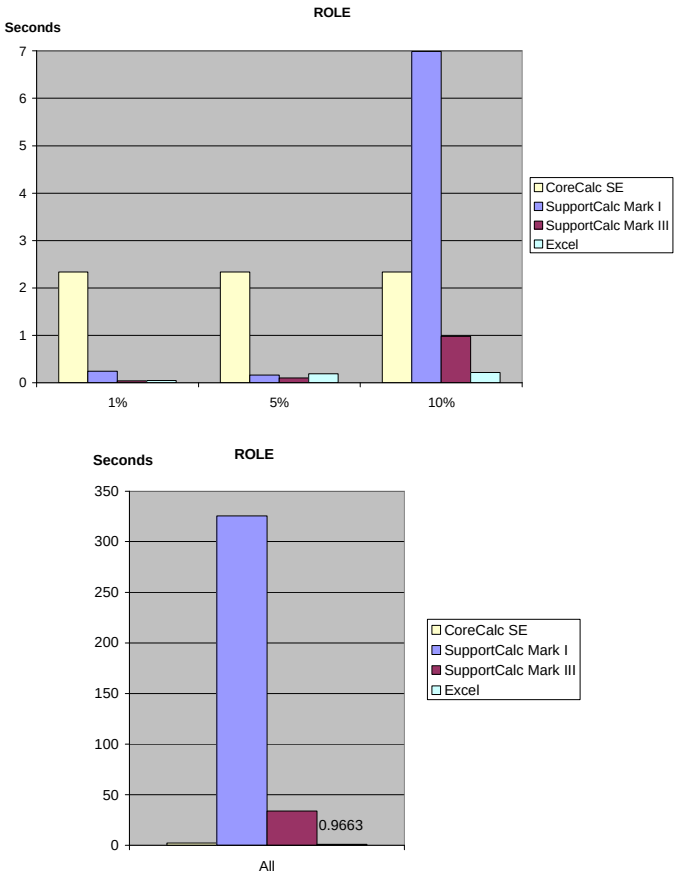


Figure 6.3: Recalculation speed benchmark (A) of ROLE

Since the benchmarks of the PROT sheet did not show expected results when comparing SupportCalc Mark I and Mark III an investigation in the results was done. The two graphs above show only the full recalculation time, but not how much the topological sorting takes vs. the evaluation of cells. Therefore a new graph was made, which shows the contribution of the topological sorting and the evaluation of cells to the recalculation time. This is shown in Figure 6.4. Only the most time consuming benchmark (except the “All” benchmark) for each sheet has been included.

Clearly this shows that spreadsheets differ in construction and it is not in all cases that the topological sorting is a big a contributor to the recalculation time as presumed during the development of the Mark III version.

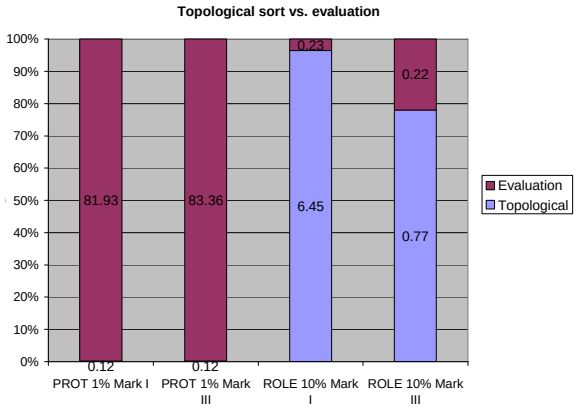


Figure 6.4: Contribution to recalculation time in percent.

6.4.2 Speed of support graph maintenance (B)

The insertion and deletion of a single cell took less than 1/10 of a second and are therefore not included in the graphs. Figure 6.5 shows the time for the support graph to make structural changes during a spreadsheet operation. For CoreCalc SE the time for restructuring the spreadsheet has been measured, which means that this should not be compared with the results of SupportCalc. It is used as a reference to measure if the time the support graph takes will use significant more time during an operation than the rest of the spreadsheet application.

Not surprisingly the un-optimized delete area that takes each cell and deletes it individually is rather slow, since FAP grids then might be split into several FAP grids for each such delete operation, instead of just once with the optimized delete area. It is a bit surprising that it seems that the Mark II version is faster than Mark I when it comes to deleting area(un-optimized), since Mark II should in fact make more overhead in this operation and not benefit in speed from the sharing of FAP grid lists. But the method for deleting a single cell was changed from a specialized method in Mark I to the general area optimized method in Mark II, and maybe some very un-optimized code was present in the specialized method.

On the other hand the insert area should normally be faster in Mark II than in Mark I since this utilizes the same calculation speed benefit as when building the support graph with shared FAP grid lists in Mark II. However since the insert in the benchmark only inserts a expression with a single reference only the FAP grid list of one cell is altered, which possibly explains why this operation is slower in Mark II than in Mark I, since Mark II has some overhead in allowing reuse of calculation results and FAP grid lists. The maintenance of the support graph is fast in comparison with the spreadsheet maintenance (CoreCalc SE) for the same operation, except for the un-optimized delete area of Mark I.

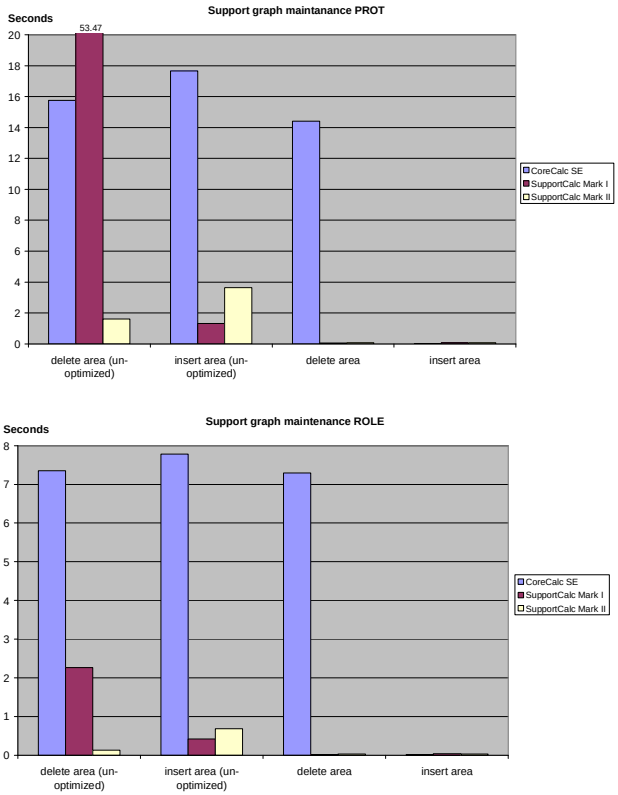


Figure 6.6: Maintenance of the support graph

6.4.3 Support graph build time (B)

Below in Figure 6.7 the build time of the support graph can be seen. Unless the support graph can be saved with a spreadsheet this would be the time it would take for the support graph to be ready when loading a spreadsheet. The spreadsheets PROT and ROLE are the most interesting in this benchmark, since the other sheets has a build time of less than a half second. When looking at the PROT sheet the build time is rather high, taking several minutes for the Mark I version, and the Mark II version takes even longer indicating that not much calculation can be reused, and possibly that not much reuse of FAP grid lists are done. Instead the possibility for sharing just creates overhead, or maybe some memory problem occurs. The benchmark regarding compactness and memory consumption might indicate where the problem lies.

With the ROLE sheet the change from Mark I to Mark II is significant. It is a bit hard to see from the graph but the build time of ROLE in the Mark II version is about 45 seconds, which is fast enough for most users to accept the waiting time. Clearly the ROLE sheet benefits from reuse of calculations.

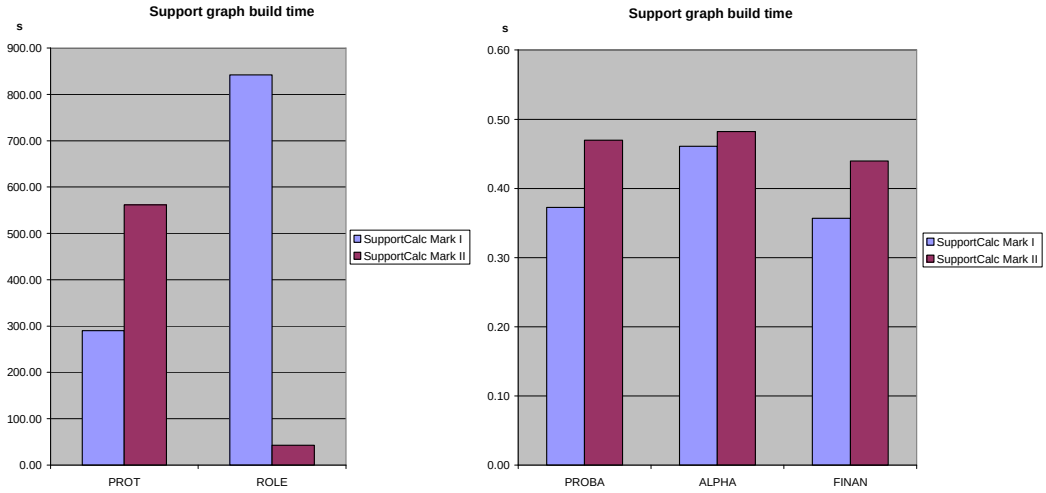


Figure 6.7: Build time of support graph

6.4.4 Compactness (C)

Figure 6.8 shows the compactness of the support graph expressed as edges pr. FAP set. As the sheets ALPHA and FINAN are quite small it is no surprise that the edges pr. FAP set is small for them, since the areas expressed by the FAP sets most likely are smaller as the sheets simply are smaller.

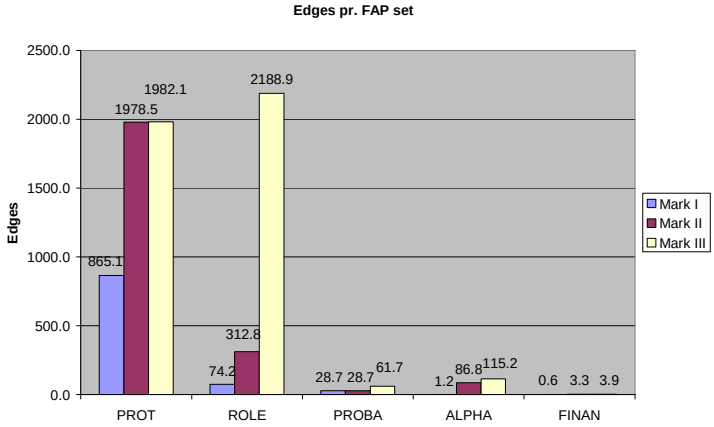


Figure 6.8: Compactness of the support graph

However the PROBA sheet is a bit bigger and is the only sheet where the change from Mark I to Mark II did not improve the overall compactness. A further investigation of the PROBA sheet revealed that almost all formulas are of the type “1-SUM(A$1:A1)” in columns where A1 is increasing as shown in Figure 6.9.

	A	B
1	1%	1-SUM(A$1:A1)
2	2%	1-SUM(A$1:A2)
3	3%	1-SUM(A$1:A3)
4	4%	1-SUM(A$1:A4)

	A	B
1	1%	(1,1,1) x (1,1,1)
2	2%	(1,1,1) x (1,1,2)
3	3%	(1,1,1) x (1,1,3)
4	4%	(1,1,1) x (1,1,4)

Figure 6.9: Example of PROBA formula structure with no sharing as no FAP grids lists are the same

The left table in Figure 6.9 shows the formulas, while the table on the right shows the FAP grids corresponding to the area the formulas depend on. This is an example of the current sharing problem as described in section 3.7.3, where the lack of FAP sets ability to have different relativeness in start and end can cause less reuse.

Another graph showing the compactness is shown in Figure 6.10 that shows the number of FAP sets pr. non null cells for the PROT and ROLE sheet. The figure suggests that there might be a proportional relationship between the number of non null cells in a sheet and the number of FAP sets.

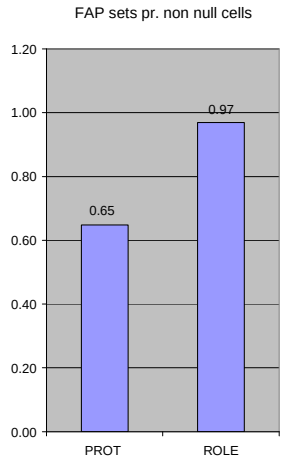


Figure 6.10: FAP sets pr. non-null cell in SupportCalc Mark III

6.4.5 Degradation of support graph (C)

In Figure 6.11 the degradation of the support graph after structural changes are shown. The results are given as the number of percent that the structural changes have increased the number of FAP sets with. The benchmark was made in the Mark II version and is divided into two categories; the actual FAP set count and the number of FAP sets there would be without sharing (so shared FAP sets are added for each time they are shared). The graph shows that a slight degradation of the support graph happen, which could make the support graph larger, but also that this degradation possibly can be avoided if duplicate FAP grid lists where eliminated after enough changes have been made[36].

[36] This can be done by running the remove duplicates method described in section 4.5

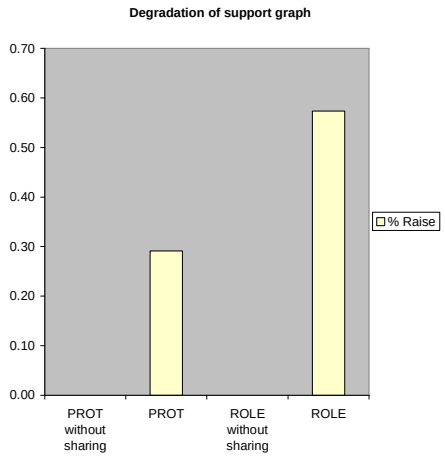


Figure 6.11: Degradation of the support graph after changes

6.4.6 Precision of support graph (D)

	A	B	C
1	"price of selected product"	"VAT inclusive"	"Total price VAT inclusive"
2	=$C$6	=$C$6*1.25	=$B$5*$C$6*1.25
3			
4	"Selected product #"		
5	45		
6			=VLOOKUP($A$5, A8:C10, 3, false)
7	"product id"	"name"	"price"
8	45	"OpenOffice Calc"	0
9	46	"Excel"	199
10	47	"CoreCalc"	0

Figure 6.12: Precision benchmark results, where the value of cell C9 has been changed. The dotted lines indicate dependencies used in the recalculation; the arrowhead points to the cell which is supported. A dark grey cell is a cell which value has been changed in the recalculation; light grey is a cell which did not change value, but where re-evaluated anyway as a result of over approximation.

Only a single cell out of four did change value, hence much unnecessary recalculation. Only changes to cell A5, A8 or C8 would result in cells C6, A2, B2 and C2 being recalculated.

6.4.7 Memory consumption (E)

The memory consumption after building the support graph with SupportCalc Mark II in the PROT sheet is actually lower than in CoreCalc SE, as can be seen in Figure 6.13. This is most likely because of lack of memory while building the graph, which most likely would result in garbage collection and swapping some memory to disk. This would also explain the slower support graph build time when using Mark II compared to Mark I as was shown in Figure 6.7. The memory usage is acceptable in all of the three versions of SupportCalc although the Mark I version is a bit high.

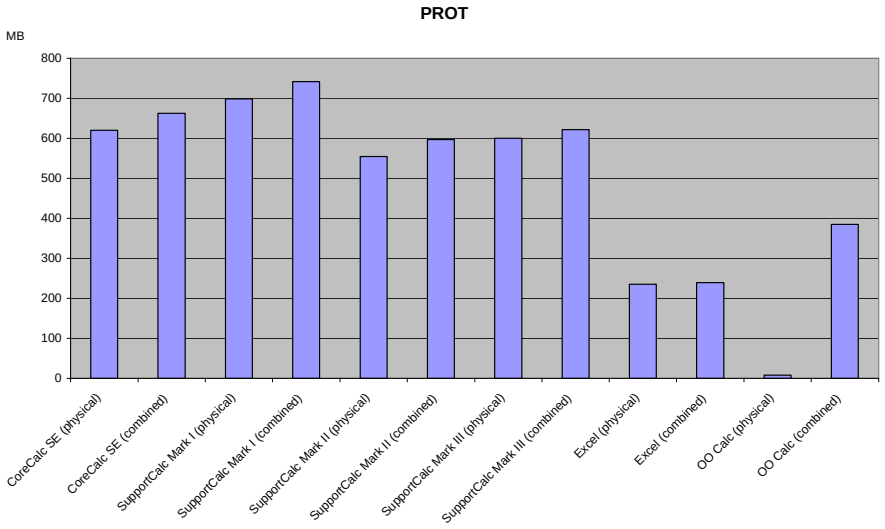


Figure 6.13: Memory usage of PROT

In Figure 6.14 the memory usage of the ROLE sheet is shown. Clearly the lack of a compact representation of the support graph in SupportCalc Mark I can be seen on the very heavy memory load that is almost 5 times the size of CoreCalc SE. The reuse of FAP grid lists really improves this in Mark II decreasing the memory to about 2 times that of CoreCalc SE. Although Mark III should give a more compact representation the memory usage is almost the same. Since the further sharing of FAP grid lists in Mark III happens by firstly building the support graph like in Mark II and first then remove duplicates, this could be caused by lack of garbage collection before the benchmark was done.

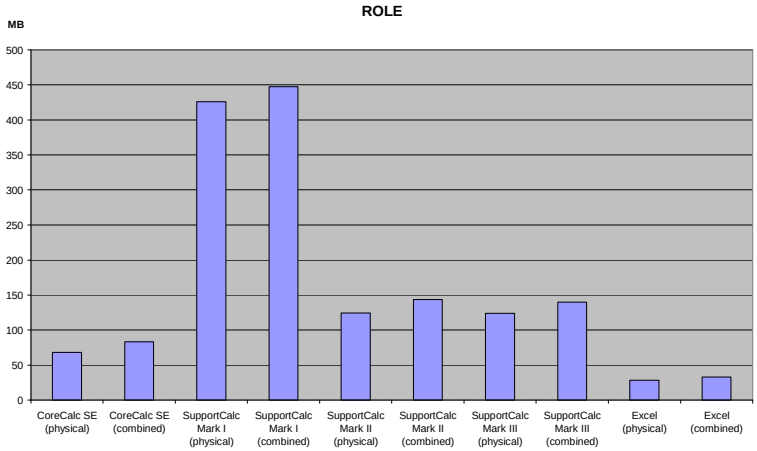


Figure 6.14: Memory usage of ROLE

The last graph, in Figure 6.15 shows the memory usage for the three smallest spreadsheets. Here the support graph uses almost no memory, which can be seen by the difference between the memory usage of CoreCalc SE and SupportCalc.

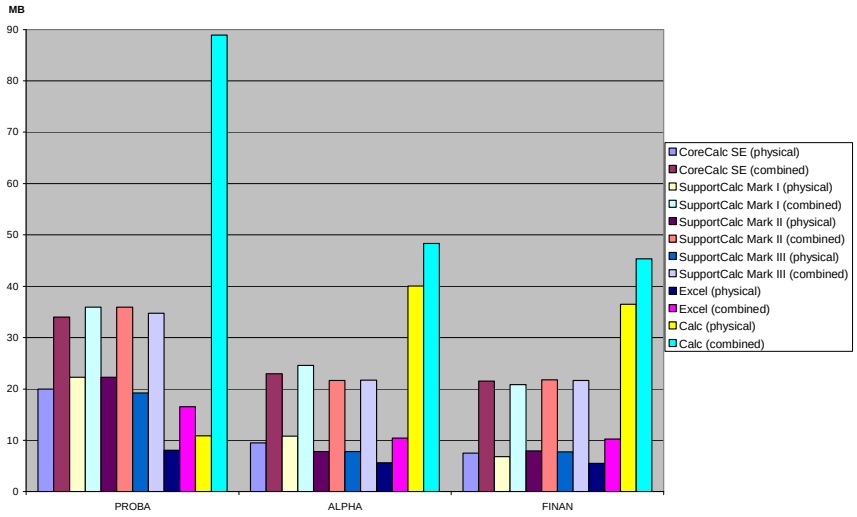


Figure 6.15: Memory usage of the smaller sheets

6.5 Benchmark evaluation

In the benchmark performed in section 6.4.1, Speed of recalculation, there is a lack of benchmarks where more than 10% of the spreadsheet is recalculated. This is a result of using a combination of the ROLE sheet and the Mark I version to decide which benchmarks to perform, since at that stage Mark I was the only existing version and the ROLE sheet was the first spreadsheet for which all necessary functions where implemented.

A statistical study of how many cells are typically used in a recalculation would help us to define more realistic benchmarks.

For most benchmarks only the two large spreadsheets PROT and ROLE showed any interesting data. Clearly more spreadsheets might show more results and may help to pinpoint or show new problems.

7 Discussion

Here follows a discussion of how well the implementation lives up to our requirements as stated by the problem formulation. The criteria where:

- Speed of recalculation
- Compactness of the support graph
- Precision of the support graph

Finally, two other important aspects of the project are discussed: Comparison with Excel/OpenOffice Calc and the need for further/better spreadsheet samples.

7.1 Speed of recalculation

Speed of recalculation involves both the speed of performing the recalculation after cells have changed, the speed of performing the necessary structural changes to the support graph as well as the speed of building the support graph.

7.1.1 Speed of performing recalculation after cells have changed

With regards to performing the recalculation when cells have changed there are two major aspects.

The first is that although the topological sorting has been vastly improved, further improvement might decrease the recalculation time of some sheets. As shown in the benchmark of recalculation after cells have been changed in section 6.4.1, most of the ROLE sheet's recalculation time is still due to the topological sort. But still the recalculation in the benchmark where 10% of the ROLE spreadsheet is calculated is an example where the support graph would definitely benefit the end user, as the waiting time have been decreased to an acceptable level.

The other aspect is that not enough focus has been devoted to decreasing the number of cell evaluations in the cases where the evaluation of cells was the major time consumer. Specifically this can be seen in the recalculation benchmark of the PROT sheet, where the topological sorting takes almost no time at all and the modifications from SupportCalc Mark I to Mark III therefore did not improve the recalculation time. An investigation into possible causes discovered that the main contributor to the evaluation time was the new spreadsheet functions implemented in this project[37]. Therefore optimization of these functions might decrease the evaluation time substantially. This also goes to show that spreadsheets can have very different structures requiring very different types of improvements to SupportCalc. Also too few different spreadsheets have been used for benchmarking and profiling during the development of the different versions, since these spreadsheets only showed the topological sorting as the main contributor to the recalculation time. However it should be mentioned that during the development the PROT sheet exhibited high memory usage in some benchmarks that resulted in extensive virtual

[37] The result of this investigation can be seen in the appendix in section 11.3.3.

memory usage possibly influencing the recalculation times during profiling and benchmarking[38].

A possible addition to SupportCalc could be a feature that automatically recalculated all cells when the list in the topological sorting became too large compared to the number of non-null cells in a sheet. This could reduce the huge recalculation time seen in the benchmarks where all cells are recalculated. However, as seen in section 6.4.1 the recalculation time does not necessarily increase with the number of cells. It is not clear how to find a better indication of when full recalculation might be faster.

The speed of recalculation is faster using the support graph than without, at least when recalculating 10% or less of a sheet. The increase in time, from the 5% dependencies case over the 10% dependencies case and up to the recalculation of all cells[39], indicate a rapid increase in time usage compared to CoreCalc SE when the number of cells to be recalculated increases. If this indication is generally true, then this extra time consumption is most likely caused by the topological sorting.

7.1.2 Speed of support graph maintenance

As the benchmarks in section 6.3.2 show, the speed of performing structural changes to the support graph is acceptable. However, this only includes insertions and deletions. Some other operations such as insertion and deletion of rows, moving and copying of large areas and matrix formulas may have a large impact on the support graph and therefore also the speed with which the necessary structural changes can be performed[40].

7.1.3 Speed of building the support graph

The speed of building the support graph varies dramatically according to the specific spreadsheet. As the benchmark in section 6.3.3 shows the PROT takes around 5 minutes for a complete build– but then even opening this sheet in Excel takes quite some time[41]. For the ROLE sheet the time for building the support graph is around 45 seconds which is more acceptable, at least when dealing with large sheets.

The building of the support graph could be avoided if the support graph was saved and loaded with the spreadsheet. This would however mean that possible degradation of the support graph would survive saves and load of the sheet. To prevent this continuous degradation the support graph should be rebuilt in the background from time to time. Alternatively, like in Excel, a possibility for the user to manually rebuild the support graph could be made.

[38] Profiling on the PROT sheet which uses all available memory was a problem since the profiling also takes a substantial amount of memory
[39] In the ALL benchmark
[40] Undocumented benchmarks where made with inserting rows in the top of the PROT sheet which took approximately 20-25 seconds.
[41] General experience, not documented.

7.2 Compactness of support graph

Overall the size of the support graph is small. The compactness problems that existed in SupportCalc Mark I were overcome with the sharing of FAP grid lists in the Mark II, which the benchmarks in section 6.4.4 clearly show. The end result is that only the ROLE spreadsheet increases the memory usage noticeable when using a support graph, with about 75% more memory usage than without using support graph. On the other sheets the usage of support graph increases the memory usage with less than 1%.

The only spreadsheet in the benchmark which was relatively large, and where high compactness was not achieved, was the PROBA sheet. This could be explained by either the relatively small size of the sheet[42] or by the fact that the sheet contains a lot of area references where at least one dimension is a combination of absolute and relative cell references, e.g. "A$1:A3". This is one of the cases where FAP grid lists can not be shared, but as described in section 3.7.3 this could be overcome by the implementation of FAP sets with independent relativeness for start and end.

7.3 Precision

SupportCalc currently re-evaluate all cells from the topological sorting. This is not always needed, as some cells might not be changed - especially cells containing non-strict functions which dynamically[43] only depend on cells not being recalculated could be excluded.

An obvious and easy way to avoid these unnecessary recalculations would be to check if a cell containing a non-strict function maintains its value during the evaluation, and if so, try to avoid that the dependent cells are recalculated[44]. It is clear from section 6.4.6 that this could pose a problem even when only a single non-strict function is present. We have not investigated this further, but both ROLE and PROT contains plenty of non-strict functions, so it seems plausible that a substantial decrease in the number of cells re-evaluated could be achieved.

Another approach is to implement a dynamic support graph as described in section 3.8.4 - thereby avoiding the recalculation of a lot of cells already in the topological sorting. This approach will involve an increase in support graph maintenance, and thus possibly further degradation of the support graph. But since the benchmarks in section 6.4.5 show that the current degradation during structural changes is nearly non-existent, and as the speed of recalculation could use improvement, a trade-off of higher degradation for faster recalculation would be desirable.

[42] Details of the sheet can be found in section 6.2.4
[43] Section 3.8.4 describes this
[44] Described in section 3.8.4

7.4 Other aspects

In all relevant[45] aspects SupportCalc performs worse than both Excel and OpenOffice Calc (although OpenOffice Calc had some major problems with some of the spreadsheets used). However, this might be a result of a major difference between SupportCalc, Excel and OpenOffice Calc in terms of the time taken to actually recalculate each individual cell.

Only two spreadsheets where large enough to really show some impact in the benchmarks; PROT and ROLE. Thus, the benchmarks and this discussion are based primarily only on two spreadsheets, making it difficult to draw certain conclusions. For future development it will be necessary to either find more large, real-life[46] spreadsheets or to create synthetic ones. Other constructions of spreadsheets than the ones encountered in this project might lead to other problems when using support graph, as demonstrated by the impact of the different construction of the ROLE and the PROT sheet[47].

Results that differ from the ones presented in this thesis were obtained in [RCG] in which CoreCalc was used to obtain performance figures which were typically faster than Calc and in some instances than Excel. There can be several reasons for this. First, many of benchmarks performed in [RCG] were simple expressions. We have benchmarked against real-life spreadsheets which contains plenty of complex expressions, references, non-strict functions and such. Many of these non-strict functions were implemented during this project and are probably less than optimal in performance as later profiling has shown[48]. In [RCG] all spreadsheets where constructed using very few different expressions, primarily a single expression which was copied to the other cells in the sheet. Arguably the spreadsheets constructed in [RCG] were far from realistic and from personal experience SupportCalc performed far better on our constructed spreadsheets, e.g. CONST1, used during development. These were built in a similar way; simple expressions and references copied to a large area. A further investigation into these performance differences would be interesting. An interesting idea is to combine the code generation from [RCG] with our support graph implementation which would certainly increase the speed of re-evaluation of cells.

[45] When using larger spreadsheets where performance matters.
[46] Which from our experience is harder than it sounds.
[47] As mentioned: during recalculation the PROT sheet uses almost all time on re-evaluating cells whereas ROLE uses almost all time on the topological sorting, see benchmark results in section 6.4.1
[48] See appendix section 11.3.3

8 Conclusion

The goal of this project was to investigate the effectiveness of a so-called support graph in speeding up recalculation in a spreadsheet implementation. More precisely, we have implemented, evaluated and improved the support graph presentation proposed by Peter Sestoft [CORE, chapter 4]. To conclude on our efforts towards equipping CoreCalc [CORE] with support graph based recalculations, lets sum up how well our SupportCalc application achieves the goal stated in the problem formulation:

> "Design and develop a support graph as an extension and thereby improvement of a spreadsheet application, with focus on the speed of recalculation, compactness and precision. The goal is to achieve similar performance to that of Excel 2007 and OpenOffice Calc 2.1"

- **Speed of recalculation:** Although the usage of a support graph definitely improved the speed of recalculation in some instances, our SupportCalc application still does not perform nearly as fast as OpenOffice Calc and especially as Excel. The limiting factor in recalculation speed varies from sheet to sheet, being either the creation of the topological list or the evaluation of the individual cells. The evaluation of cells is as such not a part of the support graph, but the underlying spreadsheet application, CoreCalc[49], and is therefore not actually the focus of this thesis.

- **Compactness of support graph:** The compactness of the support graph is excellent; in most cases the memory increase was barely noticeable. However, some expression constructs leads to poor sharing of parts of the support graph[50], possibly resulting in poorer compactness. Still, when compared to Excel and OpenOffice Calc, SupportCalc uses more memory, but this is caused by the underlying spreadsheet application – not the SupportCalc extensions.

- **Precision of support graph:** SupportCalc uses an over-approximation of the number of cells that are to be re-evaluated. Instead of only updating those cells that depend dynamically on changed cells, all the cells that statically depend either directly or indirectly on the originally changed cells are re-evaluated. This affects the recalculation speed negatively, since more cells are re-evaluated than needed. An investigation of how many cells are unnecessary evaluated was not done on the real-life spreadsheets, but would illuminate the impact on the recalculation speed.

Although the addition of a support graph in SupportCalc is definitely an improvement in some cases compared to CoreCalc, it does not perform as well as neither Excel nor OpenOffice Calc. However, further improvements of SupportCalc identified throughout this thesis may change this. Our conclusion with respect to our problem formulation is therefore that to fully evaluate whether the support graph approach can actually achieve a minimal recalculation with similar performance to Excel and OpenOffice Calc, a number of

[49] Not the original CoreCalc mentioned in [CORE], but an extension of this as described in section 6.2. Mainly this consist of added number of functions in order to enable usage of the benchmarked spreadsheets.
[50] See section 7.2

improvements identified throughout this thesis need to be investigated. As part of our conclusion we therefore sum up this list of possible improvements as “future work”.

9 Future work

Suggested improvements to various aspects of SupportCalc are presented here. This sums up suggestions discussed throughout the thesis.

Improvements to our support graph implementation in SupportCalc:

- Prevent unnecessary recalculation by either (presented in section 3.8.4):
  - Preventing recalculation of cells that are in the topological sorted list, but whose dependencies have not changed values
  - Implementing a dynamic support graph
- Increase the compactness of the support graph by changing the FAP set representation to enable relativity independently for the start and end of the FAP set. This would enable better FAP grid list sharing in some expression constructs. (presented in section 3.7.3)
- A fast mechanism for estimating dynamically whether a full recalculation without using the support graph would be faster than using the support graph. (presented in section 3.3.1)
- Implement missing operations in the support graph, i.e. delete rows and columns and support for matrices. (described in section 5.5)
- Implement a save and load mechanism for the support graph. (described in section 7.1.3)

Also some improvements to the CoreCalc application that SupportCalc is based on have been identified:

- Implement functions more efficiently to improve the speed of recalculating the individual cell. (section 7.1.1)
- Integrate spreadsheet altering operations like insert and delete, using a Command design pattern. This would enable queuing commands and undo functionality. (section 5.2.1)

Finally, benchmarking of additional large spreadsheets in order to investigate potential performance issues related to specific spreadsheet constructions would be necessary.

10 Bibliography

10.1 Reports

[CORE]
Peter Sestoft
"A spreadsheet Core Implementation in C#", Draft version 0.29 of 2006-08-30
IT University Technical Report Series, TR-2006-91, 2006
http://www.itu.dk/people/sestoft/corecalc/

[C5]
Niels Kokholm
Peter Sestoft
"The C5 Generic Collection Library for C# and CLI", Version 1.0.1 of 2006-06-26
IT University Technical Report Series, TR-2006-76, 2006
http://www.itu.dk/research/c5/

[RCG]
Thomas S. Iversen
"Runtime code generation to speed up spreadsheet computations"
Department of Computer Science (DIKU), Universitetsparken 1, DK-2100 Copenhagen Ø, July 30, 2006

10.2 Books

[GOF]
Gamma, Erich
Helm, Richard
Johnson, Ralph
Vlissides, John
"Design patterns – Elements of Reusable Object-Oriented Software"
Addison-wesley, 1995

[PIJ]
Grand, Mark
"Patterns in Java, Volume 1"
Second edition, Wiley Publishing, 2002

10.3 Internet links

[GNU]
Gnumeric spreadsheet
http://www.gnome.org/projects/gnumeric/
Last seen the 15. January 2007

[CALC]
Calc spreadsheet (part of OpenOffice)

http://www.openoffice.org/product/calc.html
Last seen the 15. January 2007

[EXCEL]
Excel 2007
http://office.microsoft.com/en-us/excel/default.aspx
Last seen the 15. January 2007

[VMS]
VMware Server virtualization software
http://www.vmware.com/products/server/
Last seen the 15. January 2007

[EXP]
Technical article describing performance related information for Excel 2007.
http://msdn2.microsoft.com/en-us/library/aa730921.aspx
Last seen the 15. January 2007

[PE]
An application for measuring process resource usage
http://www.microsoft.com/technet/sysinternals/utilities/ProcessExplorer.mspx
Last seen the 1. February 2007

[JBDT]
An application for profiling .Net applications
http://www.jetbrains.com/profiler/

[XML]
A free library to load and represent XML documents in C#
http://www.carlosag.net/Tools/ExcelXmlWriter/Default.aspx

[LOG]
An open source logging framework
http://logging.apache.org/log4net/

11 Appendix

11.1 Project process

See chapter 3 for information regarding FAP sets

The first part of the project was used mostly on trying to find relevant literature regarding FAP sets. None was found however and therefore most ideas created or based on [C5].

During the first two/thirds of the project the support graph one of the main concerns was to normalize unions of FAP sets that where allowed to overlap, the current solution was found rather late in the process.

One of the main problems during this project was the lack of real life spreadsheet examples, especially large ones. This proved to be a problem during the benchmark phase where only two of the spreadsheets obtained where really relevant.

The project was primarily based on a process of prototyping where new ideas where quickly implemented and both tested and benchmarked. This seemed ideal for a project of six months such as this, but also leaves an implemented program with a lot of code that should be either removed or re-factored. Also since ideas where quickly implemented some errors occurred, that took some time to debug.

11.1.1 Problem formulation change

Changes to the original problem formulation must be documented as per the rules of the IT-University. The original problem formulation was in Danish since the project and report was both started in Danish. Since then the language has been changed to English and the problem formulation has been made more specific to our problem. The original problem formulation in Danish was:

> "Undersøgelse af hvorvidt en specifik ide til en support graf til regneark vil forbedre ydelsen deraf"

This translates loosely into:

> "An investigation of a specific idea of using a support graph in spreadsheets to improve performance"

However this was very vague in regards to what we mean by performance. We decided to define performance as the speed of recalculation, compactness of the support graph and the precision of the support graph. Furthermore we decided to benchmark against the spreadsheet applications Excel 2007 and OpenOffice's Calc. Our current problem formulation has been changed to:

> “Design and develop a support graph as an extension and thereby improvement of a spreadsheet application, with focus on the speed of recalculation, compactness and precision. The goal is to achieve similar performance to that of Excel 2007 and OpenOffice Calc 2.1"

11.2 Spreadsheets used for testing and profiling

Here are the abbreviations and full names for the spreadsheets used in this project. If no details regarding the content of a sheet are given here it is given in section 6.2.4.

PROT:
Filename: phipsi2.xml

ROLE:
Filename: FREE character creation a023.xml

PROBA:
Filename: Probability.xml

ALPHA:
Filename: Alpha.xml

FINAN:
Filename: co2regnskab.xml

CONST1:
Filename: BookSupportGraphTest-Semi-Large-SimpleX7.xml
Notes: This is a constructed large spreadsheet used in the start phase of the project.

11.3 Profiling results

Here are the profiling results that where used during the development of the different versions.

11.3.1 Mark I to Mark II profiling

The main focus between Mark I and Mark II is the reduced memory usage, but also the time of building the support graph has been reduced.

In Figure 11.1 the first profiling can be seen. This is done with a constructed spreadsheet, in order to find the main contributor to the memory usage. The biggest part of the memory is clearly used by the FAP lists in form of the "TreeSet" class. The total usage of the support graph can be seen in the "Held memory" column, under FAPDoubleList.

Classes	Objects	Memory...	Held obje...	Held memor...
22,87 % C5.**TreeSet<FAPSet>**	419.328	33.546.240	2.096.640	77.156.352
13,72 % C5.**TreeSet<FAPSet>.Node**	419.328	20.127.744	419.328	20.127.744
7,92 % C5.**ArrayList<FAPDoubleSet<Sheet>>**	161.280	11.612.160	3.209.472	109.347.840
6,86 % C5.**TreeSet<FAPSet>.Node []**	419.328	10.063.872	419.328	10.063.872
6,07 % System.**Object**	742.279	8.907.348	742.279	8.907.348
5,72 % System.**Int32 []**	419.364	8.390.368	419.364	8.390.368
5,28 % FAP.**FAPDoubleSet<Sheet> []**	161.280	7.741.440	2.886.912	95.800.320
4,74 % System.Windows.Forms.**PropertyStore.ObjectEntry []**	217.010	6.949.780	225.491	7.131.210
4,73 % System.Windows.Forms.**DataGridViewTextBoxCell**	216.988	6.943.616	871.137	24.340.028
4,73 % System.Windows.Forms.**PropertyStore.IntegerEntry []**	216.818	6.939.096	216.818	6.939.096
4,57 % FAP.**FAPList**	419.328	6.709.248	2.515.968	83.865.600
3,43 % CoreCalc.**Formula**	209.703	5.032.872	209.729	5.033.360
2,86 % FAP.**FAPDoubleSet<Sheet>**	209.664	4.193.280	2.725.632	88.058.880
2,39 % System.Windows.Forms.**PropertyStore**	219.214	3.507.424	660.617	17.563.234
1,76 % FAP.**FAPDoubleList<Sheet>**	161.280	2.580.480	3.532.032	113.863.680
0,63 % System.**Object []**	1.134	927.676	882.111	25.568.944
0,59 % FAP.**FAPDoubleList<Sheet>** [,]	4	863.368	3.532.036	114.727.048

Figure 11.1:CONST1, Mark I

An attempt was done to decrease the memory usage by avoiding using the FAPList and thereby the "TreeSet" when only one item in them. This proved out not to work as can be seen in Figure 11.2.

Classes	Objects	Memory...	Held obje...	Held memor...
26,72 % FAP.**FAPSet []**	419.328	45.287.424	419.328	45.287.424
14,84 % C5.**SortedArray<FAPSet>**	419.328	25.159.680	1.257.984	75.479.040
6,85 % C5.**ArrayList<FAPDoubleSet<Sheet>>**	161.280	11.612.160	3.209.472	119.411.712
5,70 % System.**String**	217.984	9.662.244	217.984	9.662.244
5,26 % System.**Object**	742.281	8.907.372	742.281	8.907.372
4,57 % FAP.**FAPDoubleSet<Sheet> []**	161.280	7.741.440	2.886.912	105.864.192
4,10 % System.Windows.Forms.**PropertyStore.ObjectEntry []**	217.015	6.950.080	435.208	16.549.310
4,10 % System.Windows.Forms.**DataGridViewTextBoxCell**	216.988	6.943.616	1.080.839	33.757.608
4,09 % System.Windows.Forms.**PropertyStore.IntegerEntry []**	216.823	6.939.236	216.823	6.939.236
3,96 % FAP.DynamicTree.**DynamicTreeSet**	419.328	6.709.248	2.096.640	87.220.224
3,96 % FAP.**FAPList**	419.328	6.709.248	2.515.968	93.929.472
2,97 % CoreCalc.**Formula**	209.703	5.032.872	419.432	8.388.608
2,97 % FAP.DynamicTree.**ArrayTreeSet**	419.328	5.031.936	1.677.312	80.510.976
2,47 % FAP.**FAPDoubleSet<Sheet>**	209.664	4.193.280	2.725.632	98.122.752
2,07 % System.Windows.Forms.**PropertyStore**	219.219	3.507.504	870.344	26.981.554
2,03 % CoreCalc.**NumberValue**	215.085	3.441.360	215.085	3.441.360
1,52 % FAP.**FAPDoubleList<Sheet>**	161.280	2.580.480	3.532.032	123.927.552
0,54 % System.**Object []**	1.028	921.772	1.091.707	34.980.402
0,51 % FAP.**FAPDoubleList<Sheet>** [,]	4	863.368	3.532.036	124.790.920
0,51 % CoreCalc.**Cell** [,]	4	863.368	430.194	9.402.588
0,04 % CoreCalc.**NumberCell**	5.379	64.548	10.758	150.612
0,02 % System.Windows.Forms.**DataGridViewRow**	894	39.336	1.089.164	34.841.658
0,02 % System.Windows.Forms.**DataGridViewRowHeaderCell**	890	28.480	3.560	92.206
0,02 % System.Windows.Forms.**DataGridViewTextBoxColumn**	288	26.496	1.488	52.494
0.01 % System.Collections.**ArrayList**	987	23.688	1.083.695	34.706.126

Figure 11.2: CONST1, Mark I with changes

Later profiling was done, this time with a real life example, which can be seen in Figure 11.3.

Classes	Objects	Memory...	Held obje...	Held memor...
49,50 % C5.TreeSet<FAPSet>.Node	3.326.845	159.688.560	3.326.845	159.688.560
12,79 % C5.TreeSet<FAPSet>	515.754	41.260.320	5.389.861	241.045.776
6,30 % C5.TreeSet<FAPSet>.Node []	515.754	20.331.320	515.754	20.331.320
5,59 % System.Windows.Forms.DataGridViewTextBoxCell	563.316	18.026.112	1.688.526	45.019.074
5,57 % System.Windows.Forms.PropertyStore.IntegerEntry []	561.752	17.977.624	561.752	17.977.624
4,21 % System.Int32 []	515.791	13.580.560	515.791	13.580.560
2,83 % System.Windows.Forms.PropertyStore	570.472	9.127.552	1.143.771	27.456.300
1,92 % System.Object	516.147	6.193.764	516.147	6.193.764
1,92 % System.Collections.Generic.LinkedListNode<FAPDoubleSet<S	257.877	6.189.048	6.421.369	258.581.412
1,92 % FAP.FAPList	515.754	6.189.048	5.905.615	247.234.824
1,60 % FAP.FAPDoubleSet<Sheet>	257.877	5.157.540	6.163.492	252.392.364
1,21 % System.Collections.Generic.LinkedList<FAPDoubleSet<Sheet	139.696	3.911.488	6.561.065	262.492.900
0,75 % CoreCalc.Formula	100.941	2.422.584	229.559	4.657.820
0,73 % System.Object []	2.736	2.360.072	1.720.353	48.277.388
0,69 % FAP.FAPDoubleList<Sheet> [,]	11	2.236.292	6.700.772	266.405.544
0,69 % CoreCalc.Cell [,]	11	2.236.292	259.894	7.425.764
0,52 % FAP.FAPDoubleList<Sheet>	139.696	1.676.352	6.700.761	264.169.252
0,39 % CoreCalc.NumberValue	79.347	1.269.552	79.347	1.269.552
0,30 % System.String	28.575	969.494	28.575	969.494

Figure 11.3: ROLE, Mark I

The approach with relative FAP lists and shared FAP grid lists was implemented and a new version Mark II was born. A profiling result is shown in Figure 11.4, where the overall memory usage is significant decreased when compared to the same sheet in Mark I.

Classes	Objects	Memory...	Held obje...	Held memor...
33,44 % C5.TreeSet<FAPSet>.Node	765.364	36.737.472	765.364	36.737.472
16,41 % System.Windows.Forms.DataGridViewTextBoxCell	563.316	18.026.112	1.688.525	45.019.050
16,37 % System.Windows.Forms.PropertyStore.IntegerEntry []	561.792	17.979.004	561.792	17.979.004
8,31 % System.Windows.Forms.PropertyStore	570.512	9.128.192	1.144.005	27.463.788
5,83 % C5.TreeSet<FAPSet>	80.070	6.405.600	1.085.644	47.723.440
2,21 % CoreCalc.Formula	100.941	2.422.584	229.559	4.657.820
2,15 % System.Object []	2.732	2.359.816	1.720.372	48.277.838
2,04 % FAP.FAPDoubleList<Sheet> [,]	11	2.236.292	1.277.375	53.633.992
2,04 % CoreCalc.Cell [,]	11	2.236.292	259.894	7.425.764
1,81 % C5.TreeSet<FAPSet>.Node []	80.070	1.988.312	80.070	1.988.312
1,49 % System.Int32 []	80.109	1.635.304	80.109	1.635.304
1,17 % FAP.FAPList	80.070	1.281.120	1.165.714	49.004.560
1,16 % CoreCalc.NumberValue	79.347	1.269.552	79.347	1.269.552
0,88 % System.String	28.594	971.408	28.594	971.408
0,88 % System.Object	80.486	965.832	80.486	965.832
0,87 % System.Collections.Generic.LinkedListNode<FAPDoubleSet<Sheet>>	40.035	960.840	1.245.784	50.766.100
0,73 % FAP.FAPDoubleSet<Sheet>	40.035	800.700	1.205.749	49.805.260
0,40 % System.Collections.Generic.LinkedList<FAPDoubleSet<Sheet>>	15.790	442.120	1.261.574	51.208.220
0,38 % CoreCalc.TextValue	35.173	422.076	56.989	1.158.328
0,18 % System.Windows.Forms.PropertyStore.ObjectEntry []	5.625	192.580	14.229	397.040
0,17 % FAP.FAPDoubleList<Sheet>	15.790	189.480	1.277.364	51.397.700
0,15 % System.Windows.Forms.DataGridViewTextBoxColumn	1.819	167.348	9.143	324.616
0,10 % System.Windows.Forms.DataGridViewRow	2.523	111.012	1.710.014	47.829.434
0,07 % CoreCalc.TextCell	6.747	80.964	20.108	388.628

Figure 11.4: ROLE, Mark II

The last figure shows the how the processor usage is distributed during the building of the support graph from scratch in Mark I. It shows that the "Union", "AddAll" and the FAP list constructor methods that are used to add a FAP grid to an existing FAP grid list takes quite some time. Since one of the advantages of mark II is reusing of the calculation already done, the number of method calls to these methods would most likely be greatly decreased. A profiling of this was not done since at this stage as the relatively slow speed of

recalculation was the problem with highest priority. However in future development where memory is the focus a new memory profiling might be very beneficial.

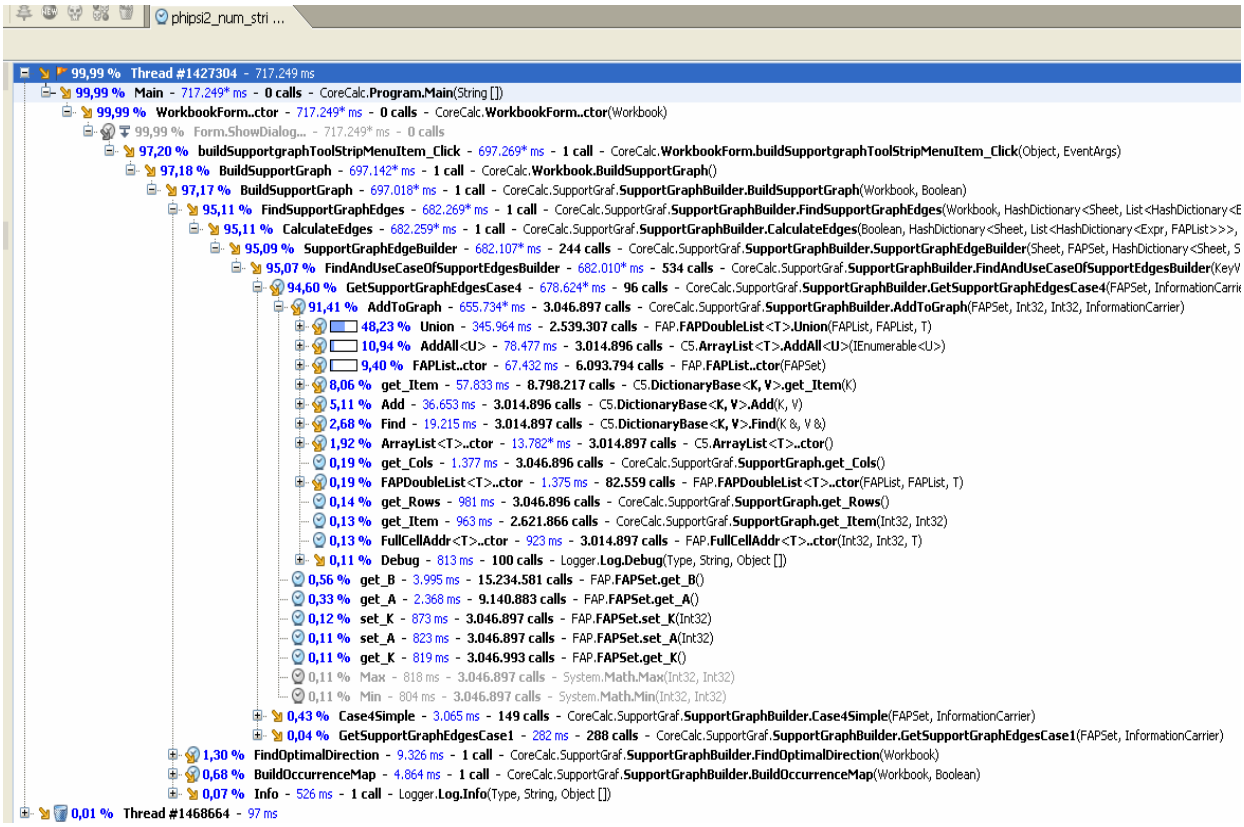


Figure 11.5: PROT sheet, Mark I

11.3.2Mark II to Mark III

The difference between Mark II and Mark III is the speed of recalculation, which is improved by improving the topological sorting. As can be seen on Figure 11.6 the single function call that takes the most time is the "Contains" method of the "HashLinkedList" that represent the topological list.

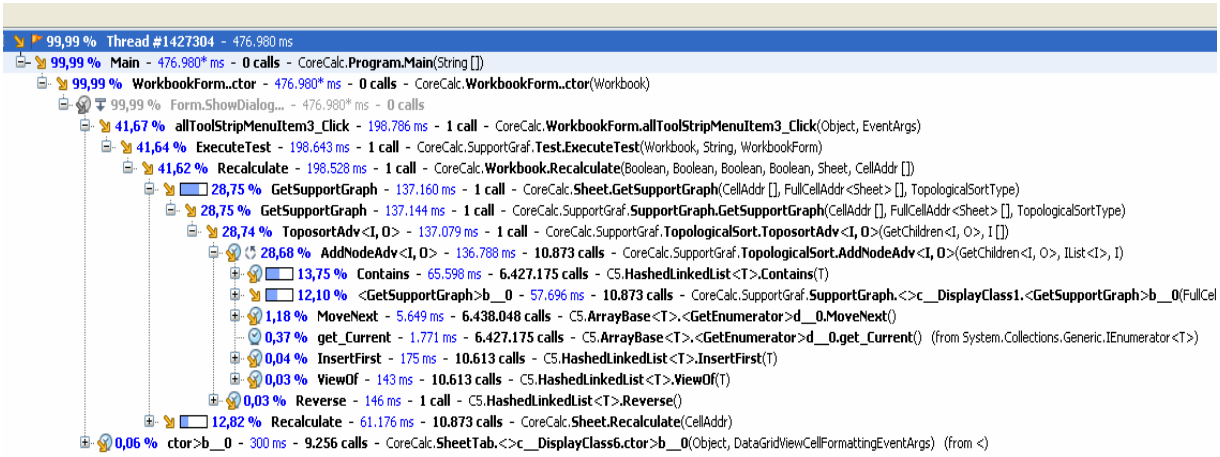


Figure 11.6: ROLE recalculate 10%, Mark II

Instead of checking if an element was in the list, this was changed so a bool 2 dimensional array was used to lookup if an element was already contained in the list.

In Figure 11.7 the result of the above change can be seen. Although this is considerably faster there is still a slight bottleneck in the "GetAdresses" method in the "FAPDoubleSet" class that represents FAP grids. This was eliminated by allowing the "FAPDoubleSet" direct access to the bool array indicating if a cell address was already in the list, thereby avoiding it to put elements in a temporary list when it was not needed.

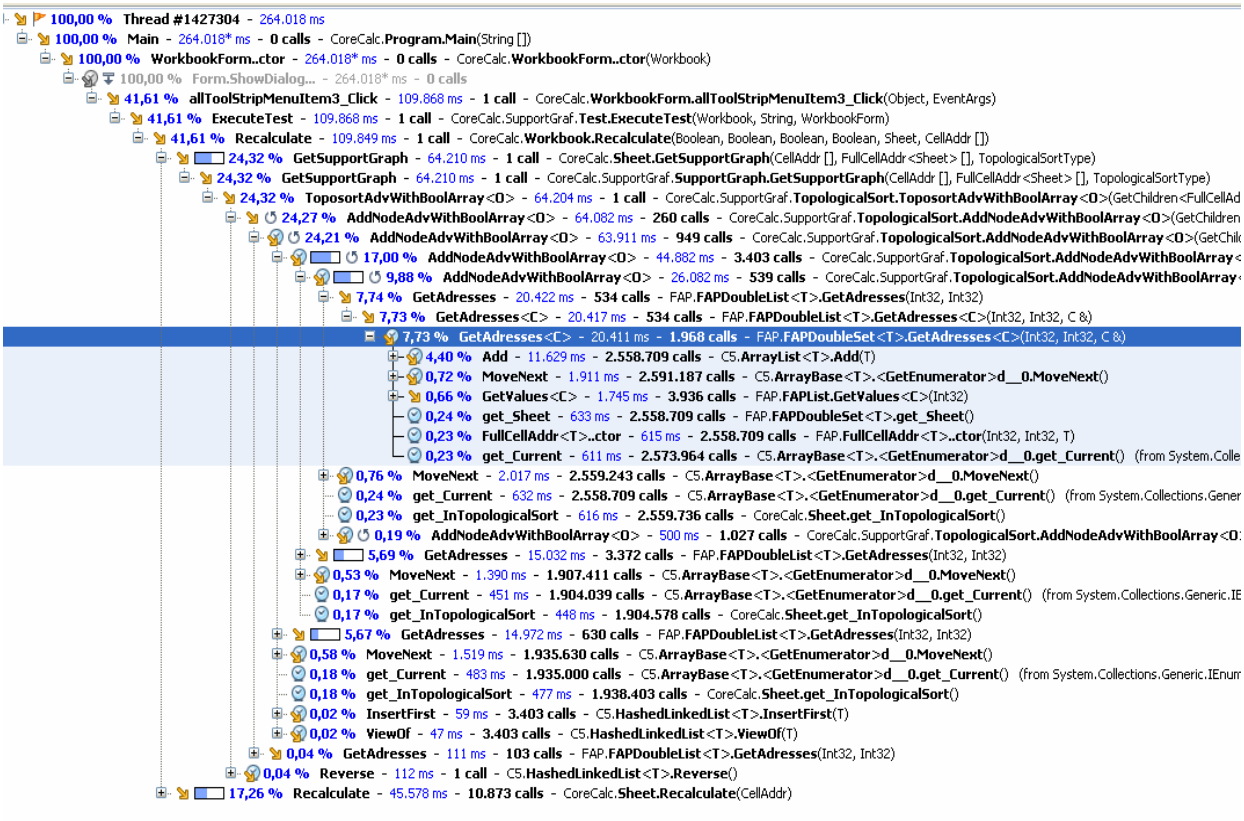


Figure 11.7: ROLE recalculate 10%, Mark II½ (boolean array used)

Figure 11.8 shows the recalculation with the final Mark III version. Here no obvious bottleneck is present.

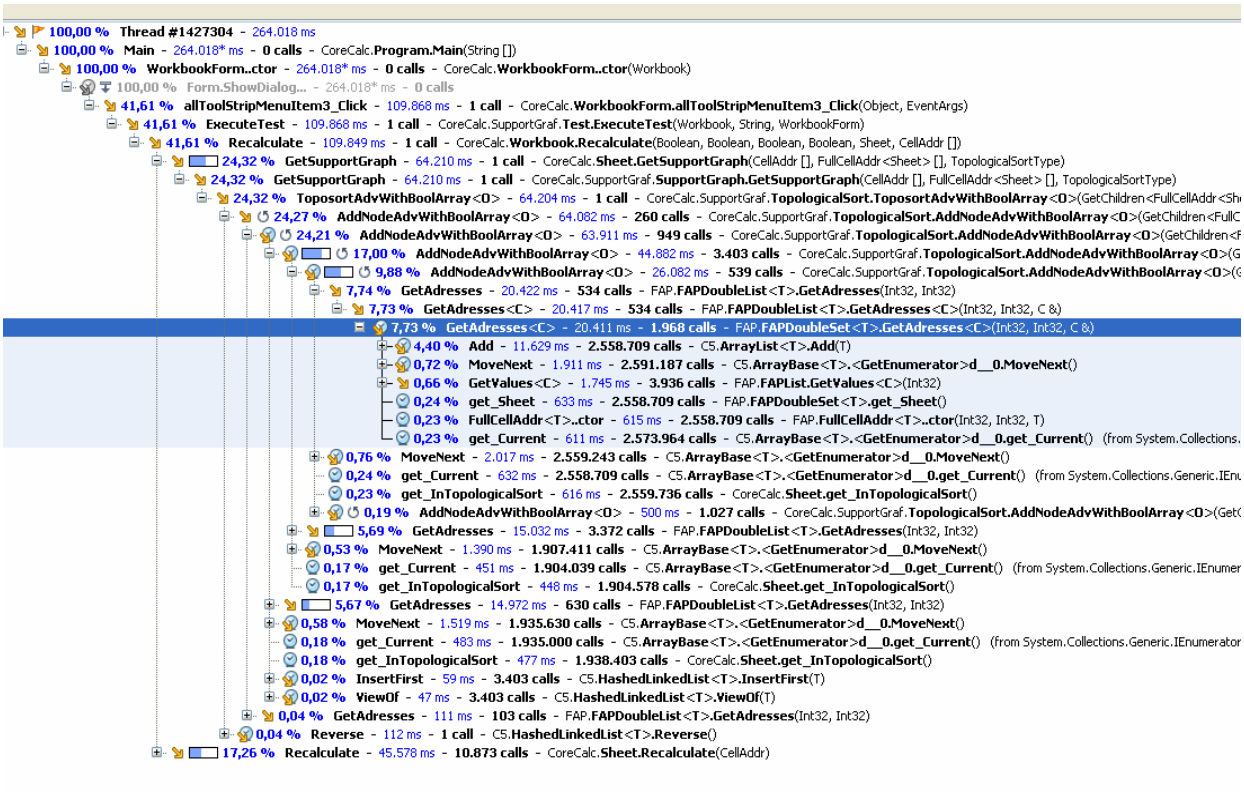


Figure 11.8: ROLE recalculate 10%, Mark III

11.3.3 Profiling performed after benchmarks

Profiling was done after the benchmarks were performed in order to confirm a suspicion that the evaluation of the added implemented functions to CoreCalc was the primary contributor to the recalculation time.

Figure 11.9: PROT investigation using the 5% Mark III benchmark

As can be seen in Figure 11.9, the CriteriaHolds() and GetMatrixValue() methods take around 65% of the recalculation time. CriteriaHolds()is a helper method to our COUNTIF() implemented function. GetMatrixValue() is used both by the COUNTIF() and LOOKUP() functions, again implemented by us. An optimization of these two non-strict functions is needed in order to get faster recalculation times in the PROT sheet.

11.4 Excel and Calc benchmark script usage

Scripts used for the Excel and Calc recalculation benchmarks are located on the CD-ROM. Both scripts contain a global parameter which defines how many runs of each benchmark are to be performed.

11.4.1Excel

Invoking “InitFullGeneric()” execute a full recalculation of the whole sheet. “InitAllGeneric()” changes all cells in the sheet according to the rules defined in section 6.2.5. The rest of the benchmarks are specific for each sheet. An example for the FINAN sheet is:

- InitFinan1PRocent()
- InitFinan5PRocent()

- InitFinan10PRocent()

The rest of the sheet follows the same naming convention.

11.4.2 Calc

As in section 11.4.1, but “InitFullGeneric()” is called “RecalculateFullGeneric” and “InitAllGeneric()” is called “RecalculateAllGeneric()”.

11.5 Benchmark result interpretation

Only examples of benchmark results are shown and a short explanation of the results. The benchmark results are located on the CD in the folder "/benchmark results".

11.5.1 Compactness related result example

Showing memory related information:

Workbook name: C:\Documents and Settings\Administrator\Desktop\support graph\kode\Snippets\FAP\FAP\CoreCalc\Rigtige regnearks eksempler\Fra Peter\FFRE character creation a023.xml
Total number of cells in sheets: 558265
Total number of non-null cells in sheets: 112754
FAPSet count: 3221850
Actual FAP set count (reuse in effect): 3221850
FAPDoubleSet (FAPGrid) count: 240435
Number of cells with supportgraph (FAPDoubleList): 139696
FAPSets/FAPGrid: 13,4000873416932
FAPSets / Number of cells with supportgraph: 23,063294582522
Number of actual FAPSets / Number of actual cells with supportgraph: 23,063294582522
Edges count: 239158945
Real edgecount: 239152019
FAPSets/Edges:0,0134719749114892
FAPSets (reuse in effect)/Edges:0,0134719749114892
Highest number of FAPSets for a single Grid:
113

Size of last topological sort: 0
Number of cells containing volatile expressions: 0
Number of cells waiting for a manual update: 0
Number of times a FAPDoubleList have been reused: 0

Notice the terminology is slightly different than from the report. FAPDoubleSet is the same as a FAP grid, while FAPDoubleList is the same as a FAP grid list.
The workbook name is the full path to the workbook loaded in the benchmark. Actual FAP set count shows how many FAP sets we have in the support graph when we consider of FAP grid lists. Number of cells with a support graph is the same as how many cells support another cell in the sheet. FAPSets / FAPGrid shows the average complexity of FAP grids. The ratio of FAPSets / Number of cells with support graph shows the average number of FAP sets a single cells support graph contains. The entry below shows the same, but takes into account FAP grid list sharing. “Edges count” shows an estimated edge count which is fast to perform compared to the “Real edgecount” which takes into account possible

overlaps of FAP grids as well as sharing of FAP grid lists caused by relative FAP sets as described in section 3.7.
“Size of last topological” sort shows how many cells were involved in the last topological sorting. “Number of cells waiting for a manual update” is how many have been changed without recalculating (used when postponing recalculation until all desired cells have been edited).

In the case of the Mark III Compactness of support graph benchmark we also include how long it took to share all FAP grid lists across columns:

```
Remove duplicate FAP sets across columns took: 00:00:00.0940940
```

11.5.2 Speed of recalculation result example

SupportCalc

```
Doing the topological sort and thereby finding the support graph took: 00:00:00.0001861
Recalculation with supportgraph took 00:00:00.0102337
….
Doing the topological sort and thereby finding the support graph took: 00:00:00.0008219
Recalculation with supportgraph took 00:00:00.0110469
Average of the 10 runs (seconds): 00:00:00.0106446
```

The first entry measures how long it takes to perform the actual topological sorting when using the support graph in a single recalculation. The measurement should be read as follows:

```
Hours:Minutes:Seconds.Milliseconds
```

The second entry shows the total time for the whole recalculation. An average of 10 runs is presented in the end. In the Mark III benchmark results we show the time it took to the evaluation of cells in addition to the above data:

```
Recalcing took: 00:00:37.1907393
```

Excel and Calc

Results from Excel and Calc are less detailed since we do not have access to the inner workings of Excel and do not look at the source code of Calc. An example from Excel is shown:

```
0,00823 Seconds
….
0,00791 Seconds
Average (milliseconds): 14,4
```

Each individual recalculation measurement is shown and the average of the runs in the end. Results from Calc are similar.

11.5.3 Memory usage result example

After load:

Private bytes: 33.836K
Private working set: 20.116K

After build:

Private bytes: 35.940K
Private working set: 22.340K

After load is the memory usage after loading the spreadsheet while after build is after the support graph in SupportCalc is built. For Excel and Calc we only measure “After load”. “Private bytes” are the combined physical and virtual bytes the spreadsheet application has been allocated which can not be shared among other processes. The “private working set” is the amount of physical memory allocated to the spreadsheet application which can not be shared among other processes.

11.5.4 Degradation result example

Showing memory related information before:
…

op1 execute
…
op4 execute
op4 undo
…
op1 undo

Showing memory related information after:
…

We show memory related information before and after we execute the support graph altering operations in the same format as shown in section 11.5.1. “opN” is the Nth operation, each with an execute and undo command. Undo commands are done in reverse order as shown. The specific operations are described in the report and can also be seen in the “Test” class.

11.5.5 Speed of support graph maintenance

Test 3 (multiple, with support graph, with area)

INSERT
Run took: 00:00:00.0613818
…
Run took: 00:00:00.0822744
Average of the 10 executes: 00:00:00.0664548

DELETE
Run took: 00:00:00.0614960
…
Run took: 00:00:00.0678868
Average of the 10 executes: 00:00:00.0680969

The name of the benchmark reveals which specific benchmark was executed. “Multiple” means that 1% of the non-null cells were involved in the execution. It could also say “Single” if only a single cell was included. “with support graph” indicates whether the support graph was used or not. The optional “with area” specifies if the area optimized methods are used in the execution. If it is not present, the un-optimized methods are used in the execution.

12 SupportCalc source code

In this appendix the source code for SupportCalc can be found. It is divided into two sections; the first contains code fully developed by us and the second section contains code modified by us in order to integrate the support graph into CoreCalc. The original CoreCalc can be found on the homepage found in [CORE].

New classes

ISheet
FullCellAdrArrayList
FullCellAddr
FAPSet
FAPDoubleSet
FAPDoubleList
FAPList
Tools
MathFunc
Excel2003XMLReader
AddressCollector
SupportGraph
SupportGraphBuilder
Test
TopologicalSort

Modified classes of CoreCalc

Spreadsheet.ATG
GUI
WorkbookIO
Cells
Functions
Sheet
Types
Workbook
CellAddressing
Expressions
Program
Values

FAP

ISheet

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace FAP
6 {
7     public interface ISheet<T> : IComparable<T>
8     {
9         bool[,] InTopologicalSort{get;}
10    }
11 }
12
```

FAP

FullCellAddrArrayList

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;

namespace FAP
{
    public class FullCellAddrArrayList<T>:IEnumerable where T:class
    {
        private const int defaultStartSize = 1000;

        private FullCellAddr<T>[] internalArray;

        private int count;

        private int size
        {
            get {return internalArray.Length;}
        }

        public int Count
        {
            get { return count; }
        }

        public FullCellAddr<T> this[int i]
        {
            get { return internalArray[i]; }
            set
            {
                internalArray[i] = value;
            }
        }

        public FullCellAddrArrayList()
        {
            internalArray = new FullCellAddr<T>[defaultStartSize];
            count = 0;
        }

        public void Add(FullCellAddr<T> add)
        {
            if (size == count)
                doubleSize();

            internalArray[count] = add;
            count++;

        }

        private void doubleSize()
```

```
52          {
53              FullCellAddr<T>[] temp = new FullCellAddr<T>[size*2];
54              for (int i = 0; i < internalArray.Length; i++)
55              {
56                  temp[i] = internalArray[i];
57              }
58              internalArray = temp;
59          }
60
61          #region IEnumerable Members
62
63          System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()
64          {
65              return internalArray.GetEnumerator();
66          }
67
68          #endregion
69
70          public FullCellAddr<T>[] Reverse()
71          {
72              FullCellAddr<T>[] back = new FullCellAddr<T>[Count];
73
74              for (int i = 0; i < Count; i++)
75              {
76                  back[Count - i - 1] = internalArray[i];
77              }
78
79              return back;
80          }
81      }
82  }
83
```

FAP

FullCellAddr

```
using System;
using System.Text;

namespace FAP
{
    /// <summary>
    /// A cell address, with a reference to the containing sheet
    /// </summary>
    public struct FullCellAddr<T>: IEquatable<FullCellAddr<T>> where T:class
    {
        public readonly int row, col;
        public readonly T sheet;

        public FullCellAddr int col, int row, T sheet
        {
            this.row = row;
            this.col = col;
            this.sheet = sheet;
        }

        #region IEquatable<FullCellAddr<T>> Members

        public bool Equals FullCellAddr<T> other
        {
            return  row == other.row && col == other.col && sheet == other.sheet ;
        }

        #endregion

        public override int GetHashCode
        {
            int result = row;
            result =  9*result + col;
            result =  9*result + sheet.GetHashCode  ;
            return result;
        }

        public override bool Equals object obj
        {
            if  ReferenceEquals this, obj   return true;
            if  ! obj is FullCellAddr<T>   return false;
            return Equals  FullCellAddr<T>  obj ;
        }

        public override string ToString
        {
            StringBuilder sb = new StringBuilder  ;
            sb.AppendLine "Sheet: " + sheet.ToString   ;
            sb.Append "Position: C" + col .AppendLine "R" + row ;
            return sb.ToString  ;
        }
```

```
52          }
53  }
54
```

FAP

FAPSet

```
using System;
using System.Collections.Generic;
using System.Text;
using C5;
using FAP;
using SpecialMath;

namespace FAP
{
    /// <summary>
    /// Describes one sequence.
    /// </summary>
    public struct FAPSet : IEquatable<FAPSet>, IComparable<FAPSet>
    {
        public static readonly FAPSet EMPTY = new FAPSet(0, 0, 0);
        public static readonly FAPSet MAX = new FAPSet(int.MaxValue, int.MaxValue, int.MaxValue);
        public static readonly FAPSet MIN = new FAPSet(0,0,0);

        private int a;
        private int b;
        private int k;


        #region Properties for a b and k
        public int B
        {
            get { return b; }
            set {
                b = value; }
        }

        public int K
        {
            get { return k; }
            set
            {
                k = value;
            }
        }

        public int A
        {
            get { return a; }
            set { a = value; }
        }

        #endregion

        public FAPSet(int a, int b, int k)
        {
            this.a = a;
```

```
52              this.b = b;
53              this.k = k;
54          }
55
56          public void ZipMultiple<C>(int commonMultiple, ref C sets)where C:C5.ICollection<FAPSet>
57          {
58              int n = commonMultiple / b;
59
60              for (int i = 0; i < n; i++)
61              {
62                  int ki = (k - 1 - i + n) / n;
63                  if (ki != 0)
64                  {
65                      sets.Add(new FAPSet((A + i * b), commonMultiple, ki));
66                  }
67              }
68          }
69
70          public override string ToString()
71          {
72              return "(" + A + "," + B + "," + K + ")";
73          }
74
75          /// <summary>
76          /// Divides a FAPSet into two new sets, dividing them at a specific K value. This set will become the low set
77          /// </summary>
78          /// <param name="chopAt">The K value to divide between. The low set (lower a and k value)
79          /// will get the chopAt as new K value</param>
80          /// <returns>The high set</returns>
81          /// <remarks>The used formula is: (a,b,chopAt + k2) = (a,b, chopAt) U (a+b*chopAt, b, k2)
82          /// Where the first set is the incomming set, the second is the referenced set and the last
83          /// set is the returned set</remarks>
84          internal void Chop(int chopAt,out FAPSet low,out FAPSet high)
85          {
86              high = new FAPSet(A + B * chopAt, B, K - chopAt);
87              low = new FAPSet(A, B , chopAt);
88          }
89
90          public FAPSet Delete(int positionToRemove)
91          {
92              FAPSet newSet = new FAPSet(A, B, K);
93              if (A <= (A + positionToRemove * B) && (A + positionToRemove * B) <= (A + K * B))
94              {
95                  int newK = (positionToRemove - A) / B;
96                  if (K == 2)
97                  {
98                      if (A + B == positionToRemove) newSet = new FAPSet(A + B, B, 1);
99                      else newSet = new FAPSet(A + B * 2, B, 1);
100                 }
101                 else if (K >= 3) newSet = new FAPSet(A + B * positionToRemove + B, B, K - newK);
102                 K = newK;
```

```
            }
            return newSet;
        }

        #region operatorAndEquality

        public override int GetHashCode()
        {
            int result = a;
            result = 29 * result + b;
            result = 29 * result + k;
            return result;
        }

        public override bool Equals(object obj)
        {
            if (ReferenceEquals(this, obj)) return true;
            if (!(obj is FAPSet)) return false;
            FAPSet fapSet = (FAPSet)obj;
            if (a != fapSet.a) return false;
            if (b != fapSet.b) return false;
            if (k != fapSet.k) return false;
            return true;
        }

        public bool Equals(FAPSet other)
        {
            return (A == other.A && b == other.b && K == other.K);
        }

        public static bool operator ==(FAPSet a, FAPSet b)
        {
            return a.Equals(b);
        }

        public static bool operator !=(FAPSet a, FAPSet b)
        {
            return !a.Equals(b);
        }

        public static bool operator <(FAPSet a, FAPSet b)
        {
            return 0 < a.CompareTo(b);
        }

        public static bool operator >(FAPSet a, FAPSet b)
        {
            return a.CompareTo(b) < 0;
        }

        #endregion
```

```
154
155         #region IComparable<FAPSet> Members
156
157         public int CompareTo(FAPSet other)
158         {
159             if (A.CompareTo(other.A) != 0)
160                 return A.CompareTo(other.A);
161             else if (B.CompareTo(other.B) != 0)
162                 return B.CompareTo(other.B);
163             else
164                 return K.CompareTo(other.K);
165         }
166
167         #endregion
168
169         public int MaxValue()
170         {
171             return a + b*(k - 1);
172         }
173
174         public void GetValues<C>(ref C valueList)where C:C5.ICollection<int>, new()
175         {
176             GetValues(0, ref valueList);
177         }
178
179         public void GetValues<C>(int relative, ref C valueList)where C:C5.ICollection<int>
180         {
181             for (int i = 0; i < k; i++)
182             {
183                 valueList.Add(a + i * b+relative);
184             }
185
186         }
187
188         public int GetEdgeCount()
189         {
190             return k;
191         }
192
193
194         public bool Contains(int r)
195         {
196             return (r >= A && (r - A) % B == 0 && (r - A) / B < K);
197         }
198
199         /// <summary>
200         /// Describes if the area from start to end (both inclusive) covers or partly covers the fap set
201         /// </summary>
202         /// <param name="start"></param>
203         /// <param name="end"></param>
204         /// <returns></returns>
```

```
205         internal bool Contains(int start, int end)
206         {
207             return Covers(start, end)  || (A <= end && A + B*(K - 1) >= end) || (A + B*(K - 1) >= start && A <= start);
208         }
209 
210         internal bool Overlap(FAPSet other)
211         {
212             if (other.A < A)
213             {
214                 return other.A + other.B * (other.K - 1) >= A;
215             }
216             else
217             {
218                 return A + B * (K - 1) >= other.A;
219             }
220         }
221 
222 
223         internal bool Covers(int start, int end)
224         {
225             return (A >= start && A + B*(k - 1) <= end);
226         }
227 
228         /// <summary>
229         /// Determinate if this set represent at least all the ints in minor
230         /// </summary>
231         internal bool BelongsToOrContinuation(FAPSet minor, out FAPSet representing)
232         {
233             if ((minor.B == B || minor.K == 1) && (minor.A - A) % B == 0)
234             {
235                 int x = (minor.A - A) / B;
236                 if (x >= 0 && minor.K < K - x)
237                     representing = this;
238                 else
239                     representing = new FAPSet(A, B, minor.K + x);
240 
241                 return true;
242             }
243             else
244             {
245                 representing = EMPTY;
246                 return false;
247             }
248         }
249 
250         internal bool BelongsToOrContinuation(FAPSet minor)
251         {
252             FAPSet nothing;
253             return BelongsToOrContinuation(minor, out nothing);
254         }
255 
```

```
256
257         internal void RemoveFAPSet(FAPSet remove, FAPList newSets)
258         {
259             TreeSet<int> removeInts = new TreeSet<int>();
260
261             FAPSetRemoveComparer comparer = new FAPSetRemoveComparer();
262             TreeSet<FAPSet> thisZipMultiple = new TreeSet<FAPSet>(comparer);
263             ArrayList<FAPSet> otherZipMultiple = new ArrayList<FAPSet>();
264             int lcm = SpecialMath.MathFunc.LowestCommonMultiple(B, remove.B);
265             this.ZipMultiple(lcm, ref thisZipMultiple);
266             remove.ZipMultiple(lcm, ref otherZipMultiple);
267
268             foreach (FAPSet set in otherZipMultiple)
269             {
270                 FAPSet start = new FAPSet(set.A, 0, 0);
271                 FAPSet end = new FAPSet(set.A, set.B + 1, 0);
272                 IDirectedCollectionValue<FAPSet> correctList = thisZipMultiple.RangeFromTo(start, end);
273
274                 foreach (FAPSet fapSet in correctList)
275                 {
276                     if(set.A < fapSet.A)
277                     {
278                         FindOverLapPoints(set, fapSet, newSets, removeInts);
279                     }
280                     else
281                         FindOverLapPoints(fapSet, set, newSets, removeInts);
282
283                 }
284
285                 thisZipMultiple.RemoveRangeFromTo(start, end);
286             }
287
288             foreach (FAPSet set in thisZipMultiple)
289             {
290                 newSets.Union(set);
291             }
292
293             FAPSet low;
294             FAPSet highest;
295             FAPSet current = this;
296
297             foreach (int i in removeInts)
298             {
299                 current.RemoveSingleInt(i, out low, out highest);
300                 if (highest == EMPTY)
301                 {
302                     if (low != EMPTY)
303                         newSets.Union(low);;
304
305                     return;
306                 }
```

```
307                 else
308                     newSets.Union(low);
309 
310                 current = highest;
311             }
312 
313             newSets.Union(current);
314         }
315 
316         private static void FindOverLapPoints(FAPSet lowest, FAPSet highest, FAPList newSets, TreeSet<int> removeInts)
317         {
318             int overLapFrom = (highest.A - lowest.A)/lowest.B;
319             int overLapTo = lowest.K - overLapFrom > highest.K ? highest.K : lowest.K - overLapFrom;
320 
321             if (overLapTo > 0)//then set reaches into fapSet
322             {
323                 for (int i = 0; i < overLapTo; i++)
324                 {
325                     removeInts.Add(highest.A + highest.B * i);
326                 }
327             }
328             else
329                 newSets.Union(highest);
330         }
331 
332         internal class FAPSetRemoveComparer:IComparer<FAPSet>
333         {
334             public int Compare(FAPSet x, FAPSet y)
335             {
336                 if (x.A != y.A && x.A % y.A != 0)
337                     return x.A.CompareTo(y.A);
338                 else if (x.B != y.B)
339                     return x.B.CompareTo(y.B);
340                 else if (x.K != y.K)
341                     return x.K.CompareTo(y.K);
342                 else
343                     return x.A.CompareTo(y.A);
344 
345             }
346         }
347 
348         /// <summary>
349         /// Removes a value in the fapset
350         /// </summary>
351         /// <param name="i">must be a value represented by the fapset</param>
352         /// <param name="before"></param>
353         /// <param name="after"></param>
354         internal void RemoveSingleInt(int i, out FAPSet before, out FAPSet after)
355         {
356             before = EMPTY;
357             after = EMPTY;
```

```
358
359             if(K==1)
360                 return;
361
362             int splitVal = (i - A)/B;
363
364             if(splitVal!=0)
365             {
366                 before = new FAPSet(A+B,B, splitVal);
367             }
368
369             if(splitVal != K)
370             {
371                 after = new FAPSet(i+B,B,K-(splitVal+1));
372             }
373
374         }
375
376         internal void RemoveArea(int start, int end, out FAPSet before, out FAPSet after, out FAPSet removed)
377         {
378             if(Covers(start, end))
379             {
380                 removed = this;
381                 after = EMPTY;
382                 before = EMPTY;
383                 return;
384             }
385
386             //The area end is before the end of the fap set
387             if (A <= end && A + B * (K - 1) > end)
388             {
389                 int newHighA = ((end+1) - A) % B + (end+1);
390                 int newHighK = (((A + B * (K - 1)) - newHighA) / B)+1;
391
392
393                 //The area splits the fap set into two
394                 if (A + B * (K - 1) >= start && A < start)
395                 {
396                     int newLowK = (start - 1 - A)/B+1;
397
398                     int removedA = A + B * newLowK;
399                     int removedK = K-(newHighK+newLowK);
400                     if (removedK == 0)
401                     {
402                         removed = EMPTY;
403                         before = this;
404                         after = EMPTY;
405                         return;
406                     }
407                     else
408                     {
```

```
409                         removed = new FAPSet(removedA, B, removedK);
410                         before = new FAPSet(A, B, newLowK);
411                         after = new FAPSet(newHighA, B, newHighK);
412                         return;
413                     }
414                 }
415                 else
416                 {
417                     before = EMPTY;
418                     removed = new FAPSet(A, B, K - newHighK);
419                     after = new FAPSet(newHighA, B, newHighK);
420                     return;
421                 }
422             }
423             //The area start is after the start of the fap set
424             else if (A + B * (K - 1) >= start && A <= start)
425             {
426                 int newLowK = (start - A) / B;
427                 int removedA = A + B * newLowK;
428 
429                 after = EMPTY;
430                 removed = new FAPSet(removedA, B, K-newLowK);
431                 before = new FAPSet(A, B, newLowK);
432                 return;
433             }
434             else //The area does not lie within the fap set
435             {
436                 removed = EMPTY;
437                 before = this;
438                 after = EMPTY;
439             }
440         }
441     }
442 }
443 
```

FAP

FAPDoubleSet

```
using System;
using System.Collections.Generic;
using System.Text;
using C5;

namespace FAP
{
    /// <summary>
    /// Describes two unions of sequences thereby describing a rectangle (only one)
    /// </summary>
    public class FAPDoubleSet<T>:IComparable<FAPDoubleSet<T>>, IEquatable<FAPDoubleSet<T>> where T : class, ISheet<T>
    {
        private FAPList column;
        private FAPList row;
        private T owner;

        public FAPList Column
        {
            get { return column; }
        }

        public FAPList Row
        {
            get { return row; }
        }

        public T Sheet
        {
            get { return owner; }
        }

        internal FAPDoubleSet(FAPList column, FAPList row, T sheet)
        {
            this.column = column;
            this.row = row;
            this.owner = sheet;
        }

        public bool Union(FAPDoubleSet<T> set)
        {
            if (set.Sheet != Sheet || set.Row.Absolute != Row.Absolute || set.Column.Absolute != Column.Absolute)
                return false;
            bool equalRow = set.Row.Equals(Row);
            bool equalColumn = set.Column.Equals(Column);
            if (equalRow || equalColumn)
                if (equalColumn && equalRow)
                    return true;
                else if (equalColumn)
                {
                   Row.Union(set.Row);
                   return true;
```

```
52                  }
53                  else
54                  {
55                      Column.Union(set.Column);
56                      return true;
57                  }
58              else
59                  return false;
60          }
61
62
63
64          internal void GetAdresses<C>(int cRef, int rRef, ref C adresses) where C : C5.ICollection<FullCellAddr<T>>
65          {
66              ArrayList<int> columns = column.GetValues<ArrayList<int>>(cRef);
67              ArrayList<int> rows = row.GetValues<ArrayList<int>>(rRef);
68
69              foreach (int c in columns)
70              {
71                  foreach (int r in rows)
72                  {
73                      adresses.Add(new FullCellAddr<T>(c, r, Sheet));
74                  }
75              }
76          }
77
78          internal void GetAdressesAlt(int cRef, int rRef, ref FullCellAddrArrayList<T> adresses)
79          {
80              ArrayList<int> columns = column.GetValues<ArrayList<int>>(cRef);
81              ArrayList<int> rows = row.GetValues<ArrayList<int>>(rRef);
82
83              foreach (int c in columns)
84              {
85                  foreach (int r in rows)
86                  {
87                      adresses.Add(new FullCellAddr<T>(c, r, Sheet));
88                  }
89              }
90          }
91
92          internal void GetAdressesExternCheck(int cRef, int rRef, ref ArrayList<FullCellAddr<T>> back)
93          {
94              ArrayList<int> columns = column.GetValues<ArrayList<int>>(cRef);
95              ArrayList<int> rows = row.GetValues<ArrayList<int>>(rRef);
96
97              foreach (int c in columns)
98              {
99                  foreach (int r in rows)
100                 {
101                     if (!Sheet.InTopologicalSort[c, r])
102                     {
```

```
103                     Sheet.InTopologicalSort[c, r] = true;
104                     back.Add(new FullCellAddr<T>(c, r, Sheet));
105                 }
106             }
107         }
108     }
109 
110     internal int GetEdgeCount()
111     {
112         int columnCount = column.GetValues<ArrayList<int>>(0).Count;
113         int rowCount = row.GetValues<ArrayList<int>>(0).Count;
114         return columnCount*rowCount;
115     }
116 
117     public override string ToString()
118     {
119         StringBuilder sb = new StringBuilder();
120         sb.Append("Sheet: " + owner.ToString() + "\r\n");
121         sb.Append("Col: \r\n");
122         sb.Append(column).Append("\r\n");
123         sb.Append("Row: \r\n");
124         sb.Append(row).Append("\r\n");
125         return sb.ToString();
126     }
127 
128     public string ShowFullCellAdresses(int cRef, int rRef)
129     {
130         StringBuilder sb = new StringBuilder();
131         ArrayList<FullCellAddr<T>> a = new ArrayList<FullCellAddr<T>>();
132         GetAdresses(cRef, rRef, ref a);
133         foreach (FullCellAddr<T> addr in a)
134         {
135             sb.Append(addr.sheet.ToString() + "!" + ColumnName(addr.col) + "," + (addr.row + 1) + "\r\n");
136         }
137         return sb.ToString();
138     }
139 
140     //TODO: maybe refactor this out
141     private String ColumnName(int col)
142     {
143         String name = "";
144         while (col >= 26)
145         {
146             name = (char)('A' + col % 26) + name;
147             col = col / 26 - 1;
148         }
149         return (char)('A' + col) + name;
150     }
151 
152     internal void AddLines(int lineNumber, int numberOfLines, bool isRows, int col, int row)
153     {
```

```
154             if (isRows)
155                 this.Row.AddLines(lineNumber, numberOfLines, row);
156             else
157                 this.Column.AddLines(lineNumber, numberOfLines, col);
158         }
159 
160         internal void DeleteLines(int startLineNumber, int numberOfLines, bool isRows)
161         {
162             if(isRows)
163                 Row.DeleteLines(startLineNumber, numberOfLines);
164             else
165                 Column.DeleteLines(startLineNumber, numberOfLines);
166         }
167 
168         public FAPDoubleSet<T> Copy()
169         {
170             return new FAPDoubleSet<T>(column.Copy(), row.Copy(), Sheet);
171         }
172 
173 
174         /// <summary>
175         ///
176         /// </summary>
177         /// <param name="c"></param>
178         /// <param name="r"></param>
179         /// <param name="sheet"></param>
180         /// <param name="back"></param>
181         /// <returns></returns>
182         internal void DeleteCell(int c, int r, T sheet, ref System.Collections.Generic.LinkedList<FAPDoubleSet<T>> back
183             , int relC, int relR)
184         {
185 
186             if (this.Sheet == sheet && this.column.Contains(c, relC) && this.row.Contains(r, relR))
187             {
188                 //TODO: in the contains methods we can get the FAPSets that need to be splitted
189                 //(this might be more ineffective)
190                 bool colmAbsolute = column.Absolute;
191                 FAPList columnSplit = column.Split(c, relC);
192                 this.column = columnSplit;
193                 if (columnSplit != null)
194                     back.AddLast(this);
195                 FAPList middle;
196                 if (colmAbsolute)
197                 {
198                     middle = new FAPList(c, 1, 1, true);
199                 }
200                 else
201                     middle = new FAPList(c - relC, 1, 1, false);
202 
203                 FAPList rowSplit = row.Split(r, relR);
204                 if (rowSplit != null)
```

```
205                     back.AddLast(new FAPDoubleSet<T>(middle, rowSplit, sheet));
206             }
207             else
208                 back.AddLast(this);
209         }
210 
211         internal bool DeleteArea(int startColumn, int endColumn, int startRow, int endRow, T sheet, int rCol, int rRow,
212                                  ref System.Collections.Generic.LinkedList<FAPDoubleSet<T>> altered, ref System.Collections.
    Generic.LinkedList<FAPDoubleSet<T>> normal)
213         {
214             if (this.Sheet == sheet && this.column.ContainsArea(startColumn, endColumn, rCol) && this.row.ContainsArea(startRow,
    endRow, rRow))
215             {
216                 //New columns x old rows (1st fapdoubleset) + deleted columns x new rows (2nd fapdoubleset)
217 
218                 FAPList newRows = Row.RemoveArea(startRow, endRow, rRow);
219 
220                 FAPList removedColumns;
221                 FAPList thisColumn;
222                 thisColumn = this.column.RemoveArea(startColumn, endColumn, rCol, out removedColumns);
223 
224                 if (thisColumn.Count != 0)
225                     altered.AddLast(new FAPDoubleSet<T>(thisColumn, Row.Copy(), this.Sheet));
226 
227                 if (removedColumns != null && newRows.Count != 0)
228                     altered.AddLast(new FAPDoubleSet<T>(removedColumns, newRows, this.Sheet));
229 
230 
231                 return true;
232             }
233             else
234             {
235                 normal.AddLast(this.Copy());
236                 return false;
237             }
238         }
239 
240         #region IComparable<FAPDoubleSet<T>> Members
241 
242         public int CompareTo(FAPDoubleSet<T> other)
243         {
244             int columnEq = column.Count.CompareTo(other.Column.Count);
245             if (columnEq != 0)
246                 return columnEq;
247 
248             int rowEq = Row.Count.CompareTo(other.Row.Count);
249             if (rowEq != 0)
250                 return rowEq;
251 
252             int TEq = Sheet.CompareTo(other.Sheet);
253 
```

```
254             if (TEq != 0)
255                 return TEq;
256 
257             int colMemEq = Column.CompareTo(other.Column);
258 
259             if (colMemEq != 0)
260                 return colMemEq;
261 
262             int rowMemEq = row.CompareTo(other.Row);
263 
264             return rowMemEq;
265 
266         }
267 
268         public override bool Equals(object other)
269         {
270             if(other == this) return true;
271             FAPDoubleSet<T> oth = other as FAPDoubleSet<T>;
272             if (oth != null)
273                 return Equals(oth);
274             else
275                 return false;
276         }
277 
278         public bool Equals(FAPDoubleSet<T> other)
279         {
280             if(!(column.Count == other.Column.Count))
281                 return false;
282 
283             if(!(Row.Count==other.Row.Count))
284                 return false;
285 
286             if(!(Sheet ==other.Sheet))
287                 return false;
288 
289 
290             if(!Column.Equals(other.Column))
291                 return false;
292 
293             if(!Row.Equals(other.Row))
294                 return false;
295 
296             return true;
297         }
298 
299         #endregion
300 
301         public override int GetHashCode()
302         {
303             int returnHashCode = 0;
304             returnHashCode = Column.GetAlternativeHashCode();
```

```
305             returnHashCode = returnHashCode*29 + Row.GetAlternativeHashCode();
306             returnHashCode = returnHashCode*29 + Sheet.GetHashCode();
307 
308             return returnHashCode;
309         }
310 
311 
312     }
313 }
314 
```

FAP

FAPDoubleList

```
using System;
using System.Collections;
using System.Text;
using System.Threading;
using System.Collections.Generic;


namespace FAP
{
    /// <summary>
    /// Describes a union of rectangles
    /// </summary>
    public class FAPDoubleList<T>  where T : class, ISheet<T>
    {

        private LinkedList<FAPDoubleSet<T>> fapList;

        private LinkedList<FAPDoubleSet<T>> FapList
        {
            get { return fapList; }
            //set { fabList = value; }
        }

        public FAPDoubleList()
        {
            fapList = new LinkedList<FAPDoubleSet<T>>();
        }

        public FAPDoubleList(FAPList col, FAPList row, T sheet)
            : this()
        {
            if(col.Count !=   && row.Count !=  )
                fapList.AddFirst(new FAPDoubleSet<T>(col, row, sheet));
        }

        //public FAPDoubleList(FAPSet col, FAPSet row, T sheet)
        //    : this()
        //{
        //    fapList.AddFirst(new FAPDoubleSet<T>(col, row, sheet));
        //}

        /// <summary>
        /// Adds rows or columns
        /// </summary>
        /// <param name="lineNumber">The line number which lines should be inserted before</param>
        /// <param name="numberOfLines">The number of lines to be inserted</param>
        /// <param name="isRows">True if rows should be added, false if columns</param>
        public FAPDoubleList<T> AddLines(T sheet, int lineNumber, int numberOfLines, bool isRows, int col, int row)
        {
            FAPDoubleList<T> back = this.Copy();
            foreach (FAPDoubleSet<T> set in back.fapList)
```

```
52                  {
53                      if (set.Sheet == sheet)
54                          set.AddLines(lineNumber, numberOfLines, isRows, col, row);
55                  }
56             return back;
57         }
58 
59         /// <summary>
60         /// Deletes rows or columns
61         /// </summary>
62         /// <param name="startLineNumber">The line number to start with (including)</param>
63         /// <param name="numberOfLines">Number of lines to delete</param>
64         /// <param name="isRows"></param>
65         public void DeleteLines(T sheet, int startLineNumber, int numberOfLines, bool isRows)
66         {
67 
68                 foreach (FAPDoubleSet<T> set in fapList)
69                 {
70                     if (set.Sheet == sheet)
71                         set.DeleteLines(startLineNumber, numberOfLines, isRows);
72                 }
73 
74         }
75 
76         public void DeleteCell(int c, int r, T sheet, int relC, int relR)
77         {
78                 LinkedList<FAPDoubleSet<T>> newFapSets = new LinkedList<FAPDoubleSet<T>>();
79                 foreach (FAPDoubleSet<T> set in fapList)
80                 {
81                     set.DeleteCell(c, r, sheet, ref newFapSets, relC, relR);
82                 }
83                 fapList = newFapSets;
84 
85 
86         }
87 
88         public FAPDoubleList<T> DeleteArea(int startCol, int endCol, int startRow, int endRow, T sheet, int rCol, int rRow)
89         {
90 
91             LinkedList<FAPDoubleSet<T>> existing = new LinkedList<FAPDoubleSet<T>>();
92             LinkedList<FAPDoubleSet<T>> altered = new LinkedList<FAPDoubleSet<T>>();
93 
94             foreach (FAPDoubleSet<T> set in fapList)
95             {
96                 set.DeleteArea(startCol, endCol, startRow, endRow, sheet, rCol, rRow, ref altered, ref existing);
97             }
98             FAPDoubleList<T> back = new FAPDoubleList<T>();
99             back.fapList = existing;
100 
101             foreach (FAPDoubleSet<T> set in altered)
102             {
```

```
103                 back.UnionWithOutCopy(set);
104             }
105 
106             return back;
107         }
108 
109         public FAPDoubleList<T> Union(FAPList col, FAPList row, T sheet)
110         {
111             return Union(new FAPDoubleSet<T>(col, row, sheet));
112         }
113 
114         //public FAPDoubleList<T> Union(FAPSet col, FAPSet row, T sheet)
115         //{
116         //    //TODO: this can be optimized
117         //    return Union(new FAPList(col), new FAPList(row), sheet);
118         //}
119 
120         private FAPDoubleList<T> Union(FAPDoubleSet<T> add)
121         {
122 
123             FAPDoubleList<T> back = Copy();
124 
125             back.UnionWithOutCopy(add);
126 
127             return back;
128         }
129 
130         /// <summary>
131         /// Used to make a union without making a copy of the object. Do not use this method without reading the remarks.
132         /// </summary>
133         /// <remarks>WARNING: DO NOT USE THIS METHOD EXCEPT ON A NEW COPY. This class is designed to be used when several cells in↙
        a
134         /// spreadsheet share the same object. Therefore the object will normally be copied before altering, so that the other    ↙
        cells
135         /// can still refer to the old object.</remarks>
136         /// <param name="add"></param>
137         internal void UnionWithOutCopy(FAPDoubleSet<T> add)
138         {
139             foreach (FAPDoubleSet<T> set in fapList)
140             {
141                 if (set.Equals(add))
142                     return;
143             }
144 
145 
146 
147             if (fapList.Count !=  )
148             {
149                 LinkedListNode<FAPDoubleSet<T>> current = fapList.First;
150                 do
151                 {
```

```
152                     if (current.Value.Union(add))
153                     {
154                         AlteredSet(current);
155                         return;
156                     }
157                 } while ((current = current.Next) != null);
158             }
159 
160             fapList.AddLast(add);
161         }
162 
163         private void AlteredSet(LinkedListNode<FAPDoubleSet<T>> altered)
164         {
165             LinkedListNode<FAPDoubleSet<T>> current = fapList.First;
166             do
167             {
168                 if (current != altered && current.Value.Equals(altered.Value))
169                 {
170                     fapList.Remove(altered);
171                     return;
172                 }
173             } while ((current = current.Next) != null);
174 
175 
176             current = fapList.First;
177             do
178             {
179                 if (current != altered)
180                 {
181                     if (current.Value.Union(altered.Value))
182                     {
183                         fapList.Remove(altered);
184                         AlteredSet(current);
185                     }
186                 }
187             } while ((current = current.Next) != null);
188 
189         }
190 
191         public void GetAdresses<C>(int cRef, int rRef, ref C back)where C:C5.ICollection<FullCellAddr<T>>
192         {
193 
194             foreach (FAPDoubleSet<T> set in fapList)
195             {
196                 set.GetAdresses(cRef, rRef, ref  back);
197             }
198 
199         }
200 
201         private void GetAdressesExternCheck(int cRef, int rRef, ref C5.ArrayList<FullCellAddr<T>> back)
202         {
```

```
203             foreach (FAPDoubleSet<T> set in fapList)
204             {
205                 set.GetAdressesExternCheck(cRef, rRef, ref  back);
206             }
207         }
208 
209         public void GetAdressesAlt(int cRef, int rRef, ref FullCellAddrArrayList<T> back)
210         {
211 
212             foreach (FAPDoubleSet<T> set in fapList)
213             {
214                 set.GetAdressesAlt(cRef, rRef, ref  back);
215             }
216 
217         }
218 
219         public int GetEdgeCount()
220         {
221             int edgeCount =  ;
222 
223             foreach (FAPDoubleSet<T> set in fapList)
224             {
225                 edgeCount += set.GetEdgeCount();
226             }
227             return edgeCount;
228         }
229 
230         public override int GetHashCode()
231         {
232             return fapList.GetHashCode();
233         }
234 
235         public int GetAlternativeHashCode()
236         {
237             int resultHashCode =  ;
238             foreach (FAPDoubleSet<T> set in fapList)
239             {
240                 resultHashCode += set.GetHashCode();
241             }
242 
243             return resultHashCode;
244         }
245 
246         public bool EqualsAlternative(FAPDoubleList<T> other)
247         {
248             if (other == null) return false;
249 
250             if(other.fapList.Count != this.fapList.Count)
251                 return false;
252 
253             C5.HashSet<FAPDoubleSet<T>> otherLinked = new C5.HashSet<FAPDoubleSet<T>>();
```

```
254
255             otherLinked.AddAll(other.FapList);
256
257             foreach (FAPDoubleSet<T> set in fapList)
258             {
259                 if(!otherLinked.Remove(set))
260                     return false;
261             }
262             return true;
263
264         }
265
266         public C5.ArrayList<FullCellAddr<T>> GetAdresses(int cRef, int rRef)
267         {
268             C5.ArrayList<FullCellAddr<T>> back = new C5.ArrayList<FullCellAddr<T>>();
269
270             GetAdresses(cRef, rRef, ref back);
271             return back;
272         }
273
274
275         public IEnumerable GetAdressesExternCheck(int col, int row)
276         {
277             C5.ArrayList<FullCellAddr<T>> back = new C5.ArrayList<FullCellAddr<T>>();
278
279             GetAdressesExternCheck(col, row, ref back);
280             return back;
281         }
282
283
284
285         public FullCellAddrArrayList<T> GetAdressesAlt(int cRef, int rRef)
286         {
287             FullCellAddrArrayList<T> back = new FullCellAddrArrayList<T>();
288
289             GetAdressesAlt(cRef, rRef, ref back);
290             return back;
291         }
292
293         /// <summary>
294         /// Makes a copy of the FAPDoubleList. Later calls to any of the FAPDoubleList,
295         /// will not modify the other
296         /// </summary>
297         /// <returns></returns>
298         public FAPDoubleList<T> Copy()
299         {
300             FAPDoubleList<T> list = new FAPDoubleList<T>();
301             foreach (FAPDoubleSet<T> set in fapList)
302             {
303                 list.fapList.AddLast(set.Copy());
304             }
```

```
305
306             return list;
307         }
308
309         public override string ToString()
310         {
311             StringBuilder sb = new StringBuilder();
312             sb.AppendLine("Hashcode: " + GetHashCode());
313             foreach (FAPDoubleSet<T> set in this.fapList)
314             {
315                 sb.Append(set).Append("\r\n");
316             }
317             return sb.ToString();
318         }
319
320         public string ShowFullCellAdresses(int cRef, int rRef)
321         {
322             StringBuilder sb = new StringBuilder();
323             foreach (FAPDoubleSet<T> set in fapList)
324             {
325                 sb.Append(set.ShowFullCellAdresses(cRef, rRef)).Append("\r\n");
326             }
327             return sb.ToString();
328         }
329
330         public int CountFapDoubleSets()
331         {
332             int counter =  ;
333             foreach (FAPDoubleSet<T> set in fapList)
334             {
335                 counter++;
336             }
337
338             return counter;
339         }
340
341         public int CountFapSets()
342         {
343             int counter =  ;
344
345             foreach (FAPDoubleSet<T> set in fapList)
346             {
347                 counter += set.Column.Count + set.Row.Count;
348             }
349
350             return counter;
351         }
352
353         public C5.ArrayList<FullCellAddr<T>> GetAdressesAbsolute()
354         {
355             return GetAdresses( ,  );
```

```
356           }
357
358       }
359
360 }
361
```

FAP

FAPList

```
1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using System.Text;
5 using C5;
6 using Logger;
7 using SpecialMath;
8
9 namespace FAP
10 {
11     /// <summary>
12     /// Describes unions of sequences
13     /// </summary>
14     public class FAPList:IEquatable<FAPList>, System.Collections.Generic.IEnumerable<FAPSet>
15     {
16         private TreeSet<FAPSet> listSet;
17
18         public int Count
19         {
20             get { return listSet.Count; }
21         }
22
23         private bool absolute;
24
25         public bool Absolute
26         {
27             get { return absolute; }
28             set { absolute = value; }
29         }
30
31         public FAPList()
32         {
33             listSet = new TreeSet<FAPSet>();
34         }
35
36         public FAPList(FAPSet set, bool absolute)
37         {
38             listSet = new TreeSet<FAPSet>();
39             listSet.Add(set);
40             this.absolute = absolute;
41         }
42
43         public FAPList(int a, int b, int k, bool absolute)
44             : this(new FAPSet(a, b, k), absolute)
45         {
46
47         }
48
49         private FAPList(TreeSet<FAPSet> list, bool absolute)
50         {
51             this.listSet = list;
```

```
52              this.absolute = absolute;
53          }
54
55          private FAPList(IEnumerable<FAPSet> list, bool absolute)
56          {
57              listSet = new TreeSet<FAPSet>();
58              listSet.AddAll(list);
59              this.absolute = absolute;
60          }
61
62          private static TreeSet<FAPSet> RangeFromTo(TreeSet<FAPSet> list, FAPSet start, FAPSet end)
63          {
64              IDirectedCollectionValue<FAPSet> range = list.RangeFromTo(start, end);
65              TreeSet<FAPSet> back = range as TreeSet<FAPSet>;
66              if (back == null)
67              {
68                  back = new TreeSet<FAPSet>();
69                  back.AddSorted(range);
70              }
71              return back;
72          }
73
74          private static TreeSet<FAPSet> FindAll(TreeSet<FAPSet> list, Fun<FAPSet, bool> fun)
75          {
76              IIndexedSorted<FAPSet> filtered = list.FindAll(fun);
77              TreeSet<FAPSet> back = filtered as TreeSet<FAPSet>;
78              if (back == null)
79              {
80                  back = new TreeSet<FAPSet>();
81                  back.AddSorted(filtered);
82              }
83              return back;
84          }
85
86          /// <summary>
87          /// Add to the list, checking specific things such as k value eq to one, and how the predesessor and successor relates
88          /// </summary>
89          /// <param name="set"></param>
90          internal void AddWithCheck(FAPSet set)
91          {
92             if(set.K == 1)
93                  set = new FAPSet(set.A, 1, 1);
94
95              listSet.Add(set);
96
97              if(listSet.Count < 2)
98                  return;
99
100             CheckSuccessorAndPredesessor(set, false);
101         }
102
```

```
103         /// <summary>
104         /// Checks if predesessor or successor can be merged with this set. Based on that this set does not
105         /// overlap with any other set
106         /// </summary>
107         /// <param name="set"></param>
108         private bool CheckSuccessorAndPredesessor(FAPSet set, bool checkForOverLapForSetWithSingleKValue)
109         {
110             bool isAltered = false;
111             int number = listSet.IndexOf(set);
112             int distanceToSuccessor = int.MinValue;
113             int distanceToPredesessor = int.MinValue;
114             FAPSet predesessor = FAPSet.EMPTY;
115             FAPSet successor = FAPSet.EMPTY;
116 
117             if(number != 0)
118             {
119                 predesessor = listSet.Predecessor(set);
120                 distanceToPredesessor = GetDistanceBetweenSets(predesessor, set);
121                 if(distanceToPredesessor <=0 && distanceToPredesessor != int.MinValue && checkForOverLapForSetWithSingleKValue)
122                 {
123                     if((set.A-predesessor.A)%predesessor.B == 0)
124                     {
125                         listSet.Remove(set);
126                     }
127                     else
128                     {
129                         listSet.Remove(predesessor);
130                         //this value will never be a int (check the if sentence)
131                         double dblsplitValue = (set.A - predesessor.A)/(double) predesessor.B;
132                         //we want the value just before this (will be less than K since distanceToPredesessor is less than 0)
133                         int splitValue = (int) dblsplitValue;
134 
135                         FAPSet before = new FAPSet(predesessor.A, predesessor.B, splitValue+1);
136                         AddWithCheck(before);
137 
138                         int afterAValue = predesessor.A + predesessor.B*(splitValue + 1);
139                         FAPSet after = new FAPSet(afterAValue, predesessor.B, predesessor.K-(splitValue+1));
140                         AddWithCheck(after);
141                     }
142                     return true;
143                 }
144             }
145 
146             if(listSet.Count > number+1)
147             {
148                 successor = listSet.Successor(set);
149                 distanceToSuccessor = GetDistanceBetweenSets(set, successor);
150                 if(distanceToSuccessor == 0 && checkForOverLapForSetWithSingleKValue)
151                 {
152                     listSet.Remove(set);
153                     return true;
```

```
154                 }
155             }
156
157             bool setBVEqToPredesessor = (set.K == 1 || set.B == distanceToPredesessor);
158             bool setBVEqToSuccessor = (set.K == 1 || set.B == distanceToSuccessor);
159             bool predesessorBEqToDistance = (predesessor.K == 1 || predesessor.B == distanceToPredesessor);
160             bool successorBEqToDistance = (successor.K == 1 || successor.B == distanceToSuccessor);
161             bool usePredesossor = distanceToPredesessor != int.MinValue;
162             bool useSuccessor = distanceToSuccessor != int.MinValue;
163
164
165             bool removeOneFromSuccessor = false;
166             bool removeOneFromPredesessor = false;
167             bool removeFirstFromSet = false;
168             bool removeLastFromSet = false;
169             int newKValue = 0;
170             int newAValue = 0;
171             int newBValue = distanceToPredesessor;
172             int new2ndKvalue = 0;
173             int new2ndAValue = 0;
174             int new2ndBValue = distanceToSuccessor;
175
176
177
178             if (usePredesossor)
179             {
180                 if (predesessorBEqToDistance)
181                 {
182                     newKValue += predesessor.K;
183                     newAValue = predesessor.A;
184                 }
185                 else
186                 {
187                     newKValue += 1;
188                     removeOneFromPredesessor = true;
189                     newAValue = predesessor.MaxValue();
190                 }
191             }
192
193             if (useSuccessor)
194             {
195                 if (successorBEqToDistance)
196                 {
197                     new2ndKvalue += successor.K;
198                 }
199                 else
200                 {
201                     new2ndKvalue += 1;
202                     removeOneFromSuccessor = true;
203                 }
204             }
```

```
205
206             if (usePredesossor || useSuccessor)
207             {
208                 if (setBVEqToPredesessor)
209                 {
210                     //check if they can all be put together
211                     if (distanceToPredesessor == distanceToSuccessor)
212                     {
213                         newKValue += set.K;
214                         newKValue += new2ndKvalue;
215                         new2ndKvalue = 0;
216                     }
217                     else if (set.K == 1)
218                     {
219                         if (!useSuccessor || (distanceToPredesessor < distanceToSuccessor && usePredesossor))//predesessor closest
220                         {
221                             new2ndKvalue = 0;
222                             newKValue += 1;
223                         }
224                         else //successor closest
225                         {
226                             newKValue = 0;
227                             new2ndKvalue += 1;
228                             new2ndAValue = set.A;
229                         }
230                     }
231                     else if (useSuccessor)
232                     {
233                         newKValue += set.K - 1;
234                         new2ndKvalue += 1;
235                         new2ndAValue = set.MaxValue();
236                         removeLastFromSet = true;
237                     }
238                     else
239                         newKValue += set.K;
240                 }
241                 else if (setBVEqToSuccessor)
242                 {
243                     if (usePredesossor)
244                     {
245                         newKValue += 1;
246                         new2ndKvalue += set.K - 1;
247                         new2ndAValue = set.A+set.B;
248                         removeFirstFromSet = true;
249                     }
250                     else
251                     {
252                         new2ndKvalue += set.K;
253                         new2ndAValue = set.A;
254                     }
255                 }
```

```
256                 else //note that set.K != 1 else setBVEqToPredesessor would be true
257                 {
258                     if (usePredesossor)
259                     {
260                         newKValue += 1;
261                         removeFirstFromSet = true;
262                     }
263                     if (useSuccessor)
264                     {
265                         new2ndKvalue += 1;
266                         new2ndAValue = set.MaxValue();
267                         removeLastFromSet = true;
268                     }
269                 }
270 
271                 if (newKValue != 0 && new2ndKvalue != 0 || distanceToPredesessor == distanceToSuccessor)
272                 {
273                     listSet.RemoveInterval(number - 1, 3);
274                 }
275                 else if (newKValue != 0)
276                 {
277                     listSet.RemoveInterval(number - 1, 2);
278                 }
279                 else if (new2ndKvalue != 0)
280                 {
281                     listSet.RemoveInterval(number, 2);
282                 }
283 
284                 TreeSet<FAPSet> newAdd = new TreeSet<FAPSet>();
285 
286                 if (newKValue != 0)
287                     newAdd.Add(new FAPSet(newAValue, newBValue, newKValue));
288 
289                 if (new2ndKvalue != 0)
290                     newAdd.Add(new FAPSet(new2ndAValue, new2ndBValue, new2ndKvalue));
291 
292                 if (removeOneFromPredesessor)
293                 {
294                     if (predesessor.K > 1)
295                         newAdd.Add(new FAPSet(predesessor.A, predesessor.B, predesessor.K - 1));
296                 }
297 
298                 if (removeOneFromSuccessor)
299                 {
300                     if (successor.K > 1)
301                         newAdd.Add(new FAPSet(successor.A + successor.B, successor.B, successor.K - 1));
302                 }
303 
304                 if (removeFirstFromSet && removeLastFromSet)
305                 {
306                     if (set.K > 2)
```

```
307                         newAdd.Add(new FAPSet(set.A + set.B, set.B, set.K - 2));
308
309                 }
310                 else if (removeFirstFromSet)
311                     newAdd.Add(new FAPSet(set.A + set.B, set.B, set.K - 1));
312                 else if (removeLastFromSet)
313                     newAdd.Add(new FAPSet(set.A, set.B, set.K - 1));
314
315                 listSet.AddSorted(newAdd);
316                 isAltered = true;
317             }
318             else
319                 listSet.Add(set);
320
321             return isAltered;
322
323         }
324
325         /// <summary>
326         ///
327         /// </summary>
328         /// <param name="first"></param>
329         /// <param name="last"></param>
330         /// <returns>-1 if it not suitable</returns>
331         private static int GetDistanceBetweenSets(FAPSet first, FAPSet last)
332         {
333             int distanceBetweenSets = last.A - first.MaxValue();
334             bool okWithFirst = first.K == 1 || first.B >= distanceBetweenSets;
335             bool okWithLast = last.K == 1 || last.B >= distanceBetweenSets;
336
337             if (okWithFirst && okWithLast)
338                 return distanceBetweenSets;
339             else
340                 return int.MinValue;
341         }
342
343
344         public void AddToList(int a, int b, int k)
345         {
346             if(k == 1)
347                 b = 1;
348             listSet.Add(new FAPSet(a, b, k));
349         }
350
351         public void Union(FAPList list)
352         {
353             //TODO implement this much more effectively (if needed)
354             foreach (FAPSet set in list.listSet)
355             {
356                 Union(set);
357             }
```

```
358         }
359
360         public void Union(int a, int b, int k)
361         {
362             Union(new FAPSet(a,b,k));
363         }
364
365         internal void Union(FAPSet add)
366         {
367             if(listSet.Find(ref add))
368                 return;
369
370             if(add.K == 0 || add.B == 0)
371                 return;
372
373             Overlap overlap = Overlaps(add, true);
374             if (overlap == Overlap.yes)
375             {
376                 ReallyPrimitiveAddOverlappingSet(add);
377             }
378             else if(overlap == Overlap.no)
379             {
380                 AddWithCheck(add);
381             }
382         }
383
384         private void ReallyPrimitiveAddOverlappingSet(FAPSet add)
385         {
386             ArrayList<int> points = new ArrayList<int>();
387             add.GetValues(ref points);
388             foreach (int i in points)
389             {
390                 AddSingleWithCheck(new FAPSet(i, 1, 1));
391             }
392         }
393
394         private void AddSingleWithCheck(FAPSet add)
395         {
396             listSet.Add(add);
397             CheckSuccessorAndPredesessor(add, true);
398         }
399
400         public void AddWhenBuildingSupportGraph(int i)
401         {
402             int max = listSet.Count-1;
403             if(max >= 0)
404             {
405                 FAPSet last = listSet[max];
406                 int distance = i - last.MaxValue();
407                 if(distance == last.B || last.K == 1)
408                 {
```

```
409                     listSet.Remove(last);
410                     listSet.Add(new FAPSet(last.A, distance, last.K + 1));
411                     return;
412                 }
413             }
414 
415             listSet.Add(new FAPSet(i, 1, 1));
416         }
417 
418         private enum Overlap {no, yes, fapsetShouldNotBeAdded}
419 
420         private Overlap Overlaps(FAPSet add, bool alterList)
421         {
422             bool back = false;
423             bool forward = true;
424             bool backward = true;
425             FAPSet next = add;
426             FAPSet prev = add;
427             listSet.Add(add);
428             int index = listSet.IndexOf(add);
429             int indexNext = index;//we remove it again therefore not minus one
430             int indexBack = index - 1;
431             listSet.Remove(add);
432 
433             while(forward && indexNext<listSet.Count)
434             {
435                 next = listSet[indexNext];
436                 if (next.Overlap(add))
437                 {
438                     FAPSet newSet;
439                     if (alterList && indexNext == index + 1 && add.BelongsToOrContinuation(next, out newSet))
440                     {
441                         listSet.Remove(next);
442                         listSet.Add(newSet);
443                         return Overlap.fapsetShouldNotBeAdded;
444                     }
445                     //back = true;
446                     return Overlap.yes;
447                 }
448                 else
449                     forward = false;
450             }
451 
452             while(backward && indexBack >= 0)
453             {
454                 prev = listSet[indexBack];
455                 if(prev.Overlap(add))
456                 {
457                     FAPSet newSet;
458                     if (alterList && indexBack == index - 1 && prev.BelongsToOrContinuation(add, out newSet))
459                     {
```

```
460                         listSet.Remove(prev);
461                         listSet.Add(newSet);
462                         return Overlap.fapsetShouldNotBeAdded;
463                     }
464                     //back = true;
465                     return Overlap.yes;
466                 }
467                 else
468                     backward = false;
469             }
470 
471             return back ? Overlap.yes : Overlap.no;
472         }
473 
474         #region ToString Methods
475         public override string ToString()
476         {
477             StringBuilder builder = new StringBuilder();
478             builder.AppendLine("Is absolute: " + Absolute);
479             foreach (FAPSet set in listSet)
480             {
481                 builder.AppendLine(set.ToString());
482             }
483             return builder.ToString();
484 
485         }
486 
487         public string ToString(bool sorted)
488         {
489             if (!sorted)
490                 return ToString();
491             else
492             {
493                 SortedArray<FAPSet> list = new SortedArray<FAPSet>();
494                 list.AddAll(listSet);
495 
496                 StringBuilder builder = new StringBuilder();
497                 foreach (FAPSet set in list)
498                 {
499                     builder.AppendLine(set.ToString());
500                 }
501                 return builder.ToString();
502             }
503         }
504         #endregion
505 
506         #region IEquatable<FAPList> Members
507 
508 
509         public bool Equals(FAPList other)
510         {
```

```
511             //TODO: think about if they both have to be absolute or both relative
512             if (listSet.Count == other.listSet.Count && Absolute == other.Absolute)
513             {
514                 bool stillTrue = true;
515                 for (int i = 0; i < listSet.Count && stillTrue; i++)
516                 {
517                     stillTrue = listSet[i] == other.listSet[i];
518                 }
519                 return stillTrue;
520             }
521             else
522                 return false;
523         }
524 
525         #endregion
526 
527         internal C GetValues<C>(int relative)where C:C5.ICollection<int>, new()
528         {
529             C back = new C();
530             if (absolute)
531             {
532                 foreach (FAPSet set in listSet)
533                 {
534                     set.GetValues(ref back);
535                 }
536             }
537             else
538             {
539                 foreach (FAPSet set in listSet)
540                 {
541                     set.GetValues(relative, ref back);
542                 }
543             }
544 
545             return back;
546         }
547 
548 
549 
550         internal int GetEdgeCount()
551         {
552             int count = 0;
553             foreach (FAPSet set in listSet)
554             {
555                 count += set.GetEdgeCount();
556             }
557 
558             return count;
559         }
560 
561         public IEnumerator<FAPSet> GetEnumerator()
```

```
562         {
563             return listSet.GetEnumerator();
564         }
565 
566         IEnumerator IEnumerable.GetEnumerator()
567         {
568             return GetEnumerator();
569         }
570 
571         internal void AddLines(int lineNumber, int numberOfLines, int relative)
572         {
573             //TODO we might not need to union all fap sets in add, maybe those with a>linenumber
574             //does only need to be added directly to the listset
575 
576             ArrayList add = new ArrayList();
577             ArrayList remove = new ArrayList();
578             if (!Absolute)
579                 lineNumber -= relative;
580             foreach (FAPSet set in listSet)
581             {
582                 AddLinesToSet(set, lineNumber, numberOfLines, add, remove);
583             }
584             foreach (FAPSet set in remove)
585                 Remove(set);
586 
587             foreach (FAPSet set in add)
588                 Union(set);
589 
590             if (listSet.Count == 0)
591             {
592                 Log.Fatal(this, "There was zero elements left after addlines");
593             }
594         }
595 
596         internal void DeleteLines(int startLineNumber, int numberOfLines)
597         {
598             ArrayList add = new ArrayList();
599             ArrayList remove = new ArrayList();
600             foreach (FAPSet set in listSet)
601             {
602                 DeleteLinesFromSet(set, startLineNumber, numberOfLines, add, remove);
603             }
604             foreach (FAPSet set in remove)
605                 Remove(set);
606 
607             foreach (FAPSet set in add)
608                 Union(set);
609         }
610 
611 
612 
```

```
613
614
615         private void AddLinesToSet(FAPSet set, int lineNumber, int numberOfLines, ArrayList listToAdd, ArrayList listToRemove)
616         {
617             if (set.A + set.B * (set.K - 1) >= lineNumber)
618             {
619                 listToRemove.Add(set);
620
621                 if (set.A >= lineNumber)
622                 {
623                     set.A += numberOfLines;
624                     listToAdd.Add(set);
625                 }
626                 else
627                 {
628
629                     int k_new = (lineNumber - set.A + set.B - 1) / set.B;
630                     listToAdd.Add(new FAPSet(set.A + k_new * set.B + numberOfLines, set.B, set.K - k_new));
631
632                     set.K = k_new;
633                     listToAdd.Add(set);
634                 }
635             }
636         }
637
638         private void Remove(FAPSet set)
639         {
640             listSet.Remove(set);
641         }
642
643         //TODO this probably doesn't work - need same mods as add
644         private void DeleteLinesFromSet(FAPSet set, int startLineNumber, int numberOfLines, ArrayList listToAdd, ArrayList
    listToRemove)
645         {
646             if (set.A + set.B * (set.K - 1) >= startLineNumber)
647             {
648                 listToRemove.Add(set);
649                 if (set.A >= startLineNumber + numberOfLines)
650                 {
651                     set.A -= numberOfLines;
652                     listToAdd.Add(set);
653                 }
654                 else
655                 {
656                     int k1 = (startLineNumber - set.A + set.B - 1) / set.B;
657                     int k2 = (startLineNumber + numberOfLines - set.A + set.B - 1) / set.B;
658
659                     if(k2<k1)
660                         listToAdd.Add(new FAPSet(set.A + set.B * k2 - numberOfLines, set.B, set.K - k2));
661
662                     if (1 <= k1)
```

```
663                     {
664                         set.K = k1;
665                         listToAdd.Add(set);
666                     }
667                 }
668             }
669         }
670 
671         public FAPList Copy()
672         {
673             return new FAPList((TreeSet<FAPSet>)this.listSet.Clone(), this.absolute);
674         }
675 
676         public bool Contains(int r, int relR)
677         {
678             if (!Absolute)
679                 r -= relR;
680             foreach (FAPSet set in listSet)
681             {
682                 if(set.Contains(r))
683                     return true;
684             }
685             return false;
686         }
687 
688         public bool ContainsArea(int start, int end, int relative)
689         {
690             if(!Absolute)
691             {
692                 start -= relative;
693                 end -= relative;
694             }
695             foreach (FAPSet set in listSet)
696             {
697                 if(set.Contains(start, end))
698                     return true;
699             }
700             return false;
701         }
702 
703         internal bool OverlapsFAPSet(FAPSet containSet)
704         {
705             foreach (FAPSet set in listSet)
706             {
707                 if (set.Overlap(containSet))
708                     return true;
709             }
710             return false;
711         }
712 
713         public void RemoveFAPSet(FAPSet remove)
```

```
714         {
715             ArrayList<FAPSet> alterSets = new ArrayList<FAPSet>();
716 
717             if(listSet.Count == 0)
718                 return;
719 
720             FAPSet first = listSet[0];
721             FAPSet last = listSet[listSet.Count - 1];
722 
723             if(remove.CompareTo(first) > 0)
724             {
725                 FAPSet predecessor = listSet.Predecessor(remove);
726                 if(predecessor.Overlap(remove))
727                 {
728                     alterSets.Add(predecessor);
729                     listSet.Remove(predecessor);
730                 }
731             }
732 
733 
734             if(remove.CompareTo(last) < 0)
735             {
736                 FAPSet current = remove;
737 
738                 while(current.CompareTo(last) < 0 && (current = listSet.Successor(current)).Overlap(remove))
739                 {
740                     alterSets.Add(current);
741                     listSet.Remove(current);
742                 }
743             }
744 
745             foreach (FAPSet set in alterSets)
746             {
747                 set.RemoveFAPSet(remove, this);
748             }
749         }
750 
751         internal FAPList RemoveArea(int start, int end, int relative)
752         {
753             FAPList emptyRef;
754             return RemoveArea(start, end, relative, out emptyRef);
755         }
756 
757         /// <summary>
758         /// Removes a area
759         /// </summary>
760         /// <param name="start"></param>
761         /// <param name="end"></param>
762         /// <returns>Returns splitted part</returns>
763         internal FAPList RemoveArea(int start, int end, int relative, out FAPList Removed)
764         {
```

```
765             if(!Absolute)
766             {
767                 start -= relative;
768                 end -= relative;
769             }
770             //TODO: we can find possible sets instead of searching through them all
771             TreeSet<FAPSet> backRemovedLines = new TreeSet<FAPSet>();
772             TreeSet<FAPSet> newList = new TreeSet<FAPSet>();
773             foreach (FAPSet set in listSet)
774             {
775                 if (set.Contains(start, end))
776                 {
777                     FAPSet removed;
778                     FAPSet before;
779                     FAPSet after;
780                     set.RemoveArea(start, end, out before, out after, out removed);
781                     if (before != FAPSet.EMPTY)
782                         newList.Add(before);
783                     if (after != FAPSet.EMPTY)
784                         newList.Add(after);
785                     if (removed != FAPSet.EMPTY)
786                         backRemovedLines.Add(removed);
787                 }
788                 else
789                     newList.Add(set);
790             }
791 
792             Removed = new FAPList(backRemovedLines, this.absolute);
793 
794             FAPList listBack = new FAPList(newList, this.absolute);
795 
796             return listBack;
797         }
798 
799         public FAPList Split(int r, int rel)
800         {
801             ArrayList<FAPSet> back = new ArrayList<FAPSet>();
802             if (!Absolute)
803                 r -= rel;
804             foreach (FAPSet set in listSet)
805             {
806                 if (set.Contains(r))
807                     SplitSet(set, r, ref back);
808                 else
809                     back.Add(set);
810             }
811             if(back.Count == 0)
812             {
813                 return null;
814             }
815             FAPList listBack = new FAPList(back, Absolute);
```

```
816
817             return listBack;
818         }
819
820         private static void SplitSet(FAPSet set, int r, ref ArrayList<FAPSet> list)
821         {
822             if(r==set.A)
823             {
824                 if(set.K > 1)
825                 {
826                     set.A += set.B;
827                     set.K--;
828                     list.Add(set);
829                 }
830             }
831             else if(r == set.A+set.B*(set.K-1))
832             {
833                 set.K--;
834                 list.Add(set);
835             }
836             else
837             {
838                 FAPSet before = set;
839                 int i = (r - set.A)/set.B;
840                 before.K = i;
841                 list.Add(before);
842
843                 set.A += set.B*(i + 1);
844                 set.K -= (i+1);
845                 list.Add(set);
846             }
847         }
848
849         public int CompareTo(FAPList other)
850         {
851             int number = this.listSet.Count.CompareTo(other.listSet.Count);
852
853             if (number != 0)
854                 return number;
855
856             number = this.absolute.CompareTo(other.absolute);
857
858             if(number != 0)
859                 return number;
860
861
862             for (int i = 0; i < listSet.Count; i++)
863             {
864                 int result = listSet[i].CompareTo(other.listSet[i]);
865                 if(result != 0)
866                     return result;
```

```
867             }
868
869             return 0;
870         }
871
872         internal int GetAlternativeHashCode()
873         {
874             int returnHashCode = 0;
875             foreach (FAPSet set in listSet)
876             {
877                 returnHashCode = 29*returnHashCode + set.GetHashCode();
878             }
879             returnHashCode = returnHashCode*29 + (absolute ? 0 : 1);
880             return returnHashCode;
881         }
882     }
883
884
885 }
886
```

Utilities

Tools

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Utilities
{
    public static class Tools
    {
        public static void SwitchValues<T>(ref T val1, ref T val2)
        {
            T temp = val1;
            val1 = val2;
            val2 = temp;
        }
    }
}

```

Special math

MathFunc

```
1  using System;
2  using C5;
3
4  namespace SpecialMath
5  {
6      public class MathFunc
7      {
8          public delegate int getIntFromObject<T>(T input);
9
10         //TODO check if some of the functions here can be optimized, especially if lists should be
11         //sorted to avoid duplicates
12
13         public static int GreatestCommonDivisor(int m, int n)    //  Greatest common divisor
14         {
15             int rest = m % n;
16             while (rest != 0)
17             {
18                 m = n;
19                 n = rest;
20                 rest = m % n;
21             }
22             return n;
23
24             //TODO: PROFILE
25             //m % n == 0 ? n : gcd(n, m%n);
26         }
27
28         public static int LowestCommonMultiple(int m, int n)    //  Least common multiple
29         {
30             return m * (n / GreatestCommonDivisor(m, n));
31         }
32
33         public static int LowestCommonMultiple(int[] list)
34         {
35             //TODO check if this is fast enough, or if it has to be optimized
36             int last = 1;
37             foreach (int i in list)
38             {
39                 last = LowestCommonMultiple(last, i);
40             }
41             return last;
42         }
43
44         public static int LowestCommonMultiple<T>(getIntFromObject<T> method, T[] list)
45         {
46             int last = 1;
47             foreach (T item in list)
48             {
49                 last = LowestCommonMultiple(last, method(item));
50             }
51             return last;
```

```
        }

        /// <summary>
        /// Eliminates duplicates before finding lowest common multiple
        /// </summary>
        /// <param name="list"></param>
        /// <returns></returns>
        public static int LowestCommonMultipleWithDuplicates(int[] list)
        {
            HashSet<int> set = new HashSet<int>();
            set.AddAll(list);
            return LowestCommonMultiple(set.ToArray());
        }

        public static void Main()
        {
            Console.WriteLine(LowestCommonMultiple(1, 15));
            Console.ReadLine();
        }
    }
}

```

CoreCalc.AlternativeIO

Excel2003XMLReader

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using CarlosAg.ExcelXmlWriter;
using C5;

namespace CoreCalc.AlternativeIO
{
    class Excel2003XMLReader
    {
        public static Workbook ReadFile(string filename)
        {
            Workbook backWorkbook = new Workbook();
            CarlosAg.ExcelXmlWriter.Workbook workbook = new CarlosAg.ExcelXmlWriter.Workbook();
            if (!System.IO.File.Exists(filename))
                throw new FileNotFoundException("The file was not found: " + filename);

            workbook.Load(filename);

            foreach (CarlosAg.ExcelXmlWriter.Worksheet worksheet in workbook.Worksheets)
            {
                if (worksheet.Table.ExpandedColumnCount > 0 && worksheet.Table.ExpandedRowCount > 0)
                    new Sheet(backWorkbook, worksheet.Name, worksheet.Table.ExpandedColumnCount, worksheet.Table.ExpandedRowCount);
            }
            HashDictionary string, Expr> expressionIndex = new HashDictionary string, Expr>();

            foreach (CarlosAg.ExcelXmlWriter.Worksheet worksheet in workbook.Worksheets)
            {


                if (worksheet.Table.ExpandedColumnCount > 0 && worksheet.Table.ExpandedRowCount > 0)
                {
                    int rowIndex = -1;
                    Sheet newSheet = backWorkbook[worksheet.Name];
                    for (int r = 0; r   worksheet.Table.Rows.Count; r++)
                    {
                        int columnNumber = -1;
                        WorksheetRow row = worksheet.Table.Rows[r];
                        rowIndex = row.Index == 0 ? rowIndex+1 : row.Index-1;
                        for (int c = 0; c   row.Cells.Count; c++)
                        {
                            Cell newCell = null;
                            if (row.Cells[c].Formula != null)
                            {
                                Expr known;
                                string formula = row.Cells[c].Formula;
                                if (expressionIndex.Find(ref formula, out known))
                                {
                                    Formula form = new Formula(backWorkbook, known);
                                    newCell = form;
```

```
52                                      if (row.Cells[c].Data.Text != null)
53                                          form.SetValue(newSheet, c, r, Cell.ParseValue(row.Cells[c].Data.Text, row.Cells[c].Data.
      Type.ToString()));
54                                  }
55                                  else
56                                  {
57                                      newCell = Cell.Parse(row.Cells[c].Formula, backWorkbook, c, r);
58                                      Formula form = newCell as Formula;
59                                      if (form != null)
60                                      {
61                                          expressionIndex.Add(row.Cells[c].Formula, form.Expr);
62                                          if (row.Cells[c].Data.Text != null)
63                                              form.SetValue(newSheet, c, r, Cell.ParseValue(row.Cells[c].Data.Text, row.Cells[c].Data
      .Type.ToString()));
64                                      }
65                                  }
66
67                              }
68                              else
69                              {
70                                  if (row.Cells[c].Data.Text != null)
71                                  {
72                                      if (row.Cells[c].Data.Type == DataType.String)
73                                          newCell = Cell.Parse("'" + row.Cells[c].Data.Text, backWorkbook, c, r);
74                                      else if (row.Cells[c].Data.Type == DataType.Error)
75                                          newCell = Cell.Parse("'" + row.Cells[c].Data.Text, backWorkbook, c, r);
76                                      else if (row.Cells[c].Data.Type == DataType.Number || row.Cells[c].Data.Type == DataType.
      Integer)
77                                      {
78                                          newCell = new NumberCell(Convert.ToDouble(row.Cells[c].Data.Text));
79                                      }
80                                      else
81                                          newCell = Cell.Parse(row.Cells[c].Data.Text, backWorkbook, c, r);
82
83
84                                  }
85                              }
86                              columnNumber = row.Cells[c].Index == 0 ? columnNumber+1 : row.Cells[c].Index-1;
87
88                              newSheet[columnNumber, rowIndex] = newCell;
89                          }
90                      }
91                  }
92              }
93
94              return backWorkbook;
95          }
96      }
97  }
98
```

CoreCalc.SupportGraph

AddressCollector

```
using System;
using System.Text;
using C5;

namespace CoreCalc.SupportGraf
{
    class AddressCollector:IAdressCollector<CellArea>
    {
        public static T GetAllAdresses<T>(Expr expr)where T:C5.ICollection<CellArea>, new()
        {
            T back = new T();
            expr.GetAdresses(new AddressCollector(), ref back);

            return back;
        }

        public static void GetAllAdresses<T>(Expr expr, ref T back) where T : C5.ICollection<CellArea>, new()
        {
            expr.GetAdresses(new AddressCollector(), ref back);
        }



        #region IAdressCollector Members

        public void GetAdresses<T>(CellArea cellArea, ref T adresses)where T:ICollection<CellArea>
        {
            adresses.Add(cellArea);
        }

        public void GetAdresses<T>(CellRef cellRef, ref T adresses)where T:ICollection<CellArea>
        {
            adresses.Add(new CellArea(cellRef.sheet, cellRef.raref, cellRef.raref));
        }

        public void GetAdresses<T>(FunCall funCall, ref T adresses)where T:ICollection<CellArea>
        {
            foreach (Expr e in funCall.es)
            {
                e.GetAdresses(this, ref adresses);
            }
        }

        public void GetAdresses<T>(Expr expr, ref T adresses)where T:ICollection<CellArea>
        {}



        #endregion
    }

```

```
52
53      internal class CustemizedAdressCollector<Y>: IAdressCollector<Y>
54      {
55          internal delegate void AddressFunction(CellArea area);
56
57          private AddressFunction function;
58
59          private CustemizedAdressCollector(AddressFunction function)
60          {
61              this.function = function;
62          }
63
64          public static void GetAdresses(AddressFunction function, Expr expr)
65          {
66              ArrayList<Y> emptyList = new ArrayList<Y>();
67              expr.GetAdresses(new CustemizedAdressCollector<Y>(function), ref emptyList);
68          }
69
70          public void GetAdresses<T>(CellArea cellArea, ref T adresses) where T : ICollection<Y>
71          {
72              function(cellArea);
73          }
74
75          public void GetAdresses<T>(CellRef cellRef, ref T adresses) where T : ICollection<Y>
76          {
77              function(new CellArea(cellRef.sheet, cellRef.raref, cellRef.raref));
78          }
79
80          public void GetAdresses<T>(Expr expr, ref T adresses) where T : ICollection<Y>
81          {}
82
83          public void GetAdresses<T>(FunCall funCall, ref T adresses) where T : ICollection<Y>
84          {
85              foreach (Expr e in funCall.es)
86              {
87                  e.GetAdresses(this, ref adresses);
88              }
89          }
90      }
91
92      interface IAdressCollector<Y>
93      {
94          void GetAdresses<T>(CellArea cellArea, ref T adresses) where T : ICollection<Y>;
95
96          void GetAdresses<T>(CellRef cellRef, ref T adresses) where T : ICollection<Y>;
97
98          void GetAdresses<T>(Expr expr, ref T adresses) where T : ICollection<Y>;
99
100         void GetAdresses<T>(FunCall funCall, ref T adresses) where T : ICollection<Y>;
101
102     }
```

```
103 }
104
```

CoreCalc.SupportGraph

SupportGraph

```
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using C5;
using FAP;
using Utilities;

namespace CoreCalc.SupportGraf
{
    public enum TopologicalSortType { Standard, NoloopsAllowed, NoRecursion, UseBoolArray, NoLinkedListStandardBoolArray};

    internal class SupportGraph : IEnumerable
    {

        private Sheet owner;
        private FAPDoubleList<Sheet>[,] graph;

        internal int Cols
        {
            get { return graph.GetLength(0); }
        }

        internal int Rows
        {
            get { return graph.GetLength(1); }
        }

        internal FAPDoubleList<Sheet> this[int c,int r]
        {
            get { return graph[c,r]; }
            set { graph[c,r] = value; }
        }

        internal FAPDoubleList<Sheet> this[CellAddr ca]
        {
            get { return this[ca.col, ca.row]; }
            set { this[ca.col, ca.row] = value; }
        }

        internal FAPDoubleList<Sheet>[,] Graph
        {
            get { return graph; }
            set { graph = value; }
        }

        internal SupportGraph(int c, int r, Sheet owner)
        {
            graph = new FAPDoubleList<Sheet>[c,r];
            this.owner = owner;
        }
```

```
52
53          public C5.IList<FullCellAddr<Sheet>> GetSupportGraph(CellAddr[] alteredCells, FullCellAddr<Sheet>[] altered,
        TopologicalSortType TopologicalSorting)
54          {
55              ArrayList<FullCellAddr<Sheet>> list = new ArrayList<FullCellAddr<Sheet>>();
56              list.AddAll(altered);
57              foreach (CellAddr alteredCell in alteredCells)
58              {
59                  list.Add(new FullCellAddr<Sheet>(alteredCell.col, alteredCell.row, owner));
60              }
61
62              ArrayList<FullCellAddr<Sheet>> emptyList = new ArrayList<FullCellAddr<Sheet>>();
63              TopologicalSort.GetChildren<FullCellAddr<Sheet>, ArrayList<FullCellAddr<Sheet>>> getSupportNodeChildrens = delegate
        (FullCellAddr<Sheet> node)
64                                                                  {
65                                                                      if (node.sheet.SupportGraph[node.col, node.row] == null)
66                                                                          return emptyList;
67                                                                      else
68                                                                          return node.sheet.SupportGraph[node.col, node.row].
        GetAdresses(node.col, node.row);
69                                                                  };
70              if (TopologicalSorting == TopologicalSortType.NoloopsAllowed)
71                  return TopologicalSort.ToposortAdv<FullCellAddr<Sheet>, ArrayList<FullCellAddr<Sheet>>>(getSupportNodeChildrens,
        list.ToArray());
72              else if (TopologicalSorting == TopologicalSortType.Standard)
73                  return TopologicalSort.Toposort(getSupportNodeChildrens, list.ToArray());
74              else if (TopologicalSorting == TopologicalSortType.NoRecursion)
75                  return TopologicalSort.ToposortNoRecursive(getSupportNodeChildrens, list.ToArray());
76              else if (TopologicalSorting == TopologicalSortType.UseBoolArray)
77              {
78                  MakeBoolArrays();
79                  return TopologicalSort.ToposortAdvWithBoolArray(list.ToArray());
80              }
81              else if(TopologicalSorting == TopologicalSortType.NoLinkedListStandardBoolArray)
82              {
83                  MakeBoolArrays();
84                  return TopologicalSort.ToposortAdvWithBoolArrayAndExternAdd(list.ToArray());
85
86              }
87              else throw new NotImplementedException("This topologialsort is not enabled");
88
89          }
90
91          private void MakeBoolArrays()
92          {
93              foreach (Sheet sheet in owner.workbook.sheets)
94              {
95                  sheet.InTopologicalSort = new bool[sheet.Cols, sheet.Rows];
96              }
97          }
98
```

```
 99
100         internal void DeleteAreaFromGraph(int startColumn, int endColumn, int startRow, int endRow)
101         {
102             //A bit of explanation:
103             //The HashedArrayList is used to keep absolute adresses (we don't want dublicates)
104             HashedArrayList<FullCellAddr<Sheet>> cellAdresses = new HashedArrayList<FullCellAddr<Sheet>>();
105             //We go through all of the cells to be deleted
106             if(endRow<startRow)
107                 Tools.SwitchValues(ref startRow, ref endRow);
108             if(endColumn<startColumn)
109                 Tools.SwitchValues(ref startColumn, ref endColumn);
110 
111             for (int r = startRow; r <= endRow; r++)
112             {
113                 for (int c = startColumn; c <= endColumn; c++)
114                 {
115                     //For each cell check and get formula if exists
116                     Formula formula = owner[c, r] as Formula;
117                     if (formula != null)
118                     {
119                         //Create a delegate that from a given area reference and originating cell can tell absolute
120                         //references from that cell (same cell as in last comment) and put it into our HashedArrayList
121                         CustemizedAdressCollector<FullCellAddr<Sheet>>.AddressFunction fun = delegate(CellArea cellArea)
122                             {
123                                 Area area = new Area(cellArea, owner, new CellAddr(c, r));
124                                 //Make a reference to each cell in the area
125                                 for (int col = area.colStart; col <= area.colEnd; col++)
126                                 {
127                                     for (int row = area.rowStart; row <= area.rowEnd; row++)
128                                     {
129                                         cellAdresses.Add(new FullCellAddr<Sheet>(col, row, area.sheet));
130                                     }
131                                 }
132                             };
133                         //Use the delegate to get adresses from the expressions in the formula (from the cell)
134                         CustemizedAdressCollector<FullCellAddr<Sheet>>.GetAdresses(fun, formula.Expr);
135                     }
136                 }
137             }
138             EqualityForFAPDoubleLists eq = new EqualityForFAPDoubleLists();
139             HashSet<FAPDoubleList<Sheet>> existingGrids = new HashSet<FAPDoubleList<Sheet>>(eq);
140             foreach (FullCellAddr<Sheet> addr in cellAdresses)
141             {
142                 if (addr.sheet.SupportGraph[addr.col, addr.row] != null)
143                 {
144                     FAPDoubleList<Sheet> currentItem = addr.sheet.SupportGraph[addr.col, addr.row].DeleteArea(startColumn,
    endColumn, startRow, endRow, owner, addr.col, addr.row);
145                     if (addr.col == 2 && addr.row == 9 && currentItem.GetEdgeCount() == 0)
146                         Logger.Log.Fatal(this, "Should not happen");
147                     if (currentItem != null)
148                     {
```

```
149                         existingGrids.FindOrAdd(ref currentItem);
150                     }
151                     if (addr.col == 2 && addr.row == 9 && currentItem.GetEdgeCount() == 0)
152                         Logger.Log.Fatal(this, "Should not happen");
153                     addr.sheet.SupportGraph[addr.col, addr.row] = currentItem;
154 
155                 }
156             }
157         }
158 
159         internal void DeleteCellFromGraph(Cell cell, int c, int r)
160         {
161             this.DeleteCellFromGraph(cell, new CellAddr(c, r));
162         }
163 
164 
165 
166         internal void DeleteCellFromGraph(Cell cell, CellAddr ca)
167         {
168             Formula formula = cell as Formula;
169             if(formula != null)
170             {
171                 DeleteAreaFromGraph(ca.col, ca.col, ca.row, ca.row);
172             }
173         }
174 
175 
176 
177         internal void InsertCellToGraph(Cell oldCell, Cell newCell, CellAddr ca)
178         {
179             DeleteCellFromGraph(oldCell, ca);
180 
181             InsertCellToGraph(newCell, ca);
182 
183             Logger.Log.Info(this, "Inserting " + owner.Name + "!" + CellAddr.ColumnName(ca.col) + (ca.row + 1) + " to the supportgraph");
184         }
185 
186         private void InsertCellToGraph(Cell newCell, CellAddr ca)
187         {
188             Formula formula = newCell as Formula;
189             if(formula != null)
190             {
191                 CopyToCells(formula.Expr, ca, 1, 1);
192             }
193         }
194 
195         internal void CopyToCells(Expr ex, CellAddr ca, int cols, int rows)
196         {
197             FAPSet set;
198             C5.KeyValuePair<Expr, FAPList> valuePair;
```

```
199             if (cols > rows)
200             {
201                 set = new FAPSet(ca.col, 1, cols);
202                 valuePair = new C5.KeyValuePair<Expr, FAPList>(ex, new FAPList(set, true));
203                 for (int r = ca.row; r < ca.row + rows; r++)
204                 {
205                     FAPSet rowSet = new FAPSet(r, 1, 1);
206                     SupportGraphBuilder.SupportGraphEdgeBuilder(owner, rowSet, null, valuePair, false, true);
207                 }
208             }
209             else
210             {
211                 set = new FAPSet(ca.row, 1, rows);
212                 valuePair = new C5.KeyValuePair<Expr, FAPList>(ex, new FAPList(set, true));
213                 for (int c = ca.col; c < ca.col+cols; c++)
214                 {
215                     FAPSet colSet = new FAPSet(c, 1, 1);
216                     SupportGraphBuilder.SupportGraphEdgeBuilder(owner, colSet, null, valuePair, true, true);
217                 }
218             }
219 
220             if (owner.ContainsVolatileFunction(ex))
221             {
222                 for (int c = ca.col; c < ca.col + cols; c++)
223                 {
224                     for (int r = ca.row; c < ca.row + rows; r++)
225                     {
226                         owner.workbook.VolatileExpressions.Add(new FullCellAddr<Sheet>(c, r, owner));
227                     }
228                 }
229             }
230         }
231 
232         internal void MoveCellToGraph(Cell cell, Sheet sheetFrom, CellAddr ca)
233         {
234             //InsertCellToGraph(colTo, rowTo);
235             //DeleteCellFromGraph(colFrom, rowFrom, sheetFrom);
236             //Logger.Log.Info(this, "I should move " + sheet.Name + "!" + CellAddr.ColumnName(ca.col) + (ca.row + 1) + " to the supportgraph");
237         }
238         internal void AddLinesToGraph(int R, int N, bool doRows, HashDictionary<Sheet, List<HashDictionary<Expr, FAPList>>> expressionMap, Sheet sheet)
239         {
240             //TODO: Make this more effective (reuse) and correct when making support graph for new lines,
241             //as this still needs to make precaution for relativeness
242             ExpandSupportGraph(doRows, N, R);
243 
244             MakeSupportGraphForNewLines(R, N, doRows, expressionMap, sheet);
245         }
246 
247         private void MakeSupportGraphForNewLines(int R, int N, bool doRows, HashDictionary<Sheet, List<HashDictionary<Expr,
```

```
FAPList>>> expressionMap, Sheet sheet)
    {
        SupportGraphBuilder builder = new SupportGraphBuilder();

        SupportGraph graphForNewLines = builder.GetSupportGraphForAddedLines(sheet, expressionMap, !doRows, R);

        if(doRows)
        {
            for (int col = 0; col < Cols; col++)
            {
                FAPDoubleList<Sheet> currentGraph = graphForNewLines[col, R];
                for (int i = 0; i < N; i++)
                {
                    int row = R - i;
                    if (currentGraph != null)
                        currentGraph = currentGraph.Copy();
                    this[col, row] = currentGraph;
                }
            }
        }
        else
        {
            for (int row = 0; row < Rows; row++)
            {
                FAPDoubleList<Sheet> currentGraph = graphForNewLines[R,row];
                for (int i = 0; i < N; i++)
                {
                    int col = R - i;
                    if (currentGraph != null)
                        currentGraph = currentGraph.Copy();

                    this[col, row] = currentGraph;
                }
            }
        }
    }

    private void ExpandSupportGraph(bool doRows, int N, int R)
    {
        int oldNumbCol = graph.GetLength(0);
        int oldNumRows = graph.GetLength(1);
        foreach (Sheet sheet in owner.workbook)
        {
            for (int c = 0; c < sheet.SupportGraph.Cols; c++)
            {
                for (int r = 0; r < sheet.SupportGraph.Rows; r++)
                {
                    if (sheet.SupportGraph[c,r] != null)
                        sheet.SupportGraph[c,r] = sheet.SupportGraph[c,r].AddLines(sheet, R, N, doRows, c, r);
                }
            }
```

```
298             }
299             FAPDoubleList<Sheet>[,] newSupportGraph;
300             if(doRows)
301                 newSupportGraph = new FAPDoubleList<Sheet>[Cols, Rows+N];
302             else
303                 newSupportGraph = new FAPDoubleList<Sheet>[Cols+N, Rows];
304 
305             if (doRows)
306             {
307                 //copy normal cells
308                 for (int r = 0; r < R; r++)
309                 {
310                     for (int c = 0; c < oldNumbCol; c++)
311                         newSupportGraph[c, r] = graph[c, r];
312                 }
313 
314                 // Move the rows R, R+1, ... later by N rows in current sheet
315                 for (int r = oldNumRows - 1; r >= R + N; r--)
316                     for (int c = 0; c < oldNumbCol; c++)
317                         newSupportGraph[c, r] = graph[c, r - N];
318 
319             }
320             else
321             {
322                 //Copy normal cells
323                 for (int c = 0; c < R; c++)
324                     for (int r = 0; r < oldNumRows; r++)
325                         newSupportGraph[c, r] = graph[c, r];
326 
327                 // Move the columns R, R+1, ... later by N columns in current sheet
328                 for (int c = oldNumbCol - 1; c >= R + N; c--)
329                     for (int r = 0; r < oldNumRows; r++)
330                         newSupportGraph[c, r] = graph[c - N, r];
331             }
332 
333             graph = newSupportGraph;
334         }
335 
336         #region IEnumerable Members
337 
338         public IEnumerator GetEnumerator()
339         {
340             return graph.GetEnumerator();
341         }
342 
343         #endregion
344 
345         public void RemoveDublicateFAPDoubleLists()
346         {
347             EqualityForFAPDoubleLists eq = new EqualityForFAPDoubleLists();
348             HashSet<FAPDoubleList<Sheet>> existingGrids = new HashSet<FAPDoubleList<Sheet>>(eq);
```

```
            for (int c = 0; c < Cols; c++)
            {
                for (int r = 0; r < Rows; r++)
                {
                    FAPDoubleList<Sheet> currentItem = this[c, r];
                    if(currentItem != null)
                    {
                        if (existingGrids.FindOrAdd(ref currentItem))
                        {
                            this[c, r] = currentItem;

                        }
                    }
                }
            }
        }

        internal class EqualityForFAPDoubleLists: IEqualityComparer<FAPDoubleList<Sheet>>
        {
            public bool Equals(FAPDoubleList<Sheet> x, FAPDoubleList<Sheet> y)
            {
                return x.EqualsAlternative(y);
            }

            public int GetHashCode(FAPDoubleList<Sheet> obj)
            {
                return obj.GetAlternativeHashCode();
            }
        }
    }


    internal struct Area
    {
        internal readonly int colStart;
        internal readonly int colEnd;
        internal readonly int rowStart;
        internal readonly int rowEnd;
        internal readonly Sheet sheet;

        internal Area(int colStart, int colEnd, int rowStart, int rowEnd, Sheet sheet)
        {
            this.colStart = colStart;
            this.colEnd = colEnd;
            this.rowStart = rowStart;
            this.rowEnd = rowEnd;
            this.sheet = sheet;
        }

        internal Area(CellArea area, Sheet currentSheet, CellAddr ca)
```

```
400         {
401             int colBegin = area.ul.colAbs ? area.ul.colRef : area.ul.colRef + ca.col;
402             int colEnding = area.lr.colAbs ? area.lr.colRef : area.lr.colRef + ca.col;
403             if(colBegin>colEnding)
404                 Tools.SwitchValues(ref colEnding, ref colBegin);
405 
406             this.colStart = colBegin;
407             this.colEnd = colEnding;
408 
409             int rowBegin= area.ul.rowAbs ? area.ul.rowRef : area.ul.rowRef + ca.row;
410             int rowEnding = area.lr.rowAbs ? area.lr.rowRef : area.lr.rowRef + ca.row;
411             if(rowBegin>rowEnding)
412                 Tools.SwitchValues(ref rowBegin, ref rowEnding);
413 
414             this.rowStart = rowBegin;
415             this.rowEnd = rowEnding;
416 
417             this.sheet = area.sheet == null ? currentSheet : area.sheet;
418         }
419 
420 
421 
422     }
423 }
424 
```

CoreCalc.SupportGraph

SupportGraphBuilder

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using C5;
using FAP;
using Logger;
using Utilities;

namespace CoreCalc.SupportGraf
{
    class SupportGraphBuilder
    {
        private const int NO_STATIC_LINE = -1;
        private static int numberOfOutOfBounds = 0;

        private static readonly Type thisType = typeof(SupportGraphBuilder);

        public static HashDictionary<Sheet, SupportGraph> BuildSupportGraph(Workbook wb, bool direction)
        {
            Log.Info(thisType, "Building supportgraph for current workbook");
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Reset();
            stopwatch.Start();
            //TODO: Move this under each sheet
            bool directionColums = FindOptimalDirection(wb);
            //bool directionColums = false;
            //TODO: Also move this under each sheet
            HashDictionary<Sheet, List<HashDictionary<Expr, FAPList>>> expMap = BuildOccurrenceMap(wb, directionColums);
            HashDictionary<Sheet, SupportGraph> back = FindSupportGraphEdges(wb, expMap, directionColums);

            stopwatch.Stop();
            TimeSpan duration = stopwatch.Elapsed;
            Log.Fatal(thisType, "Building the whole supportgraph took " + duration);

            return back;
        }

        private static bool FindOptimalDirection(Workbook wb)
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Reset();
            stopwatch.Start();
            int columnCounter = 0;
            foreach (Sheet sheet in wb)
            {
                for (int c = 0; c < sheet.Cols; c++)
                {
                    int tempColCounter = 0;
                    int tempColTotal = 0;
                    HashSet<Expr> colCollection = new HashSet<Expr>();
```

```
52                      for (int r = 0; r < sheet.Rows; r++)
53                      {
54                          if (sheet[c, r] != null)
55                          {
56                              tempColTotal++;
57                              Formula f = sheet[c, r] as Formula;
58                              if (f != null)
59                              {
60                                  if (!colCollection.Add(f.Expr))
61                                  {
62                                      tempColCounter++;
63                                  }
64                              }
65                          }
66                      }
67                      //TODO: this gives a upper limit of 10*2^16 (about 650000) active cells - but right now corecalc have a much lesser limit
68                      columnCounter += (int)(tempColCounter * tempColTotal * 0.1);
69                  }
70
71              }
72              int rowCounter = 0;
73              foreach (Sheet sheet in wb)
74              {
75                  for (int r = 0; r < sheet.Rows; r++)
76                  {
77                      int tempRowCounter = 0;
78                      int tempRowTotal = 0;
79                      HashSet<Expr> rowCollection = new HashSet<Expr>();
80                      for (int c = 0; c < sheet.Cols; c++)
81                      {
82                          if (sheet[c, r] != null)
83                          {
84                              tempRowTotal++;
85                              Formula f = sheet[c, r] as Formula;
86                              if (f != null)
87                              {
88                                  if (!rowCollection.Add(f.Expr))
89                                  {
90                                      tempRowCounter++;
91                                  }
92                              }
93                          }
94                      }
95                      //TODO: this gives a upper limit of 10*2^16 active cells - but right now corecalc have a much lesser limit
96                      rowCounter += (int)(tempRowCounter * tempRowTotal * 0.1);
97                  }
98
99              }
100             stopwatch.Stop();
101             TimeSpan duration = stopwatch.Elapsed;
```

```
102
103             Log.Info(thisType, "Checking rows contra columns took: " + duration);
104             Log.Info(thisType, "In finding optimal direction the columns got: " + columnCounter +
105                 " and the rows got: " + rowCounter);
106
107             return columnCounter > rowCounter;
108         }
109
110         private static HashDictionary<Sheet, SupportGraph> FindSupportGraphEdges(Workbook wb, HashDictionary<Sheet, List
    <HashDictionary<Expr, FAPList>>> expMap, bool directionRows)
111         {
112             Log.Info(thisType, "Finding supportgraph edges");
113
114             HashDictionary<Sheet, SupportGraph> realSupportGraphs = new HashDictionary<Sheet, SupportGraph>();
115             foreach (Sheet sheet in wb)
116             {
117                 SupportGraph graph = new SupportGraph(sheet.Cols, sheet.Rows, sheet);
118                 realSupportGraphs.Add(sheet, graph);
119             }
120
121             CalculateEdges(directionRows, expMap, realSupportGraphs, NO_STATIC_LINE, null);
122
123             return realSupportGraphs;
124         }
125
126         private static void CalculateEdges(bool directionRows, HashDictionary<Sheet, List<HashDictionary<Expr, FAPList>>> expMap,
    HashDictionary<Sheet, SupportGraph> realSupportGraphs, int staticLine, Sheet addLinesTo)
127         {
128
129             foreach (C5.KeyValuePair<Sheet, List<HashDictionary<Expr, FAPList>>> sheet in expMap)
130             {
131
132                 for (int staticCounter = 0; staticCounter < sheet.Value.Count; staticCounter++)
133                 {
134                     HashDictionary<Expr, FAPList> colRow = sheet.Value[staticCounter];
135                     {
136                         if (colRow != null)
137                         {
138                             FAPSet staticSet = new FAPSet(staticCounter, 1, 1);
139                             foreach (C5.KeyValuePair<Expr, FAPList> pair in colRow)
140                             {
141                                 if (staticLine == NO_STATIC_LINE)
142                                     SupportGraphEdgeBuilder(sheet.Key, staticSet, realSupportGraphs, pair, directionRows, false);
143                                 else
144                                     SupportGraphAddLines(sheet.Key, staticSet, realSupportGraphs, pair, directionRows, staticLine,
    addLinesTo, false);
145                             }
146                         }
147
148                     }
149                 }
```

```
150             }
151         }
152 
153         private static void SupportGraphAddLines(Sheet currentSheet, FAPSet staticSet, HashDictionary<Sheet, SupportGraph> graph, ↙
        C5.KeyValuePair<Expr, FAPList> pair, bool directionRows,
154                                          int staticLine, Sheet addLinesTo, bool insertDirectlyIntoRealGraph)
155         {
156             Expr currentExpression = pair.Key;
157             ArrayList<CellArea> cells = AddressCollector.GetAllAdresses<ArrayList<CellArea>>(currentExpression);
158             HashDictionary<VisitedCellsKey, ArrayList<FAPSet>> visitedCells = null;
159             if (cells.Count != 1)
160                 visitedCells = new HashDictionary<VisitedCellsKey, ArrayList<FAPSet>>();
161             foreach (CellArea adrs in cells)
162             {
163                 if (adrs.sheet == addLinesTo || (adrs.sheet == null && currentSheet == addLinesTo))
164                 {
165                     RARef start = adrs.ul;
166                     RARef end = adrs.lr;
167                     TwoDimensionSpan span;
168                     int staticStart;
169                     int staticEnd;
170                     staticStart = staticEnd = staticLine;
171                     if (directionRows)
172                     {
173                         span = new TwoDimensionSpan(staticStart, staticEnd, start.rowRef, end.rowRef,
174                                                     start.rowAbs, end.rowAbs, adrs.sheet, directionRows, true);
175                     }
176                     else
177                     {
178                         span = new TwoDimensionSpan(staticStart, staticEnd, start.colRef, end.colRef,
179                                                     start.colAbs, end.colAbs, adrs.sheet, directionRows, true);
180                     }
181                     InformationCarrier info = new InformationCarrier(currentSheet, graph, span,
182                         staticSet, insertDirectlyIntoRealGraph, visitedCells);
183                     FindAndUseCaseOfSupportEdgesBuilder(pair, info);
184                 }
185             }
186         }
187 
188         internal SupportGraph GetSupportGraphForAddedLines(Sheet sheet, HashDictionary<Sheet, List<HashDictionary<Expr, FAPList>>>↙
         expMap, bool directionRows, int staticLine)
189         {
190             HashDictionary<Sheet, SupportGraph> supportGraph = new HashDictionary<Sheet, SupportGraph>();
191             supportGraph.Add(sheet, new SupportGraph(sheet.Cols, sheet.Rows, sheet));
192 
193             CalculateEdges(directionRows, expMap, supportGraph, staticLine, sheet);
194 
195             return supportGraph[sheet];
196         }
197 
198         internal static void HandleVolatileFunctions(FAPSet staticSet, FAPList dynamic, Sheet currentSheet, bool direction)
```

```
199         {
200             Log.Debug(thisType, "Contains volatile function: " + staticSet + " x " + dynamic);
201             FAPDoubleList<Sheet> volatileFunctions = new FAPDoubleList<Sheet>();
202             FAPList staticList = new FAPList(staticSet, true);
203             if (direction)
204             {
205                 volatileFunctions = volatileFunctions.Union(staticList, dynamic, currentSheet);
206             }
207             else
208             {
209                 volatileFunctions = volatileFunctions.Union(dynamic, staticList, currentSheet);
210             }
211             ArrayList<FullCellAddr<Sheet>> volatileFunctionsAddressres = volatileFunctions.GetAdressesAbsolute();
212             currentSheet.workbook.VolatileExpressions.AddAll(volatileFunctionsAddressres);
213         }
214 
215         internal static void SupportGraphEdgeBuilder(Sheet currentSheet, FAPSet staticSet, HashDictionary<Sheet, SupportGraph>
    graph, C5.KeyValuePair<Expr, FAPList> pair, bool directionRows, bool insertDirectlyIntoRealGraph)
216         {
217             Expr currentExpression = pair.Key;
218             if (currentSheet.ContainsVolatileFunction(currentExpression))
219             {
220                 HandleVolatileFunctions(staticSet, pair.Value, currentSheet, directionRows);
221             }
222             ArrayList<CellArea> cells = AddressCollector.GetAllAdresses <ArrayList<CellArea>>(currentExpression);
223 
224             HashDictionary<VisitedCellsKey, ArrayList<FAPSet>> visitedCells = null;
225             if (cells.Count != 1)
226                 visitedCells = new HashDictionary<VisitedCellsKey, ArrayList<FAPSet>>();
227             foreach(CellArea adrs in cells)
228             {
229                 RARef start = adrs.ul;
230                 RARef end = adrs.lr;
231                 TwoDimensionSpan span;
232                 int staticStart;
233                 int staticEnd;
234                 if (directionRows)
235                 {
236                     bool absoluteStatic = start.colAbs || end.colAbs;
237 
238                     staticStart = start.colAbs ? start.colRef : start.colRef + staticSet.A;
239                     staticEnd = end.colAbs ? end.colRef : end.colRef + staticSet.A;
240                     if(staticStart>staticEnd)
241                     {
242                         Tools.SwitchValues(ref staticEnd, ref staticStart);
243                     }
244 
245                     span = new TwoDimensionSpan(staticStart, staticEnd, start.rowRef, end.rowRef,
246                                                 start.rowAbs, end.rowAbs, adrs.sheet, directionRows, absoluteStatic);
247                 }
248                 else
```

```
249                 {
250                     bool absoluteStatic = start.rowAbs || end.rowAbs;
251 
252                     staticStart = start.rowAbs ? start.rowRef : start.rowRef + staticSet.A;
253                     staticEnd = end.rowAbs ? end.rowRef : end.rowRef + staticSet.A;
254                     if (staticStart > staticEnd)
255                     {
256                         Tools.SwitchValues(ref staticEnd, ref staticStart);
257                     }
258 
259                     span = new TwoDimensionSpan(staticStart, staticEnd, start.colRef, end.colRef,
260                                                 start.colAbs, end.colAbs, adrs.sheet, directionRows, absoluteStatic);
261                 }
262                 InformationCarrier info = new InformationCarrier(currentSheet, graph, span, staticSet,
263                     insertDirectlyIntoRealGraph, visitedCells);
264                 FindAndUseCaseOfSupportEdgesBuilder(pair, info);
265             }
266         }
267 
268 
269 
270         internal static void FindAndUseCaseOfSupportEdgesBuilder(C5.KeyValuePair<Expr, FAPList> pair, InformationCarrier info)
271         {
272             if(info.span.dynamicNormalEnd)
273             {
274                 if(info.span.dynamicNormalStart)
275                 {
276                     foreach (FAPSet set in pair.Value)
277                     {
278                         GetSupportGraphEdgesCase1(set, info);
279                     }
280                 }
281                 else
282                 {
283                     foreach (FAPSet set in pair.Value)
284                     {
285                         GetSupportGraphEdgesCase2(set, info);
286                     }
287                 }
288             }
289             else if(info.span.dynamicNormalStart)
290             {
291                 foreach (FAPSet set in pair.Value)
292                 {
293                     GetSupportGraphEdgesCase3(set, info);
294                 }
295             }
296             else
297             {
298                 foreach (FAPSet set in pair.Value)
299                 {
```

```
300                     if (info.span.normEnd - info.span.normStart <= set.B)
301                     {
302                         Case4Simple(set, info);
303                     }
304                     else
305                     {
306                         GetSupportGraphEdgesCase4(set, info);
307                     }
308                 }
309             }
310         }
311 
312         private static void Case4Simple(FAPSet set, InformationCarrier info)
313         {
314             int nomStart = info.span.normStart;
315             int nomEnd = info.span.normEnd;
316             if (nomStart > nomEnd)
317                 Tools.SwitchValues(ref nomEnd, ref nomStart);
318 
319             HashDictionary<FAPDoubleListMapper, FAPDoubleList<Sheet>> changeMap = new HashDictionary<FAPDoubleListMapper,
    FAPDoubleList<Sheet>>();
320             for(int colRowNr = info.span.staticStart; colRowNr<=info.span.staticEnd; colRowNr++)
321             {
322                 for (int i = 0; i < set.K; i++)
323                 {
324                     for (int j = nomStart; j <= nomEnd; j++)
325                     {
326                         int rrow = set.A + set.B*i;
327                         int rowColumn = rrow + j;
328                         AddToGraph(new FAPSet(rrow-rowColumn, 1, 1), colRowNr, rowColumn, info, changeMap, false);//minus
    rowcolumn because its relative
329                     }
330 
331                 }
332             }
333         }
334 
335         private static void GetSupportGraphEdgesCase4(FAPSet set, InformationCarrier info)
336         {
337             int dynamicStart = set.A + info.span.normStart;
338             int dynamicEnd = set.A + set.B * (set.K - 1) + info.span.normEnd;
339             if (dynamicStart > dynamicEnd)
340                 Tools.SwitchValues(ref dynamicStart, ref dynamicEnd);
341             HashDictionary<FAPDoubleListMapper, FAPDoubleList<Sheet>> changeMap = new HashDictionary<FAPDoubleListMapper,
    FAPDoubleList<Sheet>>();
342             for (int colRowNr = info.span.staticStart; colRowNr <= info.span.staticEnd; colRowNr++)
343             {
344                 for (int r = dynamicStart; r <= dynamicEnd; r++)
345                 {
346                     int k1 = Math.Max(0, (r - info.span.normEnd - set.A + set.B - 1)/set.B);
347                     int k2 = Math.Min(set.K, (r - info.span.normStart - set.A + set.B)/set.B);
```

```
348                     FAPSet add = set;
349                     add.A = add.A + add.B*k1-r;//-r because its relative
350                     add.K = k2 - k1;
351 
352                     AddToGraph(add, colRowNr, r, info, changeMap, false);
353                 }
354             }
355         }
356 
357         /// <summary>
358         /// Case where upper bound is relative (lowest value)
359         /// </summary>
360         /// <param name="info"></param>
361         /// <param name="set"></param>
362         private static void GetSupportGraphEdgesCase2(FAPSet set, InformationCarrier info)
363         {
364             int rowUpper = info.span.normStart + set.A;
365             int nomEnd = info.span.normEnd;
366             if (rowUpper > nomEnd)
367                 Tools.SwitchValues(ref rowUpper, ref nomEnd);
368             //Log.Debug(this, "case2: upper is relative. " + "At addr: " + adrs.ul.rowRef + "," + adrs.lr.rowRef + " The set: " + set);
369             HashDictionary<FAPDoubleListMapper, FAPDoubleList<Sheet>> changeMap = new HashDictionary<FAPDoubleListMapper, FAPDoubleList<Sheet>>();
370             for (int i = info.span.staticStart; i <= info.span.staticEnd; i++)
371             {
372                 for (int j = rowUpper; j <= nomEnd; j++)
373                 {
374                     int k1 = (j - rowUpper)/set.B + 1; //Notice how the first row only gets k = 1.
375                     FAPSet add = set;
376                     add.K = Math.Min(set.K, k1);
377                     AddToGraph(add, i, j, info, changeMap, true);
378                 }
379             }
380         }
381 
382         /// <summary>
383         /// Case where lower bound is relative
384         /// </summary>
385         private static void GetSupportGraphEdgesCase3(FAPSet set, InformationCarrier info)
386         {
387             int lowRow = info.span.normEnd + set.A + set.B * (set.K - 1);
388             int nomStart = info.span.normStart;
389             if (nomStart > lowRow)
390                 Tools.SwitchValues(ref lowRow, ref nomStart);
391             //Log.Debug(this, "Case3: lower is relative. " + "At addr: " + adrs.ul.rowRef + "," + adrs.lr.rowRef + " The set: " + set);
392             HashDictionary<FAPDoubleListMapper, FAPDoubleList<Sheet>> changeMap = new HashDictionary<FAPDoubleListMapper, FAPDoubleList<Sheet>>();
393             for (int i = info.span.staticStart; i <= info.span.staticEnd; i++)
394             {
```

```
395                 for (int j = nomStart; j <= lowRow; j++)
396                 {
397                     //TODO: the calculations with set can be moved out of loop
398                     //TODO: check if this is right
399                     int k1 = Math.Min(set.K, ((lowRow-j)/set.B)+1);
400                     FAPSet add = set;
401                     add.K = k1;
402                     add.A = set.A + set.B*(set.K-k1);
403                     AddToGraph(add, i, j, info, changeMap, true);
404                 }
405             }
406 
407         }
408 
409         private static void GetSupportGraphEdgesCase1(FAPSet set, InformationCarrier info)
410         {
411             int nomStart = info.span.normStart;
412             int nomEnd = info.span.normEnd;
413             if (nomStart > nomEnd)
414                 Tools.SwitchValues(ref nomEnd, ref nomStart);
415             HashDictionary<FAPDoubleListMapper, FAPDoubleList<Sheet>> changeMap = new HashDictionary<FAPDoubleListMapper,
    FAPDoubleList<Sheet>>();
416             for (int i = info.span.staticStart; i <= info.span.staticEnd; i++)
417             {
418                 for (int j = nomStart; j <= nomEnd; j++)
419                 {
420                     AddToGraph(set, i, j, info, changeMap, true);
421                 }
422             }
423         }
424 
425         private static void AddToGraph(FAPSet set, int col, int row, InformationCarrier info, HashDictionary<FAPDoubleListMapper,
    FAPDoubleList<Sheet>> changeMap,
426                                        bool dynamicAbsoluteReference)
427         {
428 
429             //fapset1 equals staticset in info
430             Sheet sheet = info.span.sheet;
431             if (sheet == null)
432                 sheet = info.currentSheet;
433 
434             FAPSet fSet1;
435             if (!info.span.staticAbsolute)
436                 fSet1 = new FAPSet(info.staticSet.A - col, info.staticSet.B, info.staticSet.K);
437             else
438                 fSet1 = info.staticSet;
439             FAPSet fSet2 = set;
440 
441 
442 
443             FAPSet temp;
```

```
444
445             if (!info.span.directionRows)
446             {
447
448                 int tCol = col;
449                 col = row;
450                 row = tCol;
451                 temp = fSet1;
452                 fSet1 = fSet2;
453                 fSet2 = temp;
454             }
455
456
457             SupportGraph addToSheet;
458             if (info.insertDirectlyIntoRealGraph)
459                 addToSheet = sheet.SupportGraph;
460             else
461                 addToSheet = info.graph[sheet];
462
463
464             if (col >= addToSheet.Cols || row >= addToSheet.Rows)
465             {
466                 if (numberOfOutOfBounds < 100)
467                     Log.Debug(thisType, "Tried to add a doubleFapList to a cell outside the bounds of the sheet. \tSheet is: {0}. ↙
    \tCell was: ({1}, {2})",
468                                 sheet.Name, col, row);
469                 numberOfOutOfBounds++;
470                 return;
471             }
472
473             FAPDoubleList<Sheet> current = addToSheet[col, row];
474
475             FAPDoubleList<Sheet> newFAPGrid;
476             if (changeMap.Find(new FAPDoubleListMapper(current, fSet1, fSet2, info.currentSheet, dynamicAbsoluteReference, info. ↙
    span.staticAbsolute), out newFAPGrid))
477             {
478                 addToSheet[col, row] = newFAPGrid;
479                 if (col == 2 && row == 9 && addToSheet[col, row].GetEdgeCount() == 0)
480                     Log.Fatal(thisType, "Should not happen");
481                 return; //we just copied the reference instead of calculating
482             }
483
484             bool set1Abs;
485             bool set2Abs;
486             if (info.span.directionRows)
487             {
488                 set1Abs = info.span.staticAbsolute;
489                 set2Abs = dynamicAbsoluteReference;
490
491             }
492             else
```

```
493             {
494                 set1Abs = dynamicAbsoluteReference;
495                 set2Abs = info.span.staticAbsolute;
496             }
497             FAPList set1 = new FAPList(fSet1, set1Abs);
498             FAPList set2 = new FAPList(fSet2, set2Abs);
499 
500 
501             if (info.visitedCells != null)
502             {
503                 //We do not care about info.span.directionRows, the adress will be unique anyway
504                 FullCellAddr<Sheet> currentAddr = new FullCellAddr<Sheet>(col, row, sheet);
505                 FAPList alter = info.span.directionRows ? set2 : set1;
506                 VisitedCellsKey key = new VisitedCellsKey(currentAddr, dynamicAbsoluteReference, info.span.staticAbsolute, info.staticSet.A);
507 
508                 ArrayList<FAPSet> tmpList;
509 
510                 if (info.visitedCells.Find(ref key, out tmpList))
511                 {
512                     foreach (FAPSet removeSet in tmpList)
513                     {
514                        alter.RemoveFAPSet(removeSet);
515                     }
516                     tmpList.AddAll(alter);
517                 }
518                 else
519                 {
520                     ArrayList<FAPSet> setsAdded = new ArrayList<FAPSet>();
521                     setsAdded.AddAll(alter);
522                     info.visitedCells.Add(key, setsAdded);
523                 }
524             }
525 
526 
527                 if (current != null)
528                 {
529                     addToSheet[col, row] = current.Union(set1, set2, info.currentSheet);
530                 }
531                 else
532                     addToSheet[col, row] = new FAPDoubleList<Sheet>(set1, set2, info.currentSheet);
533 
534                 if (col == 2 && row == 9 && addToSheet[col, row].GetEdgeCount() == 0)
535                     Log.Fatal(thisType, "Should not happen");
536 
537                 changeMap.Add(new FAPDoubleListMapper(current, fSet1, fSet2, info.currentSheet, dynamicAbsoluteReference, info.span.staticAbsolute), addToSheet[col, row]);
538 
539 
540 
541             }
```

```
542
543         private static HashDictionary<Sheet, List<HashDictionary<Expr, FAPList>>> BuildOccurrenceMap(Workbook wb, bool directionRows)
544         {
545             DateTime startTime = DateTime.Now;
546             Log.Info(thisType, "Building copy-invariant expression occurrencemap");
547             //TODO: implement the solution to the "This scheme can be defeated by..." on page 62 (do this in the faplist class)
548             HashDictionary<Sheet, List<HashDictionary<Expr, FAPList>>> expressionOccurrenceMapForSheets = new HashDictionary<Sheet, List<HashDictionary<Expr, FAPList>>>();
549             int numberOfUnions = 0;
550             Formats fo = new Formats();
551             fo.RefFmt = Formats.RefType.A1;
552             foreach (Sheet sheet in wb)
553             {
554                 expressionOccurrenceMapForSheets.Add(sheet, new List<HashDictionary<Expr, FAPList>>());
555                 if (sheet != null)
556                 {
557                     int dimensionStatic = directionRows ? sheet.Cols : sheet.Rows;
558                     int dimensionDynamic = directionRows ? sheet.Rows : sheet.Cols;
559
560                     for (int i = 0; i < dimensionStatic; i++)
561                     {
562                         HashDictionary<Expr, FAPList> currentColumn = new HashDictionary<Expr, FAPList>();
563                         for (int j = 0; j < dimensionDynamic; j++)
564                         {
565                             int col = directionRows ? i : j;
566                             int row = directionRows ? j : i;
567                             Cell c = sheet[col, row];
568                             if (c != null)
569                             {
570                                 Formula formula = c as Formula;
571                                 //TODO: can c be a MatrixFomula? A scan of the code only shows usage of them when inserting columns and rows
572                                 if (formula != null)
573                                 {
574                                     if (AddressCollector.GetAllAdresses<ArrayList<CellArea>>(formula.Expr).Count != 0)
575                                         BuildOccurenceMapFromFormulaCell(formula.Expr, currentColumn, j);
576                                 }
577                             }
578                         }
579                         expressionOccurrenceMapForSheets[sheet].Add(currentColumn);
580                     }
581                 }
582             }
583
584             DateTime stopTime = DateTime.Now;
585             TimeSpan duration = stopTime - startTime;
586             Log.Info(thisType, "Number of unions performed: " + numberOfUnions);
587             Log.Info(thisType, "Building the expressionmap supportgraph took " + duration);
588             return expressionOccurrenceMapForSheets;
589         }
```

```
590
591         private static void BuildOccurenceMapFromFormulaCell(Expr expression, HashDictionary<Expr, FAPList>
        currentColumnExpressionMap, int j)
592         {
593             //TODO: A1:A$1 or A$1:A1 != the same expression as a cellarea in regards to internal representation
594             //of expressions?
595             //maybe because the first is represented as a single cell type, where the other is a cellarea type.
596             FAPList use;
597             if (currentColumnExpressionMap.Find(expression, out use))
598             {
599                 //Just union the fap set, the faplist will take care of the rest
600                 use.AddWhenBuildingSupportGraph(j);
601             }
602             else
603             {
604                 FAPList list = new FAPList(j, 1, 1, true);
605                 currentColumnExpressionMap.Add(expression, list);
606             }
607         }
608
609         private void PrintOccurenceMap(HashDictionary<Sheet, List<HashDictionary<Expr, FAPList>>> expMap)
610         {
611             Formats fo = new Formats();
612             fo.RefFmt = Formats.RefType.A1;
613             StringBuilder sb = new StringBuilder();
614             int counter = 0;
615             foreach (C5.KeyValuePair<Sheet, List<HashDictionary<Expr, FAPList>>> sheet in expMap)
616             {
617                 sb.Append("\r\n- " + sheet.Key.Name);
618                 foreach (HashDictionary<Expr, FAPList> col in sheet.Value)
619                 {
620                     sb.Append("\r\n-- Column " + CellAddr.ColumnName(counter));
621                     foreach (C5.KeyValuePair<Expr, FAPList> pair in col)
622                     {
623                         sb.Append("\r\nKey example: " + pair.Key.Show(1, 1, 0, fo) + " Fap list:\r\n" +
624                                   pair.Value.ToString());
625                     }
626                     counter++;
627                 }
628                 Log.Info(typeof(SupportGraphBuilder), sb.ToString());
629                 sb.Length = 0;
630                 counter = 0;
631             }
632         }
633
634         internal static void ApplySupportGraphs(HashDictionary<Sheet, SupportGraph> realSupportGraphs)
635         {
636             foreach (C5.KeyValuePair<Sheet, SupportGraph> pair in realSupportGraphs)
637             {
638                 pair.Key.SupportGraph = pair.Value;
639             }
```

```
640         }
641     }
642 
643     internal struct VisitedCellsKey:IEquatable<VisitedCellsKey>
644     {
645         public readonly FullCellAddr<Sheet> currentAddr;
646         public readonly bool dynamicAbsolute;
647         public readonly bool staticAbsolute;
648         public readonly int staticNumber;
649 
650         internal VisitedCellsKey(FullCellAddr<Sheet> currentAddr, bool dynamicAbsolute, bool staticAbsolute,
651         int staticNumber)
652         {
653             this.currentAddr = currentAddr;
654             this.dynamicAbsolute = dynamicAbsolute;
655             this.staticAbsolute = staticAbsolute;
656             this.staticNumber = staticNumber;
657         }
658 
659         public override int GetHashCode()
660         {
661             int hashCode = 0;
662             hashCode = currentAddr.GetHashCode();
663             hashCode = hashCode * 29 + (dynamicAbsolute ? 1 : 0);
664             hashCode = hashCode * 29 + (staticAbsolute ? 1 : 0);
665             hashCode = hashCode * 29 + staticNumber;
666             return hashCode;
667         }
668 
669         public override bool Equals(object obj)
670         {
671             if(obj is VisitedCellsKey)
672                 return Equals((VisitedCellsKey)obj);
673             else
674                 return false;
675         }
676 
677         public bool Equals(VisitedCellsKey other)
678         {
679             if (staticNumber == other.staticNumber &&
680                 this.staticAbsolute == other.staticAbsolute &&
681                 this.dynamicAbsolute == other.dynamicAbsolute &&
682                 this.currentAddr.Equals(other.currentAddr))
683                 return true;
684             else
685                 return false;
686         }
687 
688     }
689 
690     internal struct TwoDimensionSpan
```

```
691     {
692         internal readonly int staticStart;
693         internal readonly int staticEnd;
694         internal readonly int normStart;
695         internal readonly int normEnd;
696         internal readonly bool dynamicNormalStart;
697         internal readonly bool dynamicNormalEnd;
698         internal readonly Sheet sheet;
699         internal readonly bool directionRows;
700         internal readonly bool staticAbsolute;
701 
702         public TwoDimensionSpan(int staticStart, int staticEnd, int normStart, int normEnd, bool dynamicNormalStart, bool dynamicNormalEnd, Sheet sheet, bool directionRows, bool staticAbsolute)
703         {
704             this.staticStart = staticStart;
705             this.staticEnd = staticEnd;
706             this.normStart = normStart;
707             this.normEnd = normEnd;
708             this.dynamicNormalStart = dynamicNormalStart;
709             this.dynamicNormalEnd = dynamicNormalEnd;
710             this.sheet = sheet;
711             this.directionRows = directionRows;
712             this.staticAbsolute = staticAbsolute;
713         }
714 
715     }
716 
717     internal struct InformationCarrier
718     {
719         internal readonly Sheet currentSheet;
720         internal readonly HashDictionary<Sheet, SupportGraph> graph;
721         internal readonly TwoDimensionSpan span;
722         internal readonly FAPSet staticSet;
723         internal readonly bool insertDirectlyIntoRealGraph;
724         internal readonly HashDictionary<VisitedCellsKey, ArrayList<FAPSet>> visitedCells;
725 
726         internal InformationCarrier(Sheet currentSheet, HashDictionary<Sheet, SupportGraph> graph, TwoDimensionSpan span, FAPSet staticSet,
727             bool insertDirectlyIntoRealGraph, HashDictionary<VisitedCellsKey, ArrayList<FAPSet>> visitedCells)
728         {
729             this.currentSheet = currentSheet;
730             this.graph = graph;
731             this.span = span;
732             this.staticSet = staticSet;
733             this.insertDirectlyIntoRealGraph = insertDirectlyIntoRealGraph;
734             this.visitedCells = visitedCells;
735         }
736     }
737 
738     internal struct FAPDoubleListMapper:IEquatable<FAPDoubleListMapper>
739     {
```

```
740         internal readonly FAPDoubleList<Sheet> orgFAPGrid;
741         internal readonly FAPSet newStatic;
742         internal readonly FAPSet newDynamic;
743         internal readonly bool dynamicAbsolute;
744         internal readonly bool staticAbsolute;
745         internal readonly Sheet newSheet;
746 
747 
748         public FAPDoubleListMapper(FAPDoubleList<Sheet> orgFAPGrid, FAPSet newStatic, FAPSet newDynamic, Sheet newSheet, bool dynamicAbsolute, bool staticAbsolute)
749         {
750             this.orgFAPGrid = orgFAPGrid;
751             this.newStatic = newStatic;
752             this.newDynamic = newDynamic;
753             this.newSheet = newSheet;
754             this.dynamicAbsolute = dynamicAbsolute;
755             this.staticAbsolute = staticAbsolute;
756         }
757 
758         public override int GetHashCode()
759         {
760             int result = orgFAPGrid != null ? orgFAPGrid.GetHashCode() : 0;
761             result = 29*result + newStatic.GetHashCode();
762             result = 29*result + newDynamic.GetHashCode();
763             result = 29*result + newSheet.GetHashCode();
764             result = 29*result + (dynamicAbsolute ? 1 : 0);
765             result = 29 * result + (staticAbsolute ? 1 : 0);
766             return result;
767         }
768 
769         public override bool Equals(object obj)
770         {
771             if (ReferenceEquals(this, obj)) return true;
772             if (!(obj is FAPDoubleListMapper)) return false;
773             FAPDoubleListMapper fapDoubleListMapper = (FAPDoubleListMapper) obj;
774             return Equals(fapDoubleListMapper);
775         }
776 
777         #region IEquatable<FAPDoubleListMapper> Members
778 
779         public bool Equals(FAPDoubleListMapper other)
780         {
781             if (orgFAPGrid != other.orgFAPGrid ||
782                 newStatic != other.newStatic ||
783                 newDynamic != other.newDynamic ||
784                 newSheet != other.newSheet ||
785                 dynamicAbsolute != other.dynamicAbsolute ||
786                 staticAbsolute != other.staticAbsolute)
787                 return false;
788             else
789                 return true;
```

```
790             }
791
792             #endregion
793         }
794 }
795
```

CoreCalc.SupportGraph

Test

```
using System;
using System.Diagnostics;
using System.Text;
using C5;
using FAP;
using Logger;

namespace CoreCalc.SupportGraf
{
    static class Test
    {
        private const int numberOfTestsToPerform = 10;

        private static Sheet GetSheetByName(string name, Workbook wb)
        {
            foreach (Sheet sheet in wb)
            {
                if(sheet.Name == name) return sheet;
            }
            return null;
        }

        //private static FullCellAddr

        private static readonly Type THIS_TYPE = typeof(Test);

        public static string DummyTestAllSupportGraphReferencesContraEdges(Workbook wb, bool printDirectlyToLog)
        {
            StringBuilder sb = new StringBuilder();


            foreach (Sheet sheet in wb)
            {
                StringBuilder tempBuilder = new StringBuilder();
                SupportGraph sGraph = sheet.SupportGraph;
                for (int c = 0; c < sheet.Cols; c++)
                {
                    for (int r = 0; r < sheet.Rows; r++)
                    {
                        FAPDoubleList<Sheet> currentGraph = sGraph[c, r];
                        if(currentGraph != null)
                        {
                            ArrayList<FullCellAddr<Sheet>> supportedCells = currentGraph.GetAdresses(c, r);
                            FullCellAddr<Sheet> currentAddr = new FullCellAddr<Sheet>(c, r, sheet);
                            foreach (FullCellAddr<Sheet> cell in supportedCells)
                            {
                                Formula form = cell.sheet[cell.col, cell.row] as Formula;

                                if(form == null)
                                {
                                    tempBuilder.Append("The supported cell did not contain a formula. The current cell was: " +
```

```
                                        cell);
                                    tempBuilder.AppendLine("The searched cell was: " + cell.sheet.Name + "   C" + cell.col + "R" +
                                        cell.row);
                                }
                                else
                                {

                                    HashSet<FullCellAddr<Sheet>> adrs = new HashSet<FullCellAddr<Sheet>>();
                                    CustemizedAdressCollector<FullCellAddr<Sheet>>.GetAdresses((delegate(CellArea cellArea)
                                    {
                                        Area area = new Area(cellArea, sheet, new CellAddr(cell.col, cell.row));
                                        for (int cNew = area.colStart;
                                             cNew <= area.colEnd;
                                             cNew++)
                                        {
                                            for (int rNew = area.rowStart;
                                                 rNew <= area.rowEnd;
                                                 rNew++)
                                            {
                                                adrs.Add(new FullCellAddr<Sheet>(cNew, rNew, area.sheet));
                                            }
                                        }
                                    }), form.Expr);

                                    if(!adrs.Contains(currentAddr))
                                    {
                                        tempBuilder.Append("The supported cell did not contain a reference to the cell.");
                                        tempBuilder.AppendLine("The current cell was: " + cell);
                                        tempBuilder.AppendLine("The searched cell was: " + cell.sheet.Name + "   C" + cell.col + "R" +
                                            cell.row);
                                    }
                                }
                            }
                        }
                        if (tempBuilder.Length != 0)
                        {
                            if (printDirectlyToLog)
                            {
                                Log.Error(THIS_TYPE, tempBuilder.ToString());
                            }
                            else
                            {
                                sb.Append(tempBuilder.ToString());
                            }
                        }

                    }
                }
            }
```

```
101             return sb.ToString();
102         }
103 
104         public static string DummyTestAllSupportGraphEdgesContraReferences(Workbook wb, bool printDirectlyToLog, bool dontCareAboutOutsideRefs)
105         {
106             StringBuilder sb = new StringBuilder();
107             foreach (Sheet sheet in wb)
108             {
109                 for (int c = 0; c < sheet.Cols; c++)
110                 {
111                     for (int r = 0; r < sheet.Rows; r++)
112                     {
113                         Formula currentCell = sheet[c, r] as Formula;
114 
115                         if(currentCell != null)
116                         {
117                             FullCellAddr<Sheet> currentFullCellAdr = new FullCellAddr<Sheet>(c, r, sheet);
118                             StringBuilder tempBuilder = new StringBuilder();
119                             ArrayList<CellArea> adrs = AddressCollector.GetAllAdresses<ArrayList<CellArea>>(currentCell.Expr);
120                             foreach (CellArea adr in adrs)
121                             {
122                                 Sheet currentSheet = adr.sheet == null ? sheet : adr.sheet;
123                                 Area area = new Area(adr, currentSheet, new CellAddr(c, r));
124 
125                                 for (int cA = area.colStart; cA <= area.colEnd; cA++)
126                                 {
127                                     for (int rA = area.rowStart; rA <= area.rowEnd; rA++)
128                                     {
129                                         if (area.sheet.Cols < cA || area.sheet.Rows < rA)
130                                         {
131                                             if(!dontCareAboutOutsideRefs)
132                                             tempBuilder.AppendLine(
133                                                 "Either column nr or row nr on ref was outside bounds for cell: Sheet: " + sheet.Name + " Col:" +
134                                                 c + " Row: " + r);
135                                         }
136                                         else
137                                         {
138                                             HashSet<FullCellAddr<Sheet>> back = new HashSet<FullCellAddr<Sheet>>();
139                                             FAPDoubleList<Sheet> currentSupGraph = area.sheet.SupportGraph[cA, rA];
140                                             if (currentSupGraph == null)
141                                             {
142                                                 tempBuilder.Append(
143                                                     "The cell was not found!, Did not have supportgraph. Current cell: Sheet: " + sheet.Name +
144                                                     " Col:" + c +
145                                                     " Row:" + r);
146                                                 tempBuilder.AppendLine("\t The wished cell was: Sheet: " +
147                                                                        area.sheet.Name + " Col:" + cA +
148                                                                        " Row:" + rA);
```

```
                                        }
                                        else
                                        {
                                            area.sheet.SupportGraph[cA, rA].GetAdresses(cA, rA, ref back);

                                            if (!back.Find(ref currentFullCellAdr))
                                            {
                                                tempBuilder.Append(
                                                    "The cell was not found!. Current cell: Sheet: " + sheet.Name +
                                                    " Col:" + c +
                                                    " Row:" + r);
                                                tempBuilder.AppendLine("\t The wished cell was: Sheet: " +
                                                                       area.sheet.Name + " Col:" + cA +
                                                                       " Row:" + rA);
                                            }
                                        }

                                    }

                                    if (tempBuilder.Length != 0)
                                    {
                                        if (printDirectlyToLog)
                                            Log.Error(THIS_TYPE, tempBuilder.ToString());
                                        else
                                            sb.Append(tempBuilder);
                                    }
                                }
                            }
                        }
                    }
                }
            }

            return sb.ToString();
        }

        public static string PrintMemoryRelatedInfo(Workbook wb)
        {
            StringBuilder sb = new StringBuilder();
            sb.AppendLine("\r\nShowing memory related information:\r\n");
            int fapDoubleSetCounter = 0;
            int fapSetCounter = 0;
            int fapDoubleListCounter = 0;
            int edgesCount = 0;
            int totalCellCount = 0;
            int totalNonNullCellCount = 0;
            int totalReuseOfFAPDoubleList = 0;
            int fapSetsHighest = 0;
            int totalUsedFAPsetCount = 0;
            int realEdgeCount = 0;
```

```
200
201             foreach (Sheet sheet in wb)
202             {
203                 totalCellCount += sheet.Cols * sheet.Rows;
204                 for (int c = 0; c < sheet.Cols; c++)
205                 {
206                     for (int r = 0; r < sheet.Rows; r++)
207                     {
208                         Cell cell = sheet[c, r];
209                         if (cell != null)
210                         {
211                             totalNonNullCellCount++;
212                         }
213                     }
214                 }
215                 HashSet<FAPDoubleList<Sheet>> gridDictionary = new HashSet<FAPDoubleList<Sheet>>();
216                 for (int c = 0; c < sheet.Cols; c++)
217                 {
218                     for (int r = 0; r < sheet.Rows; r++)
219                     {
220                         FAPDoubleList<Sheet> fapDoubleList = sheet.SupportGraph[c, r];
221
222                         if (fapDoubleList != null)
223                         {
224                             fapDoubleListCounter++;
225                             fapDoubleSetCounter += fapDoubleList.CountFapDoubleSets();
226                             int fapSets = fapDoubleList.CountFapSets();
227                             fapSetCounter += fapSets;
228                             if (fapSets > fapSetsHighest)
229                                 fapSetsHighest = fapSets;
230
231                             edgesCount += fapDoubleList.GetEdgeCount();
232                             HashSet<FullCellAddr<Sheet>> tempSet = new HashSet<FullCellAddr<Sheet>>();
233                             fapDoubleList.GetAdresses(c, r, ref tempSet);
234                             realEdgeCount += tempSet.Count;
235
236                             FAPDoubleList<Sheet> list = fapDoubleList;
237                             if (gridDictionary.Find(ref list))
238                                 totalReuseOfFAPDoubleList++;
239                             else
240                                 gridDictionary.Add(fapDoubleList);
241                         }
242
243
244                     }
245                 }
246
247
248                 foreach (FAPDoubleList<Sheet> list in gridDictionary)
249                 {
250                     totalUsedFAPsetCount += list.CountFapSets();
```

```
251                 }
252             }
253 
254             sb.AppendLine("Workbook name: " + wb.WorkbookName);
255             sb.AppendLine("Total number of cells in sheets: " + totalCellCount);
256             sb.AppendLine("Total number of non-null cells in sheets: " + totalNonNullCellCount);
257             sb.AppendLine("FAPSet count: " + fapSetCounter);
258             sb.AppendLine("Actual FAP set count (reuse in effect): " + totalUsedFAPsetCount);
259             sb.AppendLine("FAPDoubleSet (FAPGrid) count: " + fapDoubleSetCounter);
260             sb.Append("Number of cells with supportgraph (FAPDoubleList): ").AppendLine(fapDoubleListCounter.ToString());
261             sb.AppendLine("FAPSets/FAPGrid: " + (double)fapSetCounter / fapDoubleSetCounter);
262             sb.AppendLine("FAPSets / Number of cells with supportgraph: " + (double)fapSetCounter / fapDoubleListCounter);
263             sb.AppendLine("Number of actual FAPSets / Number of actual cells with supportgraph: " + (double)totalUsedFAPsetCount ↙
    / (fapDoubleListCounter - totalReuseOfFAPDoubleList));
264             sb.AppendLine("Edges count: " + edgesCount);
265             sb.AppendLine("Real edgecount: " + realEdgeCount);
266             sb.AppendLine("FAPSets/Edges:" + (double)fapSetCounter / realEdgeCount);
267             sb.AppendLine("FAPSets (reuse in effect)/Edges:" + (double)totalUsedFAPsetCount / realEdgeCount);
268             sb.AppendLine("Highest number of FAPSets for a single Grid: ").AppendLine(fapSetsHighest.ToString());
269             sb.AppendLine("");
270             sb.AppendLine("Size of last topological sort: " + wb.sizeOfLastTopologicalSort);
271             sb.AppendLine("Number of cells containing volatile expressions: " + wb.volatileExpressions.Count);
272             sb.AppendLine("Number of cells waiting for a manual update: " + wb.waitingForManualUpdate.Count);
273             sb.Append("Number of times a FAPDoubleList have been reused: ").AppendLine(totalReuseOfFAPDoubleList.ToString());
274             return sb.ToString();
275         }
276 
277         public static string GetNumberOfDifferentCellTypes(Workbook wb)
278         {
279             int nullCells = 0;
280             int numberCells = 0;
281             int textCells = 0;
282             int formulaCells = 0;
283             int cellsWithError = 0;
284 
285             Type numberCellType = new NumberCell(0.0).GetType();
286             Type textCellType = new TextCell("").GetType();
287             Type formulaCellType = new Formula(new Workbook(), new NumberConst(0.0)).GetType();
288 
289 
290             foreach (Sheet sheet in wb)
291             {
292                 for (int c = 0; c < sheet.Cols; c++)
293                 {
294                     for (int r = 0; r < sheet.Rows; r++)
295                     {
296                         Cell cell = sheet[c, r];
297                         if (cell == null)
298                         {
299                             nullCells++;
300                         }
```

```
                        else
                        {
                            Type type = cell.GetType();
                            if (type.Equals(numberCellType))
                                numberCells++;
                            else if (type.Equals(textCellType))
                                textCells++;
                            else if (type.Equals(formulaCellType))
                                formulaCells++;
                            else
                                cellsWithError++;
                        }


                    }
                }
            }

            StringBuilder sb = new StringBuilder();
            sb.Append("Number of NumberCells:\t").AppendLine(numberCells.ToString());
            sb.Append("Number of TextCells:\t").AppendLine(textCells.ToString());
            sb.Append("Number of formula cells:\t").AppendLine(formulaCells.ToString());
            sb.Append("Number of Null cells:\t").AppendLine(nullCells.ToString());
            if (cellsWithError > 0)
                sb.Append("Cells with unknown type:\t").Append(cellsWithError.ToString());

            return sb.ToString();
        }


        private static void FullRecalc(Workbook wb, WorkbookForm gui)
        {
            long average = 0;
            gui.UseSupportGraph = false;
            for (int i = 0; i < numberOfTestsToPerform; i++)
            {
                average += wb.RecalculateFull().Ticks;
            }
            average = (average / numberOfTestsToPerform);
            Logger.Log.Fatal(THIS_TYPE, "Average of the " + numberOfTestsToPerform
                + " runs (seconds): " + new TimeSpan(average));
        }

        private static void AddRangeToRecalculation(string start, string end, Workbook wb, Sheet sheet)
        {
            FullCellAddr<Sheet> startCell = ConvertFormat(start, sheet);
            FullCellAddr<Sheet> endCell = ConvertFormat(end, sheet);
            for (int c = startCell.col; c < endCell.col; c++)
            {
                for (int r = startCell.row; r < endCell.row; r++)
                {
```

```
352                     wb.WaitingForManualUpdate.Add(new FullCellAddr<Sheet>(c, r, sheet));
353                 }
354             }
355         }
356 
357         private static void AddAllCellsToRecalculation(Workbook wb, WorkbookForm gui)
358         {
359             gui.UseSupportGraph = true;
360             gui.ManualRecalculation = true;
361             long average = 0;
362             for (int i = 0; i < numberOfTestsToPerform; i++)
363             {
364                 foreach (Sheet s in wb.sheets)
365                 {
366                     for (int c = 0; c < s.Cols; c++)
367                     {
368                         for (int r = 0; r < s.Rows; r++)
369                         {
370                             wb.WaitingForManualUpdate.Add(new FullCellAddr<Sheet>(c, r, s));
371                         }
372                     }
373                 }
374                 average += wb.Recalculate(true, true, true, true, wb.sheets[0], new CellAddr[] { }).Ticks;
375                 Log.Info(THIS_TYPE, average);
376             }
377             average = (average / numberOfTestsToPerform);
378             Logger.Log.Fatal(THIS_TYPE, "Average of the " + numberOfTestsToPerform
379                 + " runs (seconds): " + new TimeSpan(average));
380         }
381 
382         //TODO: quick, dirty and dodgy method :)
383         private static FullCellAddr<Sheet> ConvertFormat(string cellA1Name, Sheet sheet)
384         {
385             string alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
386             int col = 0;
387             int row = 0;
388             int temp = 0;
389             int numberCharModifier = 1;
390             int notNumberCharCounter = 0;
391             for(int i = cellA1Name.Length-1; i >= 0; i--)
392             {
393                 if (int.TryParse(cellA1Name[i].ToString(), out temp))
394                 {
395                     row += temp*numberCharModifier;
396                     temp = 0;
397                     numberCharModifier *= 10;
398                 }
399                 else {
400                     if(notNumberCharCounter != 0) col += ((alphabet.ToString().IndexOf(cellA1Name[i])+1)* notNumberCharCounter*  ↙
    alphabet.Length);
401                     else col += ((alphabet.ToString().IndexOf(cellA1Name[i])));
```

```
                    notNumberCharCounter++;

                }
            }
            row -= 1;
            return new FullCellAddr<Sheet>(col,row,sheet);
        }

        public static void ExecuteTest(Workbook wb, string choice, WorkbookForm gui)
        {
            long average = 0;
            switch(choice)
            {
                case "initFullgeneric":
                    InitFullgeneric(gui, wb);
                    break;
                case "initFewFINAN":
                    Log.Fatal(typeof(Test), "init FINAN minimal 1%");
                    gui.UseSupportGraph = true;
                    gui.ManualRecalculation = true;
                    average = 0;
                    for (int i = 0; i < numberOfTestsToPerform; i++)
                    {
                        wb.WaitingForManualUpdate.AddAll(new FullCellAddr<Sheet>[] {
                            ConvertFormat("D17", GetSheetByName("Naturgas", wb))
                        });
                        average += wb.Recalculate(true, false, true, true, wb.sheets[0], new CellAddr[] { }).Ticks;
                    }
                    average = (average / numberOfTestsToPerform);
                    Logger.Log.Fatal(THIS_TYPE, "Average of the " + numberOfTestsToPerform
                        + " runs (seconds): " + new TimeSpan(average));
                    break;
                case "initHalfFINAN":
                   Log.Fatal(typeof(Test), "init FINAN minimal 5%");
                    gui.UseSupportGraph = true;
                    gui.ManualRecalculation = true;
                    average = 0;
                    for (int i = 0; i < numberOfTestsToPerform; i++)
                    {
                        wb.WaitingForManualUpdate.AddAll(new FullCellAddr<Sheet>[] {
                            ConvertFormat("C13", GetSheetByName("Naturgas", wb))
                        });
                        average += wb.Recalculate(true, false, true, true, wb.sheets[0], new CellAddr[] { }).Ticks;
                    }
                    average = (average / numberOfTestsToPerform);
                    Logger.Log.Fatal(THIS_TYPE, "Average of the " + numberOfTestsToPerform
                        + " runs (seconds): " + new TimeSpan(average));
                    break;
                case "initAllFINAN":
                    Log.Fatal(typeof(Test), "init FINAN minimal 10%");
                    gui.UseSupportGraph = true;
```

```
                    gui.ManualRecalculation = true;
                    average = 0;
                    for (int i = 0; i < numberOfTestsToPerform; i++)
                    {
                        wb.WaitingForManualUpdate.AddAll(new FullCellAddr<Sheet>[] {
                            ConvertFormat("D17", GetSheetByName("Naturgas", wb)),
                            ConvertFormat("G9", GetSheetByName("NgasGasolieFuel", wb))
                        });
                        average += wb.Recalculate(true, false, true, true, wb.sheets[0], new CellAddr[] { }).Ticks;
                    }
                    average = (average / numberOfTestsToPerform);
                    Logger.Log.Fatal(THIS_TYPE, "Average of the " + numberOfTestsToPerform
                        + " runs (seconds): " + new TimeSpan(average));
                    break;
                case "initFewALPHA":
                    Log.Fatal(typeof(Test), "init ALPHA minimal 1%");
                    gui.UseSupportGraph = true;
                    gui.ManualRecalculation = true;
                    average = 0;
                    for (int i = 0; i < numberOfTestsToPerform; i++)
                    {
                        wb.WaitingForManualUpdate.AddAll(new FullCellAddr<Sheet>[] {
                            ConvertFormat("E197", GetSheetByName("ALPHA", wb))
                        });
                        average += wb.Recalculate(true, false, true, true, wb.sheets[0], new CellAddr[] { }).Ticks;
                    }
                    average = (average / numberOfTestsToPerform);
                    Logger.Log.Fatal(THIS_TYPE, "Average of the " + numberOfTestsToPerform
                        + " runs (seconds): " + new TimeSpan(average));
                    break;
                case "initHalfALPHA":
                    Log.Fatal(typeof(Test), "init ALPHA minimal 5%");
                    gui.UseSupportGraph = true;
                    gui.ManualRecalculation = true;
                    average = 0;
                    for (int i = 0; i < numberOfTestsToPerform; i++)
                    {
                        wb.WaitingForManualUpdate.AddAll(new FullCellAddr<Sheet>[] {
                            ConvertFormat("D187", GetSheetByName("ALPHA", wb))
                        });
                        average += wb.Recalculate(true, false, true, true, wb.sheets[0], new CellAddr[] { }).Ticks;
                    }
                    average = (average / numberOfTestsToPerform);
                    Logger.Log.Fatal(THIS_TYPE, "Average of the " + numberOfTestsToPerform
                        + " runs (seconds): " + new TimeSpan(average));
                    break;
                case "initAllALPHA":
                    Log.Fatal(typeof(Test), "init ALPHA minimal 10%");
                    gui.UseSupportGraph = true;
                    gui.ManualRecalculation = true;
                    average = 0;
```

```
                    for (int i = 0; i < numberOfTestsToPerform; i++)
                    {
                        wb.WaitingForManualUpdate.AddAll(new FullCellAddr<Sheet>[] {
                            ConvertFormat("C176", GetSheetByName("ALPHA", wb))
                        });
                        average += wb.Recalculate(true, false, true, true, wb.sheets[0], new CellAddr[] { }).Ticks;
                    }
                    average = (average / numberOfTestsToPerform);
                    Logger.Log.Fatal(THIS_TYPE, "Average of the " + numberOfTestsToPerform
                        + " runs (seconds): " + new TimeSpan(average));
                    break;
                case "initFewPROBA":
                    Log.Fatal(typeof(Test), "init PROBA minimal 1%");
                    gui.UseSupportGraph = true;
                    gui.ManualRecalculation = true;
                    average = 0;
                    for (int i = 0; i < numberOfTestsToPerform; i++)
                    {
                        wb.WaitingForManualUpdate.AddAll(new FullCellAddr<Sheet>[] {
                            ConvertFormat("D91", GetSheetByName("Stack", wb))
                        });
                        average += wb.Recalculate(true, false, true, true, wb.sheets[0], new CellAddr[] { }).Ticks;
                    }
                    average = (average / numberOfTestsToPerform);
                    Logger.Log.Fatal(THIS_TYPE, "Average of the " + numberOfTestsToPerform
                        + " runs (seconds): " + new TimeSpan(average));
                    break;
                case "initHalfPROBA":
                    Log.Fatal(typeof(Test), "init PROBA minimal 5%");
                    gui.UseSupportGraph = true;
                    gui.ManualRecalculation = true;
                    average = 0;
                    for (int i = 0; i < numberOfTestsToPerform; i++)
                    {
                        AddRangeToRecalculation("B2", "D202", wb, GetSheetByName("2-200", wb));
                        AddRangeToRecalculation("D4", "E70", wb, GetSheetByName("Stack", wb));
                        average += wb.Recalculate(true, false, true, true, wb.sheets[0], new CellAddr[] { }).Ticks;
                    }
                    average = (average / numberOfTestsToPerform);
                    Logger.Log.Fatal(THIS_TYPE, "Average of the " + numberOfTestsToPerform
                        + " runs (seconds): " + new TimeSpan(average));
                    break;
                case "initAllPROBA":
                    Log.Fatal(typeof(Test), "init PROBA minimal 10%");
                    gui.UseSupportGraph = true;
                    gui.ManualRecalculation = true;
                    average = 0;
                    for (int i = 0; i < numberOfTestsToPerform; i++)
                    {
                        AddRangeToRecalculation("B2", "D202", wb, GetSheetByName("2-200", wb));
                        AddRangeToRecalculation("B1", "F102", wb, GetSheetByName("1-100", wb));
```

```
                    AddRangeToRecalculation("B1", "B98", wb, GetSheetByName("5-100", wb));
                    AddRangeToRecalculation("B2", "E22", wb, GetSheetByName("1-20", wb));
                    AddRangeToRecalculation("B2", "F20", wb, GetSheetByName("3-18", wb));
                    AddRangeToRecalculation("D4", "E110", wb, GetSheetByName("Stack", wb));
                    AddRangeToRecalculation("C2", "I8", wb, GetSheetByName("1-6", wb));
                    average += wb.Recalculate(true, false, true, true, wb.sheets[0], new CellAddr[] { }).Ticks;
                }
                average = (average / numberOfTestsToPerform);
                Logger.Log.Fatal(THIS_TYPE, "Average of the " + numberOfTestsToPerform
                    + " runs (seconds): " + new TimeSpan(average));
                break;
            case "initFewROLE":
                Log.Fatal(typeof(Test), "init ROLE minimal 1%");
                gui.UseSupportGraph = true;
                gui.ManualRecalculation = true;
                average = 0;
                for (int i = 0; i < numberOfTestsToPerform; i++)
                {
                    wb.WaitingForManualUpdate.AddAll(new FullCellAddr<Sheet>[] {
                        ConvertFormat("B9", GetSheetByName("Combat", wb)),
                        ConvertFormat("C3", GetSheetByName("Tables", wb)),
                        ConvertFormat("P4", GetSheetByName("Primary", wb)),
                        ConvertFormat("L2", GetSheetByName("Combat", wb)),
                        ConvertFormat("C10", GetSheetByName("Advantages", wb))
                    });
                    AddRangeToRecalculation("C4", "C9", wb, GetSheetByName("Primary", wb));
                    AddRangeToRecalculation("Z20", "AJ29", wb, GetSheetByName("Combat", wb));
                    AddRangeToRecalculation("J3", "K45", wb, GetSheetByName("Advantages", wb));
                    AddRangeToRecalculation("B4", "D30", wb, GetSheetByName("Soft", wb));
                    average += wb.Recalculate(true, false, true, true, wb.sheets[0], new CellAddr[] { }).Ticks;
                }
                average = (average / numberOfTestsToPerform);
                Logger.Log.Fatal(THIS_TYPE, "Average of the " + numberOfTestsToPerform
                    + " runs (seconds): " + new TimeSpan(average));
                break;
            case "initHalfROLE":
                Log.Fatal(typeof(Test), "init ROLE minimal 5%");
                gui.UseSupportGraph = true;
                gui.ManualRecalculation = true;
                average = 0;
                for (int i = 0; i < numberOfTestsToPerform; i++)
                {
                    wb.WaitingForManualUpdate.AddAll(new FullCellAddr<Sheet>[] {
                        ConvertFormat("B9", GetSheetByName("Combat", wb)),
                        ConvertFormat("B460", GetSheetByName("Skills", wb))
                    });
                    average += wb.Recalculate(true, false, true, true, wb.sheets[0], new CellAddr[] { }).Ticks;
                }
                average = (average / numberOfTestsToPerform);
                Logger.Log.Fatal(THIS_TYPE, "Average of the " + numberOfTestsToPerform
                    + " runs (seconds): " + new TimeSpan(average));
```

```
                    break;
                case "initAllROLE":
                    Log.Fatal(typeof(Test), "init ROLE minimal 10%");
                    gui.UseSupportGraph = true;
                    gui.ManualRecalculation = true;
                    average = 0;
                    for (int i = 0; i < numberOfTestsToPerform; i++)
                    {
                        wb.WaitingForManualUpdate.AddAll(new FullCellAddr<Sheet>[] {
                            ConvertFormat("E6", GetSheetByName("Config", wb)),
                            ConvertFormat("C3", GetSheetByName("Tables", wb)),
                            ConvertFormat("P4", GetSheetByName("Primary", wb)),
                            ConvertFormat("L2", GetSheetByName("Combat", wb)),
                            ConvertFormat("C10", GetSheetByName("Advantages", wb))
                        });
                        AddRangeToRecalculation("B4", "D30", wb, GetSheetByName("Soft", wb));
                        AddRangeToRecalculation("J3", "K45", wb, GetSheetByName("Advantages", wb));
                        AddRangeToRecalculation("Z1", "AJ29", wb, GetSheetByName("Combat", wb));
                        AddRangeToRecalculation("C4", "C9", wb, GetSheetByName("Primary", wb));
                        average += wb.Recalculate(true, false, true, true, wb.sheets[0], new CellAddr[] { }).Ticks;
                    }
                    average = (average / numberOfTestsToPerform);
                    Logger.Log.Fatal(THIS_TYPE, "Average of the " + numberOfTestsToPerform
                        + " runs (seconds): " + new TimeSpan(average));
                    break;
                case "initFewPROT":
                    Log.Fatal(typeof(Test), "init PROT minimal 1%");
                    gui.UseSupportGraph = true;
                    gui.ManualRecalculation = true;
                    average = 0;
                    for (int i = 0; i < numberOfTestsToPerform; i++)
                    {
                        wb.WaitingForManualUpdate.AddAll(new FullCellAddr<Sheet>[] {
                            ConvertFormat("P24554", GetSheetByName("Sheet1", wb))
                        });
                        average += wb.Recalculate(true, false, true, true, wb.sheets[0], new CellAddr[] { }).Ticks;
                    }
                    average = (average / numberOfTestsToPerform);
                    Logger.Log.Fatal(THIS_TYPE, "Average of the " + numberOfTestsToPerform
                        + " runs (seconds): " + new TimeSpan(average));
                    break;
                case "initHalfPROT":
                    Log.Fatal(typeof(Test), "init PROT minimal 5%");
                    gui.UseSupportGraph = true;
                    gui.ManualRecalculation = true;
                    average = 0;
                    for (int i = 0; i < numberOfTestsToPerform; i++)
                    {
                        wb.WaitingForManualUpdate.AddAll(new FullCellAddr<Sheet>[] {
                            ConvertFormat("D26387", GetSheetByName("Sheet1", wb))
                        });
```

```
657                         average += wb.Recalculate(true, false, true, true, wb.sheets[0], new CellAddr[] { }).Ticks;
658                     }
659                     average = (average / numberOfTestsToPerform);
660                     Logger.Log.Fatal(THIS_TYPE, "Average of the " + numberOfTestsToPerform
661                         + " runs (seconds): " + new TimeSpan(average));
662                     break;
663                 case "initAllPROT":
664                     //TODO: can not define it since SupportCalc throws an stack overflow
665                     break;
666                 case "initAllgeneric":
667                     Log.Fatal(typeof(Test), "init minimal generic");
668                     AddAllCellsToRecalculation(wb, gui);
669                     break;
670                 case "markIIStructureSpeedRole":
671                     Log.Fatal(typeof(Test), "markIIStructureSpeedRole (no support graph)");
672                     {
673                         Logger.Log.Fatal(typeof (Test), "Test 1 (single, no support graph)\n");
674                         gui.UseSupportGraph = false;
675                         gui.RecalculateVolatileFunctions = false;
676                         gui.ManualRecalculation = true;
677                         InsertPrimitiveCommand insSingle = new InsertPrimitiveCommand(wb, "Skills", "L52", "L52", gui, "=$A$1");
678                         DeletePrimitiveCommand delSingle = new DeletePrimitiveCommand(wb, "Skills", "L52", "L52", gui);
679                         for (int i = 0; i < numberOfTestsToPerform; i++)
680                         {
681                             insSingle.Execute();
682                             insSingle.Undo();
683 
684                             delSingle.Execute();
685                             delSingle.Undo();
686                         }
687                         Log.Fatal(typeof(Test), "INSERT" + insSingle.GetMeasurements());
688                         Log.Fatal(typeof(Test), "DELETE" + delSingle.GetMeasurements());
689                     }
690 
691                     {
692                         Logger.Log.Fatal(typeof(Test), "Test 2 (multiple, no support graph, no area)\n");
693                         gui.UseSupportGraph = false;
694                         gui.RecalculateVolatileFunctions = false;
695                         gui.ManualRecalculation = true;
696                         InsertPrimitiveCommand insMultiple = new InsertPrimitiveCommand(wb, "Skills", "B10", "Q198", gui, "=$A$1"↙
    );
697                         DeletePrimitiveCommand delMultiple = new DeletePrimitiveCommand(wb, "Skills", "B10", "Q198", gui);
698                         for (int i = 0; i < numberOfTestsToPerform; i++)
699                         {
700                             insMultiple.Execute();
701                             insMultiple.Undo();
702 
703                             delMultiple.Execute();
704                             delMultiple.Undo();
705                         }
706                         Log.Fatal(typeof(Test), "INSERT" + insMultiple.GetMeasurements());
```

```
                    Log.Fatal(typeof(Test), "DELETE" + delMultiple.GetMeasurements());
                }

                {
                    Logger.Log.Fatal(typeof(Test), "Test 3 (multiple, no support graph, with area)\n");
                    gui.UseSupportGraph = false;
                    gui.RecalculateVolatileFunctions = false;
                    gui.ManualRecalculation = true;
                    InsertAreaCommand insMultipleArea = new InsertAreaCommand(wb, "Skills", "B10", "Q198", gui, "=$A$1");
                    DeleteAreaCommand delMultipleArea = new DeleteAreaCommand(wb, "Skills", "B10", "Q198", gui);
                    for (int i = 0; i < numberOfTestsToPerform; i++)
                    {
                        insMultipleArea.Execute();
                        insMultipleArea.Undo();

                        delMultipleArea.Execute();
                        delMultipleArea.Undo();
                    }
                    Log.Fatal(typeof(Test), "INSERT" + insMultipleArea.GetMeasurements());
                    Log.Fatal(typeof(Test), "DELETE" + delMultipleArea.GetMeasurements());
                }

                Log.Fatal(typeof(Test), "markIIStructureSpeedRole (with support graph)");
                {
                    Logger.Log.Fatal(typeof(Test), "Test 1 (single, with support graph)\n");
                    gui.UseSupportGraph = true;
                    gui.RecalculateVolatileFunctions = false;
                    gui.ManualRecalculation = true;
                    InsertPrimitiveCommand insSingle = new InsertPrimitiveCommand(wb, "Skills", "L52", "L52", gui, "=$A$1");
                    DeletePrimitiveCommand delSingle = new DeletePrimitiveCommand(wb, "Skills", "L52", "L52", gui);
                    for (int i = 0; i < numberOfTestsToPerform; i++)
                    {
                        insSingle.Execute();
                        insSingle.Undo();

                        delSingle.Execute();
                        delSingle.Undo();
                    }
                    Log.Fatal(typeof(Test), "INSERT" + insSingle.GetMeasurements());
                    Log.Fatal(typeof(Test), "DELETE" + delSingle.GetMeasurements());
                }
                {
                    Logger.Log.Fatal(typeof(Test), "Test 2 (multiple, with support graph, no area)\n");
                    gui.UseSupportGraph = true;
                    gui.RecalculateVolatileFunctions = false;
                    gui.ManualRecalculation = true;
                    InsertPrimitiveCommand insMultiple = new InsertPrimitiveCommand(wb, "Skills", "B10", "Q198", gui, "=$A$1"↙
);
                    DeletePrimitiveCommand delMultiple = new DeletePrimitiveCommand(wb, "Skills", "B10", "Q198", gui);
                    for (int i = 0; i < numberOfTestsToPerform; i++)
                    {
```

```
757                             insMultiple.Execute();
758                             insMultiple.Undo();
759 
760                             delMultiple.Execute();
761                             delMultiple.Undo();
762                         }
763                         Log.Fatal(typeof(Test), "INSERT" + insMultiple.GetMeasurements());
764                         Log.Fatal(typeof(Test), "DELETE" + delMultiple.GetMeasurements());
765                     }
766                     {
767                         Logger.Log.Fatal(typeof(Test), "Test 3 (multiple, with support graph, with area)\n");
768                         gui.UseSupportGraph = true;
769                         gui.RecalculateVolatileFunctions = false;
770                         gui.ManualRecalculation = true;
771                         InsertAreaCommand insMultipleArea = new InsertAreaCommand(wb, "Skills", "B10", "Q198", gui, "=$A$1");
772                         DeleteAreaCommand delMultipleArea = new DeleteAreaCommand(wb, "Skills", "B10", "Q198", gui);
773                         for (int i = 0; i < numberOfTestsToPerform; i++)
774                         {
775                             insMultipleArea.Execute();
776                             insMultipleArea.Undo();
777 
778                             delMultipleArea.Execute();
779                             delMultipleArea.Undo();
780                         }
781                         Log.Fatal(typeof(Test), "INSERT" + insMultipleArea.GetMeasurements());
782                         Log.Fatal(typeof(Test), "DELETE" + delMultipleArea.GetMeasurements());
783                     }
784                     break;
785                 case "markIIStructureSpeedProt":
786                     Log.Fatal(typeof(Test), "markIIStructureSpeedProt (no support graph)");
787                     {
788                         Logger.Log.Fatal(typeof (Test), "Test 1 (single, no support graph)\n");
789                         gui.UseSupportGraph = false;
790                         gui.RecalculateVolatileFunctions = false;
791                         gui.ManualRecalculation = true;
792                         InsertPrimitiveCommand insSingle = new InsertPrimitiveCommand(wb, "Sheet1", "M65", "M65", gui, "=$A$1");
793                         DeletePrimitiveCommand delSingle = new DeletePrimitiveCommand(wb, "Sheet1", "M65", "M65", gui);
794                         for (int i = 0; i < numberOfTestsToPerform; i++)
795                         {
796                             insSingle.Execute();
797                             insSingle.Undo();
798 
799                             delSingle.Execute();
800                             delSingle.Undo();
801                         }
802                         Log.Fatal(typeof(Test), "INSERT" + insSingle.GetMeasurements());
803                         Log.Fatal(typeof(Test), "DELETE" + delSingle.GetMeasurements());
804                     }
805 
806                     {
807                         Logger.Log.Fatal(typeof(Test), "Test 2 (multiple, no support graph, no area)\n");
```

```
808                         gui.UseSupportGraph = false;
809                         gui.RecalculateVolatileFunctions = false;
810                         gui.ManualRecalculation = true;
811                         InsertPrimitiveCommand insMultiple = new InsertPrimitiveCommand(wb, "Sheet1", "C10", "CP75", gui, "=$A$1"↙
    );
812                         DeletePrimitiveCommand delMultiple = new DeletePrimitiveCommand(wb, "Sheet1", "C10", "CP75", gui);
813                         for (int i = 0; i < numberOfTestsToPerform; i++)
814                         {
815                             insMultiple.Execute();
816                             insMultiple.Undo();
817 
818                             delMultiple.Execute();
819                             delMultiple.Undo();
820                         }
821                         Log.Fatal(typeof(Test), "INSERT" + insMultiple.GetMeasurements());
822                         Log.Fatal(typeof(Test), "DELETE" + delMultiple.GetMeasurements());
823                     }
824 
825                     {
826                         Logger.Log.Fatal(typeof(Test), "Test 3 (multiple, no support graph, with area)\n");
827                         gui.UseSupportGraph = false;
828                         gui.RecalculateVolatileFunctions = false;
829                         gui.ManualRecalculation = true;
830                         InsertAreaCommand insMultipleArea = new InsertAreaCommand(wb, "Sheet1", "C10", "CP75", gui, "=$A$1");
831                         DeleteAreaCommand delMultipleArea = new DeleteAreaCommand(wb, "Sheet1", "C10", "CP75", gui);
832                         for (int i = 0; i < numberOfTestsToPerform; i++)
833                         {
834                             insMultipleArea.Execute();
835                             insMultipleArea.Undo();
836 
837                             delMultipleArea.Execute();
838                             delMultipleArea.Undo();
839                         }
840                         Log.Fatal(typeof(Test), "INSERT" + insMultipleArea.GetMeasurements());
841                         Log.Fatal(typeof(Test), "DELETE" + delMultipleArea.GetMeasurements());
842                     }
843 
844                     Log.Fatal(typeof(Test), "markIIStructureSpeedProt (with support graph)");
845                     {
846                         Logger.Log.Fatal(typeof(Test), "Test 1 (single, with support graph)\n");
847                         gui.UseSupportGraph = true;
848                         gui.RecalculateVolatileFunctions = false;
849                         gui.ManualRecalculation = true;
850                         InsertPrimitiveCommand insSingle = new InsertPrimitiveCommand(wb, "Sheet1", "M65", "M65", gui, "=$A$1");
851                         DeletePrimitiveCommand delSingle = new DeletePrimitiveCommand(wb, "Sheet1", "M65", "M65", gui);
852                         for (int i = 0; i < numberOfTestsToPerform; i++)
853                         {
854                             insSingle.Execute();
855                             insSingle.Undo();
856 
857                             delSingle.Execute();
```

```
858                             delSingle.Undo();
859                         }
860                         Log.Fatal(typeof(Test), "INSERT" + insSingle.GetMeasurements());
861                         Log.Fatal(typeof(Test), "DELETE" + delSingle.GetMeasurements());
862                     }
863                     {
864                         Logger.Log.Fatal(typeof(Test), "Test 2 (multiple, with support graph, no area)\n");
865                         gui.UseSupportGraph = true;
866                         gui.RecalculateVolatileFunctions = false;
867                         gui.ManualRecalculation = true;
868                         InsertPrimitiveCommand insMultiple = new InsertPrimitiveCommand(wb, "Sheet1", "C10", "CP75", gui, "=$A$1"↙
    );
869                         DeletePrimitiveCommand delMultiple = new DeletePrimitiveCommand(wb, "Sheet1", "C10", "CP75", gui);
870                         for (int i = 0; i < numberOfTestsToPerform; i++)
871                         {
872                             insMultiple.Execute();
873                             insMultiple.Undo();
874 
875                             delMultiple.Execute();
876                             delMultiple.Undo();
877                         }
878                         Log.Fatal(typeof(Test), "INSERT" + insMultiple.GetMeasurements());
879                         Log.Fatal(typeof(Test), "DELETE" + delMultiple.GetMeasurements());
880                     }
881                     {
882                         Logger.Log.Fatal(typeof(Test), "Test 3 (multiple, with support graph, with area)\n");
883                         gui.UseSupportGraph = true;
884                         gui.RecalculateVolatileFunctions = false;
885                         gui.ManualRecalculation = true;
886                         InsertAreaCommand insMultipleArea = new InsertAreaCommand(wb, "Sheet1", "C10", "CP75", gui, "=$A$1");
887                         DeleteAreaCommand delMultipleArea = new DeleteAreaCommand(wb, "Sheet1", "C10", "CP75", gui);
888                         for (int i = 0; i < numberOfTestsToPerform; i++)
889                         {
890                             insMultipleArea.Execute();
891                             insMultipleArea.Undo();
892 
893                             delMultipleArea.Execute();
894                             delMultipleArea.Undo();
895                         }
896                         Log.Fatal(typeof(Test), "INSERT" + insMultipleArea.GetMeasurements());
897                         Log.Fatal(typeof(Test), "DELETE" + delMultipleArea.GetMeasurements());
898                     }
899                     break;
900                 case "markIIDegradationROLE":
901                     {
902                         Logger.Log.Fatal(typeof(Test), "markIIDegradationROLE using area functions\n");
903                         gui.UseSupportGraph = true;
904                         gui.RecalculateVolatileFunctions = false;
905                         gui.ManualRecalculation = true;
906                         Log.Fatal(typeof(Test), "BEFORE:\n" + PrintMemoryRelatedInfo(wb));
907                         DeleteAreaCommand op1 = new DeleteAreaCommand(wb, "Skills", "E100", "H500", gui);
```

```
908                     InsertAreaCommand op2 = new InsertAreaCommand(wb, "Skills", "C90", "J200", gui, "=SUM(A1:C5)");
909                     DeleteAreaCommand op3 = new DeleteAreaCommand(wb, "Skills", "G120", "I300", gui);
910                     InsertAreaCommand op4 = new InsertAreaCommand(wb, "Skills", "D20", "F180", gui, "=$C5");
911
912                     //invoke changes
913                     Log.Fatal(typeof(Test), "op1 execute");
914                     op1.Execute();
915                     Log.Fatal(typeof(Test), "op2 execute");
916                     op2.Execute();
917                     Log.Fatal(typeof(Test), "op3 execute");
918                     op3.Execute();
919                     Log.Fatal(typeof(Test), "op4 execute");
920                     op4.Execute();
921
922                     //undo changes in order
923                     Log.Fatal(typeof(Test), "op4 undo");
924                     op4.Undo();
925                     Log.Fatal(typeof(Test), "op3 undo");
926                     op3.Undo();
927                     Log.Fatal(typeof(Test), "op2 undo");
928                     op2.Undo();
929                     Log.Fatal(typeof(Test), "op1 undo");
930                     op1.Undo();
931
932                     Log.Fatal(typeof(Test), "AFTER:\n" + PrintMemoryRelatedInfo(wb));
933                 }
934                 break;
935             case "markIIDegradationPROT":
936                 {
937                     Logger.Log.Fatal(typeof(Test), "markIIDegradationPROT using area functions\n");
938                     gui.UseSupportGraph = true;
939                     gui.RecalculateVolatileFunctions = false;
940                     gui.ManualRecalculation = true;
941                     Log.Fatal(typeof(Test), "BEFORE:\n" + PrintMemoryRelatedInfo(wb));
942                     DeleteAreaCommand op1 = new DeleteAreaCommand(wb, "Sheet1", "E100", "H500", gui);
943                     InsertAreaCommand op2 = new InsertAreaCommand(wb, "Sheet1", "C90", "J200", gui, "=SUM(A1:C5)");
944                     DeleteAreaCommand op3 = new DeleteAreaCommand(wb, "Sheet1", "G120", "I300", gui);
945                     InsertAreaCommand op4 = new InsertAreaCommand(wb, "Sheet1", "D20", "F180", gui, "=$C5");
946
947                     //invoke changes
948                     Log.Fatal(typeof(Test), "op1 execute");
949                     op1.Execute();
950                     Log.Fatal(typeof(Test), "op2 execute");
951                     op2.Execute();
952                     Log.Fatal(typeof(Test), "op3 execute");
953                     op3.Execute();
954                     Log.Fatal(typeof(Test), "op4 execute");
955                     op4.Execute();
956
957                     //undo changes in order
958                     Log.Fatal(typeof(Test), "op4 undo");
```

```
                    op4.Undo();
                    Log.Fatal(typeof(Test), "op3 undo");
                    op3.Undo();
                    Log.Fatal(typeof(Test), "op2 undo");
                    op2.Undo();
                    Log.Fatal(typeof(Test), "op1 undo");
                    op1.Undo();

                    Log.Fatal(typeof(Test), "AFTER:\n" + PrintMemoryRelatedInfo(wb));
                }
                break;
            case "markIIISpeedRole":
                Console.WriteLine("REMEMBER TO BUILD SUPPORT GRAPH AND USE MARK III TOPOLOGICAL SETTING");
                Console.ReadLine();
                Log.Fatal(typeof(Test), "markIIISpeedRole");
                Log.Fatal(typeof(Test), "---Mark III full role---\n");
                ExecuteTest(wb, "initFullgeneric", gui);
                Log.Fatal(typeof(Test), "---Mark III all role---\n");
                ExecuteTest(wb, "initAllgeneric", gui);
                Log.Fatal(typeof(Test), "---Mark III 1% role---\n");
                ExecuteTest(wb, "initFewROLE", gui);
                Log.Fatal(typeof(Test), "---Mark III 5% role---\n");
                ExecuteTest(wb, "initHalfROLE", gui);
                Log.Fatal(typeof(Test), "---Mark III 10% role---\n");
                ExecuteTest(wb, "initAllROLE", gui);
                break;
            case "markIIISpeedProt":
                Console.WriteLine("REMEMBER TO BUILD SUPPORT GRAPH AND USE MARK III TOPOLOGICAL SETTING");
                Console.ReadLine();
                Log.Fatal(typeof(Test), "markIIISpeedProt");
                //Log.Fatal(typeof(Test), "---Mark III full prot---\n");
                //ExecuteTest(wb, "initFullgeneric", gui);
                //Log.Fatal(typeof(Test), "---Mark III all prot---\n");
                //ExecuteTest(wb, "initAllgeneric", gui);
                Log.Fatal(typeof(Test), "---Mark III 1% prot---\n");
                ExecuteTest(wb, "initFewPROT", gui);
                Log.Fatal(typeof(Test), "---Mark III 5% prot---\n");
                ExecuteTest(wb, "initHalfPROT", gui);
                Log.Fatal(typeof(Test), "---Mark III 10% prot---\n");
                ExecuteTest(wb, "initAllPROT", gui);
                break;
            case "markISpeedCompleteFinan":
                Log.Fatal(typeof(Test), "markISpeedCompleteFinan");
                Log.Fatal(typeof(Test), "---Mark I full prot---\n");
                ExecuteTest(wb, "initFullgeneric", gui);
                Log.Fatal(typeof(Test), "---Mark I all prot---\n");
                ExecuteTest(wb, "initAllgeneric", gui);
                Log.Fatal(typeof(Test), "---Mark I 1% prot---\n");
                ExecuteTest(wb, "initFewFINAN", gui);
                Log.Fatal(typeof(Test), "---Mark I 5% prot---\n");
                ExecuteTest(wb, "initHalfFINAN", gui);
```

```
1010                    Log.Fatal(typeof(Test), "---Mark I 10% prot---\n");
1011                    ExecuteTest(wb, "initAllFINAN", gui);
1012                    break;
1013                case "test":
1014                    {
1015                        gui.UseSupportGraph = true;
1016                        gui.RecalculateVolatileFunctions = false;
1017                        gui.ManualRecalculation = true;
1018                        Log.Fatal(typeof(Test), "BEFORE:\n" + PrintMemoryRelatedInfo(wb));
1019                        DeleteAreaCommand op1 = new DeleteAreaCommand(wb, "Sheet1", "A3", "C5", gui);
1020                        InsertAreaCommand op2 = new InsertAreaCommand(wb, "Sheet1", "B2", "D9", gui, "=SUM(A1:C5)");
1021                        DeleteAreaCommand op3 = new DeleteAreaCommand(wb, "Sheet1", "C1", "D6", gui);
1022                        InsertAreaCommand op4 = new InsertAreaCommand(wb, "Sheet1", "B1", "D3", gui, "=$C5");
1023
1024                        //invoke changes
1025                        Log.Fatal(typeof(Test), "op1 execute");
1026                        //op1.Execute();
1027                        Log.Fatal(typeof(Test), "op2 execute");
1028                        op2.Execute();
1029                        Log.Fatal(typeof(Test), "op3 execute");
1030                        //op3.Execute();
1031                        Log.Fatal(typeof(Test), "op4 execute");
1032                        //op4.Execute();
1033
1034                        //undo changes in order
1035                        Log.Fatal(typeof(Test), "op4 undo");
1036                        //op4.Undo();
1037                        Log.Fatal(typeof(Test), "op3 undo");
1038                        //op3.Undo();
1039                        Log.Fatal(typeof(Test), "op2 undo");
1040                        op2.Undo();
1041                        Log.Fatal(typeof(Test), "op1 undo");
1042                        //op1.Undo();
1043
1044                        Log.Fatal(typeof(Test), "AFTER:\n" + PrintMemoryRelatedInfo(wb));
1045                    }
1046                    break;
1047                default:
1048                    Log.Info(typeof(Test), "Test no found");
1049                    break;
1050            }
1051        }
1052
1053        private static void InitFullgeneric(WorkbookForm gui, Workbook wb)
1054        {
1055            Log.Fatal(typeof(Test), "init generic full");
1056            FullRecalc(wb, gui);
1057        }
1058    }
1059
1060    class DeleteAreaCommand : Command
```

```
1061    {
1062        public DeleteAreaCommand(Workbook wb, string sheet, string ul, string lr, WorkbookForm gui)
1063        {
1064            this.wb = wb;
1065            this.sheet = sheet;
1066            this.ul = ul;
1067            this.lr = lr;
1068            this.gui = gui;
1069        }
1070
1071        internal override long ExecuteCommandSpecificCode()
1072        {
1073            gui.ManualRecalculation = true;
1074            FullCellAddr<Sheet> ulCell = GetCellFromA1Format(ul, sheet);
1075            FullCellAddr<Sheet> lrCell = GetCellFromA1Format(lr, sheet);
1076            Sheet sh = wb[sheet];
1077            if (!gui.UseSupportGraph)
1078            {
1079                Stopwatch stopwatch = new Stopwatch();
1080                for (int c = ulCell.col; c <= lrCell.col; c++)
1081                {
1082                    for (int r = ulCell.row; r <= lrCell.row; r++)
1083                    {
1084                        CellAddr ca = new CellAddr(c, r);
1085                        stopwatch.Start();
1086                        sh.DeleteCell(ca);
1087                        stopwatch.Stop();
1088                    }
1089                }
1090                return stopwatch.Elapsed.Ticks;
1091            }
1092            else
1093            {
1094                Stopwatch stopwatch = new Stopwatch();
1095                stopwatch.Start();
1096                sh.SupportGraph.DeleteAreaFromGraph(ulCell.col, lrCell.col, ulCell.row, lrCell.row);
1097                stopwatch.Stop();
1098                for (int c = ulCell.col; c <= lrCell.col; c++)
1099                {
1100                    for (int r = ulCell.row; r <= lrCell.row; r++)
1101                    {
1102                        sh.DeleteCell(new CellAddr(c, r));
1103                    }
1104                }
1105                return stopwatch.Elapsed.Ticks;
1106            }
1107        }
1108    }
1109
1110    class InsertAreaCommand : Command
1111    {
```

```
1112        public InsertAreaCommand(Workbook wb, string sheet, string ul, string lr, WorkbookForm gui, string formula)
1113        {
1114            this.wb = wb;
1115            this.sheet = sheet;
1116            this.ul = ul;
1117            this.lr = lr;
1118            this.gui = gui;
1119            this.formula = formula;
1120        }
1121
1122        internal override long ExecuteCommandSpecificCode()
1123        {
1124            gui.ManualRecalculation = true;
1125            FullCellAddr<Sheet> ulCell = GetCellFromA1Format(ul, sheet);
1126            FullCellAddr<Sheet> lrCell = GetCellFromA1Format(lr, sheet);
1127            Cell newCell = Cell.Parse(formula, wb, ulCell.col, ulCell.row);
1128            Sheet sh = wb[sheet];
1129            if (!gui.UseSupportGraph)
1130            {
1131                Stopwatch stopwatch = new Stopwatch();
1132                CellAddr ulC = new CellAddr(ulCell.col, ulCell.row);
1133                int cols = (lrCell.col - ulCell.col + 1);
1134                int rows = (lrCell.row - ulCell.row + 1);
1135                stopwatch.Start();
1136                sh.PasteCell(newCell, ulC, cols, rows);
1137                stopwatch.Stop();
1138                return stopwatch.Elapsed.Ticks;
1139            }
1140            else
1141            {
1142                Stopwatch stopwatch = new Stopwatch();
1143                Formula f = newCell as Formula;
1144                if(f != null)
1145                {
1146                    CellAddr ulC = new CellAddr(ulCell.col, ulCell.row);
1147                    int cols = (lrCell.col - ulCell.col + 1);
1148                    int rows = (lrCell.row - ulCell.row + 1);
1149                    stopwatch.Start();
1150                    sh.SupportGraph.DeleteAreaFromGraph(ulCell.col, lrCell.col, ulCell.row, lrCell.row);
1151                    stopwatch.Stop();
1152                    for (int c = ulCell.col; c <= lrCell.col; c++)
1153                    {
1154                        for (int r = ulCell.row; r <= lrCell.row; r++)
1155                        {
1156                            sh.DeleteCell(new CellAddr(c, r));
1157                        }
1158                    }
1159                    stopwatch.Start();
1160                    sh.SupportGraph.CopyToCells(f.E, ulC, cols, rows);
1161                    stopwatch.Stop();
1162                    sh.PasteCell(newCell, ulC, cols, rows);
```

```
1163                }
1164
1165                return stopwatch.Elapsed.Ticks;
1166            }
1167        }
1168    }
1169
1170    class InsertPrimitiveCommand : Command
1171    {
1172        public InsertPrimitiveCommand(Workbook wb, string sheet, string ul, string lr, WorkbookForm gui, string formula)
1173        {
1174            this.wb = wb;
1175            this.sheet = sheet;
1176            this.ul = ul;
1177            this.lr = lr;
1178            this.gui = gui;
1179            this.formula = formula;
1180        }
1181
1182        internal override long ExecuteCommandSpecificCode()
1183        {
1184            gui.ManualRecalculation = true;
1185            FullCellAddr<Sheet> ulCell = GetCellFromA1Format(ul, sheet);
1186            FullCellAddr<Sheet> lrCell = GetCellFromA1Format(lr, sheet);
1187            Sheet sh = wb[sheet];
1188            if (!gui.UseSupportGraph)
1189            {
1190                Stopwatch stopwatch = new Stopwatch();
1191                for (int c = ulCell.col; c <= lrCell.col; c++)
1192                {
1193                    for (int r = ulCell.row; r <= lrCell.row; r++)
1194                    {
1195                        CellAddr ca = new CellAddr(c, r);
1196                        stopwatch.Start();
1197                        sh.InsertCell(formula, ca);
1198                        stopwatch.Stop();
1199                    }
1200                }
1201                return stopwatch.Elapsed.Ticks;
1202            }
1203            else
1204            {
1205                Stopwatch stopwatch = new Stopwatch();
1206                for (int c = ulCell.col; c <= lrCell.col; c++)
1207                {
1208                    for (int r = ulCell.row; r <= lrCell.row; r++)
1209                    {
1210                        Cell newCell = Cell.Parse(formula, wb, c, r);
1211                        CellAddr ca = new CellAddr(c,r);
1212                        Cell oldCell = sh[c, r];
1213                        stopwatch.Start();
```

```
1214                        sh.SupportGraph.InsertCellToGraph(oldCell, newCell, ca);
1215                        stopwatch.Stop();
1216                        sh.InsertCell(formula, ca);
1217                    }
1218                }
1219                return stopwatch.Elapsed.Ticks;
1220            }
1221        }
1222    }
1223
1224    class DeletePrimitiveCommand:Command
1225    {
1226        public DeletePrimitiveCommand(Workbook wb, string sheet, string ul, string lr, WorkbookForm gui)
1227        {
1228            this.wb = wb;
1229            this.sheet = sheet;
1230            this.ul = ul;
1231            this.lr = lr;
1232            this.gui = gui;
1233        }
1234
1235        internal override long ExecuteCommandSpecificCode()
1236        {
1237            gui.ManualRecalculation = true;
1238            FullCellAddr<Sheet> ulCell = GetCellFromA1Format(ul, sheet);
1239            FullCellAddr<Sheet> lrCell = GetCellFromA1Format(lr, sheet);
1240            Sheet sh = wb[sheet];
1241
1242            if (!gui.UseSupportGraph)
1243            {
1244                Stopwatch stopwatch = new Stopwatch();
1245                for (int c = ulCell.col; c <= lrCell.col; c++)
1246                {
1247                    for (int r = ulCell.row; r <= lrCell.row; r++)
1248                    {
1249                        CellAddr ca = new CellAddr(c, r);
1250                        stopwatch.Start();
1251                        sh.DeleteCell(ca);
1252                        stopwatch.Stop();
1253                    }
1254                }
1255                return stopwatch.Elapsed.Ticks;
1256            }else
1257            {
1258                Stopwatch stopwatch = new Stopwatch();
1259                for (int c = ulCell.col; c <= lrCell.col; c++)
1260                {
1261                    for (int r = ulCell.row; r <= lrCell.row; r++)
1262                    {
1263                        Cell oldCell = sh[c, r];
1264                        stopwatch.Start();
```

```
                        sh.SupportGraph.DeleteCellFromGraph(oldCell, c, r);
                        stopwatch.Stop();
                        sh.DeleteCell(new CellAddr(c, r));
                    }
                }
                return stopwatch.Elapsed.Ticks;
            }
        }
    }

    //TODO: does not like volatile functions (extreme swapping occures, did not check why)
    abstract class Command
    {
        protected Workbook wb;
        protected string sheet;
        protected LinkedList<Cell> undoState = new LinkedList<Cell>();
        private LinkedList<long> measurements = new LinkedList<long>();
        protected string ul;
        protected string lr;
        protected WorkbookForm gui;
        private int executesPerformed = 0;
        protected string formula;

        internal abstract long ExecuteCommandSpecificCode();

        public void ResetResults()
        {
            measurements = new LinkedList<long>();
            executesPerformed = 0;
        }

        public void Execute()
        {
            if (ul != null && lr != null && sheet != null && wb != null && gui != null)
            {
                SaveState(ul, lr, sheet, wb);
            }
            else
            {
                Log.Error(this, "Failed to save initial state");
            }
            measurements.Add(ExecuteCommandSpecificCode());
            executesPerformed++;
        }

        public void Undo()
        {
            if (undoState.Count > 0 && ul != null && lr != null && sheet != null && wb != null && gui != null)
            {
                RestoreState(ul, lr, sheet, wb);
            }
```

```
1316            else
1317            {
1318                Log.Error(this, "Failed to undo state changes");
1319            }
1320        }
1321
1322        public string GetMeasurements()
1323        {
1324            StringBuilder sb = new StringBuilder();
1325            long average = 0;
1326            foreach (long l in measurements)
1327            {
1328                average += l;
1329                sb.Append("\nRun took: ").Append(new TimeSpan(l));
1330            }
1331            average = average/executesPerformed;
1332            sb.Append("\nAverage of the ").Append(executesPerformed).Append(" executes: ").Append(
1333                new TimeSpan(average)).AppendLine();
1334            return sb.ToString();
1335        }
1336
1337        private void SaveState(string ul, string lr, string sheetname, Workbook wb)
1338        {
1339            Formats fo = new Formats();
1340            fo.RefFmt = Formats.RefType.A1;
1341            FullCellAddr<Sheet> ulCell = GetCellFromA1Format(ul, sheetname);
1342            FullCellAddr<Sheet> lrCell = GetCellFromA1Format(lr, sheetname);
1343            Sheet sh = wb[sheet];
1344            for (int c = ulCell.col; c <= lrCell.col; c++)
1345            {
1346                for (int r = ulCell.row; r <= lrCell.row; r++)
1347                {
1348                    undoState.InsertLast(sh[c, r]);
1349                }
1350            }
1351        }
1352
1353        private void RestoreState(string ul, string lr, string sheetname, Workbook wb)
1354        {
1355            Formats fo = new Formats();
1356            fo.RefFmt = Formats.RefType.A1;
1357            FullCellAddr<Sheet> ulCell = GetCellFromA1Format(ul, sheetname);
1358            FullCellAddr<Sheet> lrCell = GetCellFromA1Format(lr, sheetname);
1359            Sheet sh = wb[sheet];
1360            int index = 0;
1361            for (int c = ulCell.col; c <= lrCell.col; c++)
1362            {
1363                for (int r = ulCell.row; r <= lrCell.row; r++)
1364                {
1365                    if (undoState[index] != null)
1366                        Log.Info(this, "Inserting cell " + undoState[index].Show(c, r, fo) + " into cell " + CellAddr.ColumnName ↙
```

```
        (c) + (r+1));
1367                    else Log.Info(this, "Inserting cell null into cell " + CellAddr.ColumnName(c) + (r + 1));
1368                    sh.SupportGraph.DeleteCellFromGraph(sh[c, r], new CellAddr(c, r));
1369                    sh[c, r] = null;
1370                    sh.SupportGraph.InsertCellToGraph(sh[c, r], undoState[index], new CellAddr(c, r));
1371                    sh[c, r] = undoState[index];
1372                    index++;
1373                }
1374            }
1375            undoState.Clear();
1376        }
1377
1378        protected FullCellAddr<Sheet> GetCellFromA1Format(string a1Address, string sheet)
1379        {
1380            string alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
1381            int col = 0;
1382            int row = 0;
1383            int temp = 0;
1384            int numberCharModifier = 1;
1385            int notNumberCharCounter = 0;
1386            for (int i = a1Address.Length - 1; i >= 0; i--)
1387            {
1388                if (int.TryParse(a1Address[i].ToString(), out temp))
1389                {
1390                    row += temp*numberCharModifier;
1391                    temp = 0;
1392                    numberCharModifier *= 10;
1393                }
1394                else {
1395                    if (notNumberCharCounter != 0) col += ((alphabet.ToString().IndexOf(a1Address[i]) + 1) * notNumberCharCounter↙
        * alphabet.Length);
1396                    else col += ((alphabet.ToString().IndexOf(a1Address[i])));
1397                    notNumberCharCounter++;
1398
1399                }
1400            }
1401            row -= 1;
1402            return new FullCellAddr<Sheet>(col,row, wb[sheet]);
1403        }
1404    }
1405 }
```

CoreCalc.SupportGraph

TopologicalSort

```
/// This is based on the topological sort as described in the C5 generic libary
/// by Niels Kokholm and Peter Sestoft (TR-2006-76) see http://www.itu.dk/research/c5/
/// The code is the same as decribed on page 198 of the report with some slight modifications

using System.Collections.Generic;
using C5;
using FAP;

namespace CoreCalc.SupportGraf
{
    /// <summary>
    /// Used to make a topological linked list out of a no-cycle-graph
    /// </summary>
    public class TopologicalSort
    {
        /// <summary>
        /// Method to get a list of children out of a node
        /// </summary>
        /// <typeparam name="I">The type of the node</typeparam>
        /// <typeparam name="O">The type of the list of nodes. Must obey the IEnumerable<I> interface</typeparam>
        /// <param name="node">The node to find children of</param>
        /// <returns>A list of children</returns>
        public delegate O GetChildren<I, O>(I node) where O:IEnumerable<I>;

        /// <summary>
        /// Topologically sorts a tree
        /// </summary>
        /// <typeparam name="I">The type of a node</typeparam>
        /// <typeparam name="O">The type of a list or collection of children of a node</typeparam>
        /// <param name="getChildrenMethod">Method to extract children out of a node</param>
        /// <param name="starts">The nodes to start at</param>
        /// <returns>A topologically sorted list</returns>
        public static C5.IList<I> Toposort<I, O>(GetChildren<I,O> getChildrenMethod, params I[] starts) where O:IEnumerable<I>
        {
            HashedLinkedList<I> sorted = new HashedLinkedList<I>();
            foreach (I start in starts)
                if (!sorted.Contains(start))
                    AddNode(getChildrenMethod, sorted, start);
            sorted.Reverse();
            return sorted;
        }

        private static void AddNode<I, O>(GetChildren<I, O> getChildrenMethod, C5.IList<I> sorted, I node) where O : IEnumerable
    <I>
        {
            foreach (I child in getChildrenMethod(node))
                if (!sorted.Contains(child))
                    AddNode(getChildrenMethod, sorted, child);
            sorted.InsertLast(node);
        }

```

```
51
52          /// <summary>
53          /// Topologically sorts a tree, loops will be ignored, but might cause the tree to be incorrect (at the loop)
54          /// </summary>
55          /// <typeparam name="I"></typeparam>
56          /// <typeparam name="O"></typeparam>
57          /// <param name="getChildrenMethod"></param>
58          /// <param name="starts"></param>
59          /// <returns></returns>
60          public static C5.IList<I> ToposortAdv<I, O>(GetChildren<I,O> getChildrenMethod, params I[] starts) where O:IEnumerable<I>
61          {
62              HashedLinkedList<I> sorted = new HashedLinkedList<I>();
63              foreach (I start in starts)
64              {
65                  if (!sorted.Contains(start))
66                  {
67                      sorted.InsertLast(start);
68                      AddNodeAdv(getChildrenMethod, sorted, start);
69                  }
70              }
71              sorted.Reverse();
72              return sorted;
73          }
74
75          private static void AddNodeAdv<I, O>(GetChildren<I, O> getChildrenMethod, C5.IList<I> sorted, I node) where O :
        IEnumerable<I>
76          {
77              foreach (I child in getChildrenMethod(node))
78                  if (!sorted.Contains(child))
79                  {
80                      sorted.ViewOf(node).InsertFirst(child);
81                      AddNodeAdv(getChildrenMethod, sorted, child);
82                  }
83          }
84
85
86          public static C5.IList<I> ToposortNoRecursive<I, O>(GetChildren<I, O> getChildrenMethod, params I[] starts) where O :
        IEnumerable<I>
87          {
88              HashedLinkedList<I> sorted = new HashedLinkedList<I>();
89              foreach (I start in starts)
90                  if (!sorted.Contains(start))
91                  {
92                      sorted.InsertLast(start);
93                      using (C5.IList<I> cursor = sorted.View(sorted.Count - 1, 1))
94                      {
95                          do
96                          {
97                              I child;
98                              while (PendingChild(getChildrenMethod, sorted, cursor.First, out child))
99                              {
```

```
100                                 cursor.InsertFirst(child);
101                                 cursor.Slide(0, 1);
102                             }
103                         } while (cursor.TrySlide(+1));
104                     }
105                 }
106             sorted.Reverse();
107             return sorted;
108         }
109         static bool PendingChild<I,O>(GetChildren<I, O> getChildrenMethod, C5.IList<I> sorted, I node, out I returnChild) where O ↙
    : IEnumerable<I>
110         {
111             foreach (I child in getChildrenMethod(node))
112                 if (!sorted.Contains(child))
113                 {
114                     returnChild = child;
115                     return true;
116                 }
117             returnChild = default(I);
118             return false;
119         }
120 
121 
122         /// <summary>
123         /// Mark III
124         /// </summary>
125         /// <typeparam name="I"></typeparam>
126         /// <typeparam name="O"></typeparam>
127         /// <param name="getChildrenMethod"></param>
128         /// <param name="starts"></param>
129         /// <returns></returns>
130         public static C5.IList<FullCellAddr<Sheet>> ToposortAdvWithBoolArray(params FullCellAddr<Sheet>[] starts)
131         {
132             HashedLinkedList<FullCellAddr<Sheet>> sorted = new HashedLinkedList<FullCellAddr<Sheet>>();
133             foreach (FullCellAddr<Sheet> start in starts)
134             {
135                 if (!start.sheet.InTopologicalSort[start.col, start.row])
136                 {
137                     sorted.InsertLast(start);
138                     start.sheet.InTopologicalSort[start.col, start.row] = true;
139                     AddNodeAdvWithBoolArray(sorted, start);
140                 }
141             }
142             sorted.Reverse();
143             return sorted;
144         }
145 
146         private static void AddNodeAdvWithBoolArray(HashedLinkedList<FullCellAddr<Sheet>> sorted, FullCellAddr<Sheet> node)
147         {
148             if(node.sheet.SupportGraph[node.col, node.row] == null)
149                 return;
```

```
150
151             foreach (FullCellAddr<Sheet> child in node.sheet.SupportGraph[node.col, node.row].GetAdresses(node.col, node.row))
152                 if (!child.sheet.InTopologicalSort[child.col,child.row])
153                 {
154                     sorted.ViewOf(node).InsertFirst(child);
155                     child.sheet.InTopologicalSort[child.col, child.row] = true;
156                     AddNodeAdvWithBoolArray(sorted, child);
157                 }
158         }
159
160         /// <summary>
161         ///
162         /// </summary>
163         /// <param name="starts"></param>
164         /// <returns></returns>
165         public static C5.ArrayList<FullCellAddr<Sheet>> ToposortAdvWithBoolArrayAndSpecialArray(params FullCellAddr<Sheet>[] starts)
166         {
167             System.Collections.Generic.LinkedList<FullCellAddr<Sheet>> sorted = new System.Collections.Generic.LinkedList<FullCellAddr<Sheet>>();
168             foreach (FullCellAddr<Sheet> start in starts)
169             {
170                 if (!start.sheet.InTopologicalSort[start.col, start.row])
171                 {
172                     sorted.AddLast(start);
173                     start.sheet.InTopologicalSort[start.col, start.row] = true;
174                     AddNodeAdvWithBoolArrayAndSpecialArray(sorted, sorted.Last);
175                 }
176             }
177
178             ArrayList<FullCellAddr<Sheet>> back = new ArrayList<FullCellAddr<Sheet>>(sorted.Count);
179             if(sorted.Count != 0)
180             {
181                 LinkedListNode<FullCellAddr<Sheet>> current = sorted.Last;
182                 do
183                 {
184                     back.Add(current.Value);
185                 } while ((current = current.Previous) != null);
186             }
187
188             return back;
189         }
190
191         private static void AddNodeAdvWithBoolArrayAndSpecialArray(System.Collections.Generic.LinkedList<FullCellAddr<Sheet>> sorted, LinkedListNode<FullCellAddr<Sheet>> listNode)
192         {
193             FullCellAddr<Sheet> node = listNode.Value;
194
195             if (node.sheet.SupportGraph[node.col, node.row] == null)
196                 return;
197
```

```
198             FullCellAddrArrayList<Sheet> tempList = node.sheet.SupportGraph[node.col, node.row].GetAdressesAlt(node.col, node.row)↙
    ;
199
200             for (int i = 0; i < tempList.Count; i++)
201             {
202                 FullCellAddr<Sheet> child = tempList[i];
203                 if (!child.sheet.InTopologicalSort[child.col, child.row])
204                 {
205                     LinkedListNode<FullCellAddr < Sheet >> newNode = new LinkedListNode<FullCellAddr<Sheet>>(child);
206                     sorted.AddBefore(listNode, newNode);
207                     child.sheet.InTopologicalSort[child.col, child.row] = true;
208                     AddNodeAdvWithBoolArrayAndSpecialArray(sorted, newNode);
209                 }
210             }
211         }
212
213
214         /// <summary>
215         /// Mark IV
216         /// </summary>
217         /// <param name="starts"></param>
218         /// <returns></returns>
219         public static C5.IList<FullCellAddr<Sheet>> ToposortTestWithBoolArrayAndArrayList(params FullCellAddr<Sheet>[] starts)
220         {
221             ArrayList<FullCellAddr<Sheet>> sorted = new ArrayList<FullCellAddr<Sheet>>();
222             foreach (FullCellAddr<Sheet> start in starts)
223                 if (!start.sheet.InTopologicalSort[start.col, start.row])
224                     AddNodeTestWithBoolArrayAndArrayList(sorted, start);
225             sorted.Reverse();
226             return sorted;
227         }
228
229         private static void AddNodeTestWithBoolArrayAndArrayList(C5.IList<FullCellAddr<Sheet>> sorted, FullCellAddr<Sheet> node)
230         {
231             if (node.sheet.SupportGraph[node.col, node.row] != null)
232             {
233                 foreach (FullCellAddr<Sheet> child in node.sheet.SupportGraph[node.col, node.row].GetAdresses(node.col, node.row))
234                     if (!child.sheet.InTopologicalSort[child.col, child.row])
235                     {
236                         AddNodeTestWithBoolArrayAndArrayList(sorted, child);
237                     }
238             }
239             sorted.InsertLast(node);
240             node.sheet.InTopologicalSort[node.col, node.row] = true;
241         }
242
243         public static C5.IList<FullCellAddr<Sheet>> ToposortAdvWithBoolArrayAndExternAdd(params FullCellAddr<Sheet>[] starts)
244         {
245             HashedLinkedList<FullCellAddr<Sheet>> sorted = new HashedLinkedList<FullCellAddr<Sheet>>();
246             foreach (FullCellAddr<Sheet> start in starts)
247             {
```

```
248                 if (!start.sheet.InTopologicalSort[start.col, start.row])
249                 {
250                     sorted.InsertLast(start);
251                     start.sheet.InTopologicalSort[start.col, start.row] = true;
252                     AddNodeAdvWithBoolArrayAndExternAdd(sorted, start);
253                 }
254             }
255             sorted.Reverse();
256             return sorted;
257         }
258 
259         private static void AddNodeAdvWithBoolArrayAndExternAdd(HashedLinkedList<FullCellAddr<Sheet>> sorted, FullCellAddr<Sheet> ↙
    node)
260         {
261             if (node.sheet.SupportGraph[node.col, node.row] == null)
262                 return;
263 
264             foreach (FullCellAddr<Sheet> child in node.sheet.SupportGraph[node.col, node.row].GetAdressesExternCheck(node.col,    ↙
    node.row))
265             {
266                 sorted.ViewOf(node).InsertFirst(child);
267                 AddNodeAdvWithBoolArrayAndExternAdd(sorted, child);
268             }
269         }
270     }
271 }
272 
```

Modified classes

CoreCalc.Coco

Spreadsheet.ATG

```
1 // CoreCalc, a spreadsheet core implementation
2
3 // ------------------------------------------------------------------------
4 // Copyright (c) 2006 Peter Sestoft and Thomas S. Iversen
5
6 // Permission is hereby granted, free of charge, to any person
7 // obtaining a copy of this software and associated documentation
8 // files (the "Software"), to deal in the Software without
9 // restriction, including without limitation the rights to use, copy,
10 // modify, merge, publish, distribute, sublicense, and/or sell copies
11 // of the Software, and to permit persons to whom the Software is
12 // furnished to do so, subject to the following conditions:
13
14 //  * The above copyright notice and this permission notice shall be
15 //    included in all copies or substantial portions of the Software.
16
17 //  * The software is provided "as is", without warranty of any kind,
18 //    express or implied, including but not limited to the warranties of
19 //    merchantability, fitness for a particular purpose and
20 //    noninfringement.  In no event shall the authors or copyright
21 //    holders be liable for any claim, damages or other liability,
22 //    whether in an action of contract, tort or otherwise, arising from,
23 //    out of or in connection with the software or the use or other
24 //    dealings in the software.
25 // ------------------------------------------------------------------------
26
27 // Coco/R grammar for spreadsheet formulas
28
29 // To build:
30 //   coco -namespace CoreCalc Spreadsheet.ATG
31
32 // New:
33 //  - RaRefs in the R1C1 format
34 //  - The string concatenation operator &
35 //  - Numbers in scientific notation
36 //  - Sheetreferences in the style: [Alpha{Alpha}!]Raref
37 //  - ^ (power) operator (April 2006).
38 //
39
40 using System.Collections;
41
42 COMPILER Spreadsheet
43
44   private int col, row;
45   private Workbook workbook;
46   private Cell cell;
47   System.Globalization.NumberFormatInfo ni = null;
48
49   public void SetNumberDecimalSeparator(String nds) {
50          System.Globalization.CultureInfo ci =
51             System.Globalization.CultureInfo.InstalledUICulture;
```

```
         this.ni = (System.Globalization.NumberFormatInfo)
            ci.NumberFormat.Clone();
         this.ni.NumberDecimalSeparator = nds;
  }


  public Cell ParseCell(Workbook workbook, int col, int row) {
    this.workbook = workbook;
    this.col = col; this.row = row;

    // US decimal point, regardless of culture
    SetNumberDecimalSeparator(".");
    Parse();
    return cell;
  }

/*--------------------------------------------------------------------------*/
CHARACTERS
  letter = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.".
  atoi   = "ABCDEFGHIabcdefghi".
  digit = "0123456789".
  Alpha = letter + digit.
  cr  = '\r'.
  lf  = '\n'.
  tab = '\t'.
  exclamation = '!'.
  dollar  = '$'.
  numbersign = '#'.
  question = '?'.
  newLine = cr + lf.
  strchar    = ANY - '"' - '\\' - newLine.
  char       = ANY - '\\' - newLine.


TOKENS
  name       = letter { letter } CONTEXT("(").
  number     =
               digit { digit }
               [                                  /* optional fraction          */
                [ "." digit { digit }]            /* optional fractional digits */
                [ ( "E" | "e" )                   /* optional exponent          */
                  [ "+" | "-" ]                   /* optional exponent sign     */
                  digit { digit }
                ]
               ].
  sheetref   = Alpha { Alpha } exclamation.
  raref      = [ dollar ] atoi { letter } [ dollar ] digit { digit }.
  xmlssraref11= "RC".
  xmlssraref12= "RC" digit { digit }.
  xmlssraref13= "RC[" ["+"|"-"] digit { digit } "]".
  xmlssraref21= "R" digit { digit } "C".
```

```
  xmlssraref22= "R" digit { digit } "C" digit {digit}.
  xmlssraref23= "R" digit { digit } "C[" ["+"|"-"] digit { digit } "]".
  xmlssraref31= "R[" ["+"|"-"] digit { digit } "]C".
  xmlssraref32= "R[" ["+"|"-"] digit { digit } "]C" digit { digit }.
  xmlssraref33= "R[" ["+"|"-"] digit { digit } "]C[" ["+"|"-"] digit { digit } "]".
  string     = "\"" { strchar } "\"".
  textcell   = "\'" { char }.
  error      =  numbersign { strchar } [exclamation|question].

COMMENTS FROM "/*" TO "*/" NESTED
COMMENTS FROM "//" TO cr lf

IGNORE cr + lf + tab


PRODUCTIONS
/*-------------------------------------------------------------------------*/
ConcatOp<out String op>
=                       (. op = "&"; .)
 ('&').


AddOp<out String op>
=                        (. op = "+"; .)
  ( '+'
  | '-'                  (. op = "-"; .)
  ).

LogicalOp<out String op>
=                        (. op = "=="; .)
  ( "="
  | "=="
  | "<>"                 (. op = "<>"; .)
  | "<"                  (. op = "<";  .)
  | "<="                 (. op = "<="; .)
  | ">"                  (. op = ">";  .)
  | ">="                 (. op = ">="; .)
  ).

/*-------------------------------------------------------------------------*/
Expr<out Expr e>         (. Expr e2; String op; e = null; .)
=  Concat<out e>
  { LogicalOp<out op>
    Concat<out e2>  (. e = new FunCall(op, new Expr[] { e, e2 }); .)
  }
  .

Concat<out Expr e>         (. Expr e2; String op; .)
=   LogicalTerm<out e>
  { ConcatOp<out op>
    LogicalTerm<out e2>    (. e = new FunCall(op, new Expr[] { e, e2 } ); .)
```

```
154   }
155   .
156
157
158 LogicalTerm<out Expr e>  (. Expr e2; String op; e = null; .)
159 = Term<out e>
160   { AddOp<out op>
161     Term<out e2>         (. e = new FunCall(op, new Expr[] { e, e2 }); .)
162   }
163   .
164 /*-------------------------------------------------------------------------*/
165 Prefix<out Expr e>      (. e = null; .)
166 = "-" Prefix<out e>     (. e = new FunCall("-", new Expr[] {e} ); .)
167   | "+" Prefix<out e>
168   | Factor<out e>
169   .
170
171
172 /*-------------------------------------------------------------------------*/
173 Factor<out Expr e>       (. RARef r1, r2; Sheet s1 = null; double d; e = null; .)
174 = (
175    | sheetref            (.
176                                String sheetname = t.val.TrimEnd("!".ToCharArray());
177                                s1 = workbook[sheetname];
178                          .)
179   )
180    Raref<out r1> (       (. e = new CellRef(s1, r1);      .)
181     | ':' Raref<out r2>  (. e = new CellArea(s1, r1, r2); .)
182    )
183   | "TRUE"               (. e = new NumberConst(1.0); .)
184   | "FALSE"              (. e = new NumberConst(0.0); .)
185   | Number<out d>        (. e = new NumberConst(d);         .)
186   | string               (. int len = t.val.Length-2;
187                             e = new TextConst(t.val.Substring(1, len));
188                          .)
189   | '(' Expr<out e> ')'
190   | Application<out e>
191   .
192
193 /*-------------------------------------------------------------------------*/
194 PowFactor<out Expr e>    (. Expr e2; e= null; .)
195 =  Prefix<out e>
196   { '^'
197     Prefix<out e2>       (. e = new FunCall("^", new Expr[] { e, e2 } ); .)
198   }
199   .
200 /*-------------------------------------------------------------------------*/
201
202 Raref<out RARef raref>   (. raref = null;.)
203 = raref                  (. raref = new RARef(t.val, col, row); .)
204  | xmlssraref11           (. raref = new RARef(t.val); .)
```

```
 | xmlssraref12           (. raref = new RARef(t.val); .)
 | xmlssraref13           (. raref = new RARef(t.val); .)
 | xmlssraref21           (. raref = new RARef(t.val); .)
 | xmlssraref22           (. raref = new RARef(t.val); .)
 | xmlssraref23           (. raref = new RARef(t.val); .)
 | xmlssraref31           (. raref = new RARef(t.val); .)
 | xmlssraref32           (. raref = new RARef(t.val); .)
 | xmlssraref33           (. raref = new RARef(t.val); .)
.

/*-------------------------------------------------------------------------*/
Number<out double d>     (. d = 0.0;                       .)
= ( number               (. d = double.Parse(t.val, ni); .)
 /* | '-' number           (. d = -double.Parse(t.val, ni); .)*/
  ) .
/*-------------------------------------------------------------------------*/
Application<out Expr e>  (. String s; Expr[] es; e = null; .)
= Name<out s> '('
  ( ')'                  (. e = new FunCall(s, new Expr[0]); .)
    | Exprs1<out es> ')' (. e = new FunCall(s, es); .)
  )
.
/*-------------------------------------------------------------------------*/
Exprs1<out Expr[] es>    (. Expr e1, e2;
                            ArrayList elist = new ArrayList();
                         .)
= ( Expr<out e1>         (. elist.Add(e1); .)
    { (';' | ',') Expr<out e2>   (. elist.Add(e2); .)
    }
  )                      (. es = new Expr[elist.Count];
                            elist.CopyTo(es, 0);
                         .)
.
/*-------------------------------------------------------------------------*/
Name<out String s>
= name                   (. s = t.val; .)
.
/*-------------------------------------------------------------------------*/
MulOp<out String op>
=                        (. op = "*"; .)
  ( '*'
  | '/'                  (. op = "/"; .)
  ).
/*-------------------------------------------------------------------------*/
Term<out Expr e>         (. Expr e2; String op; .)
=   PowFactor<out e>
  { MulOp<out op>
    PowFactor<out e2>    (. e = new FunCall(op, new Expr[] { e, e2 } ); .)
  }
  .
/*-------------------------------------------------------------------------*/
```

```
Spreadsheet              (. Expr e; double d;     .)
= ( '=' Expr<out e>      (. this.cell = new Formula(workbook, e); .)
  | textcell             (. this.cell = new TextCell(t.val.Substring(1)); .)
  | Number<out d>        (. this.cell = new NumberCell(d);  .)
  | string               (. this.cell = new TextCell(t.val.Substring(1)); .)
  | error                (. this.cell = new Formula(workbook, new Error(t.val.Substring(1))); .)

  ).

END Spreadsheet.

```

CoreCalc.Gui

GUI

```
// CoreCalc, a spreadsheet core implementation

// ------------------------------------------------------------------------
// Copyright (c) 2006 Peter Sestoft and Thomas Iversen

// Permission is hereby granted, free of charge, to any person
// obtaining a copy of this software and associated documentation
// files (the "Software"), to deal in the Software without
// restriction, including without limitation the rights to use, copy,
// modify, merge, publish, distribute, sublicense, and/or sell copies
// of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:

//  * The above copyright notice and this permission notice shall be
//    included in all copies or substantial portions of the Software.

//  * The software is provided "as is", without warranty of any kind,
//    express or implied, including but not limited to the warranties of
//    merchantability, fitness for a particular purpose and
//    noninfringement.  In no event shall the authors or copyright
//    holders be liable for any claim, damages or other liability,
//    whether in an action of contract, tort or otherwise, arising from,
//    out of or in connection with the software or the use or other
//    dealings in the software.
// ------------------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Drawing;                   // Size
using System.Text;                      // StringBuilder
using System.Windows.Forms;
using C5;
using CoreCalc.SupportGraf;
using FAP;
using CoreCalc.AlternativeIO;
// DataGridView, Form, etc

namespace CoreCalc {
  // ---------------------------------------------------------------
  // A WorkbookForm is the user interface of a workbook

  public partial class WorkbookForm : Form {
    private Workbook workbook;
    private bool useSupportGraph;
    private bool recalculateVolatileFunctions;
    private bool manualRecalculation;
    private bool showTotalNumberOfSupportedCells;
    private TopologicalSortType topoSort;

      public bool UseSupportGraph
```

```
52        {
53            get { return useSupportGraph; }
54            set { useSupportGraph = value; }
55        }
56
57        public bool RecalculateVolatileFunctions
58        {
59            get { return recalculateVolatileFunctions; }
60            set { recalculateVolatileFunctions = value; }
61        }
62
63        public bool ManualRecalculation
64        {
65            get { return manualRecalculation; }
66            set { manualRecalculation = value; }
67        }
68
69        public bool GetShowTotalNumberOfSupportedCells
70        {
71            get { return showTotalNumberOfSupportedCells; }
72        }
73
74      public WorkbookForm(Workbook workbook) {
75        this.Workbook = workbook;
76        SetToposort(workbook.Topsort);
77        InitializeComponent();
78
79        const int guiHeight = 500;
80        const int guiWidth = 800;
81
82        //StartPosition = FormStartPosition.CenterScreen;
83        //Use AllScreens to get all screens but remember that all coordinates are relative to the primary screen
84        Screen screen = Screen.PrimaryScreen;
85        Rectangle primaryScreen = screen.WorkingArea;
86        StartPosition = FormStartPosition.Manual;
87        Size = new Size(guiWidth, guiHeight);
88        Logger.Log.Debug(typeof(WorkbookForm), "Primary display resolution detected: "+primaryScreen.Width + "x" + primaryScreen.
      Height);
89        Location = new Point(primaryScreen.Width-guiWidth, primaryScreen.Height-guiHeight);
90        statusRefFmt.Text = workbook.Format.RefFmt.ToString();
91
92        RecalculateVolatileFunctions = recalculateVolatileFunctionsToolStripMenuItem.Checked;
93        UseSupportGraph = recalculateToolStripMenuItem.Checked;
94        ManualRecalculation = manualRecalculationToolStripMenuItem.Checked;
95        workbook.attachGui(this);
96        DisplayWorkbook();
97        ShowDialog();
98
99
100     }
101
```

```
102       private void SetToposort(TopologicalSortType topologicalSortType)
103       {
104           if (TopoSortStandardToolStripMenuItem != null)
105           {
106               if (topologicalSortType == TopologicalSortType.Standard)
107                   TopoSortStandardToolStripMenuItem.SelectedIndex = 0;
108               else if (topologicalSortType == TopologicalSortType.NoloopsAllowed)
109                   TopoSortStandardToolStripMenuItem.SelectedIndex = 1;
110               else if (topologicalSortType == TopologicalSortType.NoRecursion)
111                   TopoSortStandardToolStripMenuItem.SelectedIndex = 2;
112               else if (topologicalSortType == TopologicalSortType.UseBoolArray)
113                   TopoSortStandardToolStripMenuItem.SelectedIndex = 3;
114               else if (topologicalSortType == TopologicalSortType.NoLinkedListStandardBoolArray)
115                   TopoSortStandardToolStripMenuItem.SelectedIndex = 4;
116               else
117                   throw new NotImplemented xception("This sort of topological sort is not implemented as being default");
118           }
119       }
120
121     private void exitToolStripMenuItem_Click(object sender,  ventArgs e) {
122       System. nvironment. xit(0);
123     }
124
125     private void aboutToolStripMenuItem1_Click(object sender,  ventArgs e) {
126       Form aboutBox = new AboutBox();
127       aboutBox.ShowDialog();
128     }
129
130     private void a1ToolStripMenuItem_Click(object sender,  ventArgs e) {
131       Workbook.Format.RefFmt = Formats.RefType.A1;
132       Reshow();
133     }
134
135     private void c0R0ToolStripMenuItem_Click(object sender,  ventArgs e) {
136       Workbook.Format.RefFmt = Formats.RefType.C0R0;
137       Reshow();
138     }
139
140     private void r1C1ToolStripMenuItem_Click(object sender,  ventArgs e) {
141       Workbook.Format.RefFmt = Formats.RefType.R1C1;
142       Reshow();
143     }
144
145     // Add new sheet
146     private void newWorkbookToolStripMenuItem_Click(object sender,  ventArgs e) {
147       if (Workbook != null) {
148         String name = "Sheet" + (Workbook.SheetCount + 1);
149         SheetTab sheetTab = new SheetTab(this, new Sheet(Workbook, name));
150         sheetHolder.TabPages.Add(sheetTab);
151         sheetHolder.SelectTab(sheetTab);
152       }
```

```
153     }
154
155     private void recalculateToolStripMenuItem_Click(object sender,  ventArgs e) {
156       RecalculateFull();
157     }
158
159     // Recalculate and reshow workbook
160
161     private void RecalculateFull() {
162       if (Workbook != null) {
163           Workbook.RecalculateFull();
164         Reshow();
165       }
166     }
167
168     private void Reshow() {
169       if (SelectedSheet != null)
170         formulaBox.Text = SelectedSheet.Reshow();
171       statusRefFmt.Text = Workbook.Format.RefFmt.ToString();
172     }
173
174     // Copy cell
175     private void copyToolStripMenuItem_Click(object sender,  ventArgs e) {
176       if (SelectedSheet != null)
177         SelectedSheet.Copy();
178     }
179
180     // Delete cell
181     private void deleteToolStripMenuItem_Click(object sender,  ventArgs e) {
182       if (SelectedSheet != null)
183         SelectedSheet.Delete();
184     }
185
186     // Insert column
187     private void columnToolStripMenuItem_Click(object sender,  ventArgs e) {
188       if (SelectedSheet != null)
189         SelectedSheet.InsertColumns(1);
190     }
191
192     // Insert row
193     private void rowToolStripMenuItem_Click(object sender,  ventArgs e) {
194       if (SelectedSheet != null)
195         SelectedSheet.InsertRows(1);
196     }
197
198     // Paste copied or cut cells, or text
199     private void pasteToolStripMenuItem_Click(object sender,  ventArgs e) {
200       if (SelectedSheet != null)
201         SelectedSheet.Paste();
202     }
203
```

```
204     // Get selected sheet if any, else null
205     private SheetTab SelectedSheet {
206       get {
207         return sheetHolder.TabCount > 0 ? sheetHolder.SelectedTab as SheetTab : null;
208       }
209     }
210 
211       private Workbook Workbook
212       {
213           get { return workbook; }
214           set
215           {
216               if(workbook != null && value != workbook)
217                   workbook.Dispose();
218               workbook = value;
219           }
220       }
221 
222       public TopologicalSortType TopoSort
223       {
224           get { return topoSort; }
225           set { topoSort = value; }
226       }
227 
228       private void formulaBox_TextChanged(object sender,  ventArgs e) {
229       if (SelectedSheet != null)
230         SelectedSheet.ChangeCurrentText(formulaBox.Text);
231     }
232 
233     private void formulaBox_KeyPress(object sender, KeyPress ventArgs e) {
234       if (SelectedSheet != null) {
235         if (e.KeyChar == (char)Keys.Return) {
236           SelectedSheet.SetCurrentCell(formulaBox.Text);
237           e.Handled = true;
238         } else if (e.KeyChar == (char)Keys. scape) {
239           SelectedSheet.Focus();
240           e.Handled = true;
241         }
242       }
243     }
244 
245     private void importSheetToolStripMenuItem_Click(object sender,  ventArgs e) {
246       OpenFileDialog ofd = new OpenFileDialog();
247       WorkBookIO workbookio = new WorkBookIO();
248       ofd.Filter = workbookio.SupportedFormatFilter();
249       ofd.FilterIndex = workbookio.DefaultFormatIndex();
250       if (ofd.ShowDialog() == DialogResult.OK) {
251         Clear();
252         System.Diagnostics.Stopwatch stopwatch = new Stopwatch();
253         stopwatch.Start();
254         Workbook = workbookio.Read(ofd.FileName);
```

```
255           stopwatch.Stop();
256           Logger.Log.Fatal(this, "Spreadsheet loaded in " + stopwatch. lapsed);
257         Workbook.attachGui(this);
258         DisplayWorkbook();
259       }
260     }
261
262     private void DisplayWorkbook() {
263       if (Workbook != null) {
264         foreach (Sheet sheet in Workbook)
265           sheetHolder.TabPages.Add(new SheetTab(this, sheet));
266         if (sheetHolder.TabCount > 0)
267           sheetHolder.SelectTab(0);
268         if(UseSupportGraph)
269         {
270             workbook.BuildSupportGraph();
271         }
272         if (!ManualRecalculation) Reshow();
273       }
274     }
275
276
277     public void Clear() {
278       Workbook.Clear();
279       sheetHolder.TabPages.Clear();
280     }
281
282       private void recalculateToolStripMenuItem1_CheckedChanged(object sender,  ventArgs e)
283       {
284           UseSupportGraph = ((ToolStripMenuItem) sender).Checked;
285       }
286
287       private void buildSupportgraphToolStripMenuItem_Click(object sender,  ventArgs e)
288       {
289           Workbook.BuildSupportGraph();
290       }
291
292       private void startSupportGraphNormalizerToolStripMenuItem_Click(object sender,  ventArgs e)
293       {
294           Workbook.StartNormalizer();
295       }
296
297       private void stopSupportGraphNormalizerToolStripMenuItem_Click(object sender,  ventArgs e)
298       {
299           Workbook.StopNormalizer();
300       }
301
302       private void stopNormalizerLogToolStripMenuItem_Click(object sender,  ventArgs e)
303       {
304           Workbook.ThreadLogging(false);
305
```

```
306         }
307
308         private void startNormalizerLogToolStripMenuItem_Click(object sender,  ventArgs e)
309         {
310             Workbook.ThreadLogging(true);
311         }
312
313         private void cutToolStripMenuItem_Click(object sender,  ventArgs e)
314         {
315             if (SelectedSheet != null)
316                 SelectedSheet.Cut();
317         }
318
319         private void showVolatileFunctionsToolStripMenuItem_Click(object sender,  ventArgs e)
320         {
321             workbook.PrintListOfVolatileFunctions();
322         }
323
324         private void recalculateVolatileFunctionsToolStripMenuItem_Click(object sender,  ventArgs e)
325         {
326             RecalculateVolatileFunctions = ((ToolStripMenuItem)sender).Checked;
327             if(RecalculateVolatileFunctions) Logger.Log.Info(this, "Recalculate volatile functions enabled");
328             else Logger.Log.Info(this, "Recalculate volatile functions disabled");
329         }
330
331         private void manualRecalculationToolStripMenuItem_Click(object sender,  ventArgs e)
332         {
333             ManualRecalculation = ((ToolStripMenuItem)sender).Checked;
334             if (RecalculateVolatileFunctions) Logger.Log.Info(this, "Manual recalculation enabled");
335             else Logger.Log.Info(this, "Manual recalculation disabled");
336         }
337
338         private void showToolStripMenuItem_Click(object sender,  ventArgs e)
339         {
340             workbook.PrintListOfQeuedCellsWaitingForManualUpdate();
341         }
342
343         private void showMemoryRelatedInfoToolStripMenuItem_Click(object sender,  ventArgs e)
344         {
345             Logger.Log.Fatal(this, Test.PrintMemoryRelatedInfo(Workbook));
346         }
347
348         private void RemoveDucblicatesToolStripMenuItem_Click(object sender,  ventArgs e)
349         {
350             Stopwatch timer = new Stopwatch();
351             timer.Start();
352             foreach (Sheet sheet in workbook)
353             {
354                 sheet.SupportGraph.RemoveDublicateFAPDoubleLists();
355             }
356             timer.Stop();
```

```
357             Logger.Log.Fatal(this, "Remove duplicate FAP sets acress columns took: " + timer. lapsed);
358         }
359
360         private void showTotalNumberOfSupportedCellsToolStripMenuItem_Click(object sender,  ventArgs e)
361         {
362             showTotalNumberOfSupportedCells = ((ToolStripMenuItem)sender).Checked;
363         }
364
365         private void changeSelectedCells11TimesToolStripMenuItem_Click(object sender,  ventArgs e)
366         {
367
368         }
369
370         private void fewToolStripMenuItem_Click(object sender,  ventArgs e)
371         {
372             Test. xecuteTest(workbook, "initFewFINAN", this);
373         }
374
375         private void toolStripMenuItem4_Click(object sender,  ventArgs e)
376         {
377             Test. xecuteTest(workbook, "initHalfFINAN", this);
378         }
379
380         private void allToolStripMenuItem_Click(object sender,  ventArgs e)
381         {
382             Test. xecuteTest(workbook, "initAllFINAN", this);
383         }
384
385         private void fullRecalculationToolStripMenuItem_Click(object sender,  ventArgs e)
386         {
387
388         }
389
390         private void fullToolStripMenuItem_Click(object sender,  ventArgs e)
391         {
392             Test. xecuteTest(workbook, "initFullFINAN", this);
393         }
394
395         private void fullToolStripMenuItem1_Click(object sender,  ventArgs e)
396         {
397             Test. xecuteTest(workbook, "initFullALPHA", this);
398         }
399
400         private void fewToolStripMenuItem1_Click(object sender,  ventArgs e)
401         {
402             Test. xecuteTest(workbook, "initFewALPHA", this);
403         }
404
405         private void halfToolStripMenuItem_Click(object sender,  ventArgs e)
406         {
407             Test. xecuteTest(workbook, "initHalfALPHA", this);
```

```
408         }
409
410         private void allToolStripMenuItem1_Click(object sender,  ventArgs e)
411         {
412             Test. xecuteTest(workbook, "initAllALPHA", this);
413         }
414
415         private void fullToolStripMenuItem2_Click(object sender,  ventArgs e)
416         {
417             Test. xecuteTest(workbook, "initFullPROBA", this);
418         }
419
420         private void fewToolStripMenuItem2_Click(object sender,  ventArgs e)
421         {
422             Test. xecuteTest(workbook, "initFewPROBA", this);
423         }
424
425         private void halfToolStripMenuItem1_Click(object sender,  ventArgs e)
426         {
427             Test. xecuteTest(workbook, "initHalfPROBA", this);
428         }
429
430         private void allToolStripMenuItem2_Click(object sender,  ventArgs e)
431         {
432             Test. xecuteTest(workbook, "initAllPROBA", this);
433         }
434
435         private void fullToolStripMenuItem3_Click(object sender,  ventArgs e)
436         {
437             Test. xecuteTest(workbook, "initFullROL ", this);
438         }
439
440         private void fewToolStripMenuItem3_Click(object sender,  ventArgs e)
441         {
442             Test. xecuteTest(workbook, "initFewROL ", this);
443         }
444
445         private void halfToolStripMenuItem2_Click(object sender,  ventArgs e)
446         {
447             Test. xecuteTest(workbook, "initHalfROL ", this);
448         }
449
450         private void allToolStripMenuItem3_Click(object sender,  ventArgs e)
451         {
452             Test. xecuteTest(workbook, "initAllROL ", this);
453         }
454
455         private void fullToolStripMenuItem4_Click(object sender,  ventArgs e)
456         {
457             Test. xecuteTest(workbook, "initFullPROT", this);
458         }
```

```
459
460         private void fewToolStripMenuItem4_Click(object sender,  ventArgs e)
461         {
462             Test. xecuteTest(workbook, "initFewPROT", this);
463         }
464
465         private void halfToolStripMenuItem3_Click(object sender,  ventArgs e)
466         {
467             Test. xecuteTest(workbook, "initHalfPROT", this);
468         }
469
470         private void allToolStripMenuItem4_Click(object sender,  ventArgs e)
471         {
472             Test. xecuteTest(workbook, "initAllPROT", this);
473         }
474
475         private void altImportToolStripMenuItem_Click(object sender,  ventArgs e)
476         {
477             OpenFileDialog ofd = new OpenFileDialog();
478             if (ofd.ShowDialog() == DialogResult.OK)
479             {
480                 Clear();
481                 System.Diagnostics.Stopwatch stopwatch = new Stopwatch();
482                 stopwatch.Start();
483                 Workbook =  xcel2003XMLReader.ReadFile(ofd.FileName);
484                 stopwatch.Stop();
485                 Logger.Log.Fatal(this, "Spreadsheet loaded in " + stopwatch. lapsed);
486                 Workbook.WorkbookName = ofd.FileName.ToString();
487                 Workbook.attachGui(this);
488                 DisplayWorkbook();
489             }
490         }
491
492         private void showCellTypeInformationToolStripMenuItem_Click(object sender,  ventArgs e)
493         {
494             Logger.Log.Info(this, Test.GetNumberOfDifferentCellTypes(Workbook));
495         }
496
497         private void supportGraphCorrectnesToolStripMenuItem_Click(object sender,  ventArgs e)
498         {
499             Test.DummyTestAllSupportGraph dgesContraReferences(Workbook, true, true);
500         }
501
502         private void supportGraphPreToolStripMenuItem_Click(object sender,  ventArgs e)
503         {
504             Test.DummyTestAllSupportGraphReferencesContra dges(Workbook, true);
505         }
506
507         private void genericAllTestToolStripMenuItem_Click(object sender,  ventArgs e)
508         {
509             Test. xecuteTest(workbook, "initAllgeneric", this);
```

```
510         }
511
512         private void genericFullnoSupportGraphToolStripMenuItem_Click(object sender,  ventArgs e)
513         {
514             Test. xecuteTest(workbook, "initFullgeneric", this);
515         }
516
517         private void TopoSortStandardToolStripMenuItem_SelectedIndexChanged(object sender,  ventArgs e)
518         {
519             int selectedItem = TopoSortStandardToolStripMenuItem.SelectedIndex;
520             if (selectedItem == 0)
521                 workbook.Topsort = TopologicalSortType.Standard;
522             else if (selectedItem == 1)
523                 workbook.Topsort = TopologicalSortType.NoloopsAllowed;
524             else if (selectedItem == 2)
525                 workbook.Topsort = TopologicalSortType.NoRecursion;
526             else if (selectedItem == 3)
527                 workbook.Topsort = TopologicalSortType.UseBoolArray;
528             else if (selectedItem == 4)
529                 workbook.Topsort = TopologicalSortType.NoLinkedListStandardBoolArray;
530             else
531                 throw new NotImplemented xception("Have to implement this sort type");
532         }
533
534         private void WorkbookForm_Load(object sender,  ventArgs e)
535         {
536             if(workbook != null)
537               SetToposort(workbook.Topsort);
538         }
539
540         private void initialTestsToolStripMenuItem_Click(object sender,  ventArgs e)
541         {
542
543         }
544
545         private void rOL ToolStripMenuItem1_Click(object sender,  ventArgs e)
546         {
547             Test. xecuteTest(workbook, "markIIStructureSpeedRole", this);
548         }
549
550         private void pROTToolStripMenuItem1_Click(object sender,  ventArgs e)
551         {
552             Test. xecuteTest(workbook, "markIIStructureSpeedProt", this);
553         }
554
555         private void t STToolStripMenuItem1_Click(object sender,  ventArgs e)
556         {
557             Test. xecuteTest(workbook, "test", this);
558         }
559
560         private void rOL ToolStripMenuItem2_Click(object sender,  ventArgs e)
```

```
561         {
562             Test. xecuteTest(workbook, "markIIDegradationROL ", this);
563         }
564 
565         private void pROTToolStripMenuItem2_Click(object sender,  ventArgs e)
566         {
567             Test. xecuteTest(workbook, "markIIDegradationPROT", this);
568         }
569 
570         private void rOL ToolStripMenuItem3_Click(object sender,  ventArgs e)
571         {
572             Test. xecuteTest(workbook, "markIIISpeedRole", this);
573         }
574 
575         private void pROTToolStripMenuItem3_Click(object sender,  ventArgs e)
576         {
577             Test. xecuteTest(workbook, "markIIISpeedProt", this);
578         }
579 
580         private void completeTestToolStripMenuItem_Click(object sender,  ventArgs e)
581         {
582             Test. xecuteTest(workbook, "markISpeedCompleteFinan", this);
583         }
584 
585         private void completeTestToolStripMenuItem1_Click(object sender,  ventArgs e)
586         {
587             Test. xecuteTest(workbook, "markISpeedCompleteAlpha", this);
588         }
589 
590         private void completeTestToolStripMenuItem2_Click(object sender,  ventArgs e)
591         {
592             Test. xecuteTest(workbook, "markISpeedCompleteProba", this);
593         }
594 
595         private void completeTestToolStripMenuItem3_Click(object sender,  ventArgs e)
596         {
597             Test. xecuteTest(workbook, "markISpeedCompleteRole", this);
598         }
599 
600         private void completeTestToolStripMenuItem4_Click(object sender,  ventArgs e)
601         {
602             Test. xecuteTest(workbook, "markISpeedCompleteProt", this);
603         }
604     }
605 
606     // ------------------------------------------------------------
607     // A SheetTab is a displayed sheet: a tab page on the tab control
608 
609     public class SheetTab : TabPage {
610       private readonly Sheet sheet;
611       private readonly DataGridView dgv;
```

```
612     private readonly WorkbookForm gui;
613
614     public SheetTab(WorkbookForm gui, Sheet sheet) : base(sheet.Name) {
615       this.gui = gui;
616       this.sheet = sheet;
617       this.dgv = new DataGridView();
618       dgv.Show ditingIcon = false;
619       dgv.Dock = DockStyle.Fill;
620       Dock = DockStyle.Fill;
621       // Display formula in the current cell and computed value in other cells
622       dgv.CellFormatting +=
623         delegate(Object sender, DataGridViewCellFormatting ventArgs e) {
624           int col = e.ColumnIndex, row = e.RowIndex;
625           if (col == dgv.CurrentCellAddress.X && row == dgv.CurrentCellAddress.Y) {
626             Object obj = sheet.Show(col, row);
627             if (obj != null) {
628               e.Value = obj;
629               e.FormattingApplied = true;
630             }
631           } else {
632             Object obj = sheet.ShowValue(col, row);
633             if (obj != null) {
634               e.Value = obj;
635               e.FormattingApplied = true;
636             }
637           }
638         };
639       // Show current cell's address, and show formula in formula box
640       dgv.Cell nter +=
641         delegate(Object sender, DataGridViewCell ventArgs arg) {
642           int row = arg.RowIndex, col = arg.ColumnIndex;
643           dgv.TopLeftHeaderCell.Value = new CellAddr(col, row).ToString();
644           gui.formulaBox.Text = (String)dgv.CurrentCell.FormattedValue;
645           if (gui.UseSupportGraph)
646           {
647               try
648               {
649                   if (sheet.SupportGraph[col, row] != null)
650                   {
651                       string s = CellAddr.ColumnName(col) +
652                                  +(row + 1) + " directly supports \r\n" + sheet.SupportGraph[col, row].ToString()
653                       +"\r\n Cells: \r\n" + sheet.SupportGraph[col, row].ShowFullCellAdresses(col, row);
654                       if (gui.GetShowTotalNumberOfSupportedCells) s += "\r\nTotal number of supported cells: " +
    GetSupportedCellCount(col, row, sheet, null).Count;
655                       Logger.Log.Info(this, s);
656                   }else Logger.Log.Info(this, CellAddr.ColumnName(col) + (row + 1) + " supports nothing");
657
658               }
659               catch (System.IndexOutOfRange xception) { Logger.Log. rror(this, "Index error"); }
660           }
661         };
```

```
662       // Store cell's contents (formula) in sheet after edit
663       dgv.Cell nd dit +=
664         delegate(Object sender, DataGridViewCell ventArgs arg) {
665           int row = arg.RowIndex, col = arg.ColumnIndex;
666           Object value = dgv.CurrentCell.Value;
667
668           if (value != null)
669           {
670               sheet.InsertCell(value.ToString(), new CellAddr(col, row));
671               SetCell(col, row, value.ToString());
672           }
673           else
674               sheet.DeleteCell(new CellAddr(col, row));
675         };
676       dgv.ColumnCount = sheet.Cols + 1;
677       dgv.RowCount = sheet.Rows + 1;
678       dgv.AllowUserToAddRows = false;
679       // Put labels on columns and rows:
680       for (int col = 0; col < dgv.ColumnCount; col++)
681         dgv.Columns[col].Name = CellAddr.ColumnName(col);
682       for (int row = 0; row < dgv.RowCount; row++)
683         dgv.Rows[row].HeaderCell.Value = (row + 1).ToString();
684       Controls.Add(dgv);
685     }
686
687
688       //TODO: move this code out of this section
689       //TODO: ugly method, fix it :)
690       public C5.HashSet<FullCellAddr<Sheet>> GetSupportedCellCount(int c, int r, Sheet sheet, C5.HashSet<FullCellAddr<Sheet>>
      list)
691       {
692           if(list == null) list = new C5.HashSet<FullCellAddr<Sheet>>();
693           if (sheet.SupportGraph[c, r] == null) return list;
694           foreach (FullCellAddr<Sheet> adress in sheet.SupportGraph[c, r].GetAdresses(c, r))
695           {
696               if(list.Add(adress))
697                 GetSupportedCellCount(adress.col, adress.row, adress.sheet, list);
698           }
699           return list;
700       }
701
702     // Attempt to parse s as cell contents, and set selected cell(s)
703
704     public void SetCell(int col, int row, String text) {
705       Cell cell = Cell.Parse(text, sheet.workbook, col, row);
706       DataGridViewSelectedCellCollection dgvscc = dgv.SelectedCells;
707       if (dgvscc.Count > 1 && cell is Formula) { // Matrix formula
708         int ulCol = col, ulRow = row, lrCol = col, lrRow = row;
709         foreach (DataGridViewCell dgvc in dgvscc) {
710           ulCol = Math.Min(ulCol, dgvc.ColumnIndex);
711           ulRow = Math.Min(ulRow, dgvc.RowIndex);
```

```
712           lrCol = Math.Max(lrCol, dgvc.ColumnIndex);
713           lrRow = Math.Max(lrRow, dgvc.RowIndex);
714         }
715         CellAddr ulCa = new CellAddr(ulCol, ulRow),
716           lrCa = new CellAddr(lrCol, lrRow);
717         sheet.InsertMatrixFormula(cell, col, row, ulCa, lrCa);
718       } else // One-cell formula, or constant
719         sheet[col, row] = cell;
720         sheet.ContainsVolatileFunction(cell);
721         if(gui.ManualRecalculation)
722         {
723             sheet.workbook.WaitingForManualUpdate.Add(new FullCellAddr<Sheet>(col, row, sheet));
724         }
725         else RecomputeAndShow(new CellAddr(col, row));
726
727     }
728
729     // Copy sheet's currently active cell to Clipboard, also in text format
730
731     public void Copy() {
732       CellAddr ca = new CellAddr(dgv.CurrentCellAddress);
733       DataObject data = new DataObject();
734       data.SetData(DataFormats.Text, sheet[ca].Show(ca.col, ca.row, sheet.workbook.Format));
735       data.SetData(ClipboardCell.COPI D_C LL, new ClipboardCell(sheet.Name, ca));
736       Clipboard.Clear();
737       Clipboard.SetDataObject(data, false);
738     }
739
740       public void Cut()
741       {
742           CellAddr ca = new CellAddr(dgv.CurrentCellAddress);
743           DataObject data = new DataObject();
744           data.SetData(DataFormats.Text, sheet[ca].Show(ca.col, ca.row, sheet.workbook.Format));
745           data.SetData(ClipboardCell.CUT_C LL, new ClipboardCell(sheet.Name, ca));
746           Clipboard.Clear();
747           Clipboard.SetDataObject(data, false);
748       }
749
750     // Paste from the Clipboard.  Need to distinguish:
751     // 1. Paste from CoreCalc formula Copy: preserve sharing
752     // 2. Paste from CoreCalc cell Cut: adjust referring cells and this, if formula
753     // 3. Paste from text (e.g. from  xcel), parse to new cell
754     //TODO: does 2 and 3 ever get called?
755
756     public void Paste() {
757       if (Clipboard.ContainsData(ClipboardCell.COPI D_C LL)) {
758         // Copy CoreCalc cell
759           Stopwatch stopwatch = new Stopwatch();
760           stopwatch.Start();
761         ClipboardCell cc = (ClipboardCell)Clipboard.GetData(ClipboardCell.COPI D_C LL);
762         Logger.Log.Debug(this, "Pasting copied cell " + Clipboard.GetText());
```

```
763         Cell cell = sheet.workbook[cc.FromSheet][cc.FromCellAddr];
764         DataGridViewSelectedCellCollection dgvscc = dgv.SelectedCells;
765         ArrayList<CellAddr> cellsAltered = new ArrayList<CellAddr>();
766         int colStart;
767         int col nd;
768         int rowStart;
769         int row nd;
770         if (FindArea(dgvscc, out colStart, out col nd, out rowStart, out row nd, ref cellsAltered))
771         {
772             CellAddr newAddr = new CellAddr(colStart, rowStart);
773             sheet.PasteCell(cell, newAddr, col nd-colStart+1, row nd-rowStart+1);
774         }
775         else
776         {
777             foreach (DataGridViewCell dgvc in dgvscc)
778             {
779                 CellAddr newAddr = new CellAddr(dgvc.ColumnIndex, dgvc.RowIndex);
780                 sheet.PasteCell(cell, newAddr,
781                                 1, 1);
782             }
783         }
784         RecomputeAndShow(cellsAltered.ToArray());
785         stopwatch.Stop();
786         Logger.Log.Fatal(this, "Copy cells took: " + stopwatch. lapsed);
787       } else if (Clipboard.ContainsData(ClipboardCell.CUT_C LL)) {
788         // Move CoreCalc cell
789         Logger.Log.Debug(this, "Pasting moved cell");
790           //TODO: only moves data, but does not update direct dependencies
791           ClipboardCell cc = (ClipboardCell)Clipboard.GetData(ClipboardCell.CUT_C LL);
792           Console.WriteLine("Pasting moved cell " + Clipboard.GetText());
793           Cell cell = sheet.workbook[cc.FromSheet][cc.FromCellAddr];
794           DataGridViewSelectedCellCollection dgvscc = dgv.SelectedCells;
795           ArrayList<CellAddr> cellsAltered = new ArrayList<CellAddr>();
796           int colStart;
797           int col nd;
798           int rowStart;
799           int row nd;
800           if (FindArea(dgvscc, out colStart, out col nd, out rowStart, out row nd, ref cellsAltered))
801           {
802               CellAddr newAddr = new CellAddr(colStart, rowStart);
803               sheet.PasteCell(cell, newAddr, col nd - colStart + 1, row nd - rowStart + 1);
804           }
805           else
806           {
807               foreach (DataGridViewCell dgvc in dgvscc)
808               {
809                   CellAddr newAddr = new CellAddr(dgvc.ColumnIndex, dgvc.RowIndex);
810                   sheet.PasteCell(cell, newAddr,
811                                   1, 1);
812               }
813           }
```

```
814             //sheet.MoveCell(cc.FromCellAddr.col, cc.FromCellAddr.row,
815             //               dgvc.ColumnIndex, dgvc.RowIndex);
816             sheet.workbook[cc.FromSheet][cc.FromCellAddr] = null;
817             RecomputeAndShow(cellsAltered.ToArray());
818           Console.WriteLine("Pasting moved cell");
819         } else if (Clipboard.ContainsText()) {
820           // Insert text (from external source)
821           CellAddr ca = new CellAddr(dgv.CurrentCellAddress);
822           String text = Clipboard.GetText();
823           Logger.Log.Debug(this, "Pasting text " + text);
824           SetCell(ca.col, ca.row, text);
825         }
826       }
827
828         private bool FindArea(DataGridViewSelectedCellCollection dgvscc, out int colStart, out int col nd,
829             out int rowStart, out int row nd, ref ArrayList<CellAddr> cellsAltered)
830         {
831             //TODO: Can make below faster by using some kind of sorting
832             if (dgvscc.Count != 0)
833             {
834                 colStart = col nd = dgvscc[0].ColumnIndex;
835                 rowStart = row nd = dgvscc[0].RowIndex;
836
837                 foreach (DataGridViewCell cell in dgvscc)
838                 {
839                     int col = cell.ColumnIndex;
840                     int row = cell.RowIndex;
841
842                     if(col<colStart)
843                         colStart = col;
844                     if(col>col nd)
845                         col nd = col;
846
847                     if(row<rowStart)
848                         rowStart = row;
849                     if(row>row nd)
850                         row nd = row;
851
852                     cellsAltered.Add(new CellAddr(col, row));
853                 }
854
855                 bool[,] area = new bool[col nd-colStart+1,row nd-rowStart+1];
856                 foreach (DataGridViewCell cell in dgvscc)
857                 {
858                     int col = cell.ColumnIndex;
859                     int row = cell.RowIndex;
860
861                     area[col - colStart, row - rowStart] = true;
862                 }
863
864                 foreach (bool b in area)
```

```
865                 {
866                     if(!b)
867                         return false;
868                 }
869
870                 return true;
871
872             }
873             colStart = col nd = rowStart = row nd = -1;
874             return false;
875         }
876
877         public void Delete() {
878           Stopwatch stopwatch = new Stopwatch();
879           stopwatch.Start();
880         DataGridViewSelectedCellCollection dgvscc = dgv.SelectedCells;
881         C5.LinkedList<CellAddr> deleteCells = new C5.LinkedList<CellAddr>();
882
883         foreach (DataGridViewCell dgvc in dgvscc)
884         {
885             CellAddr ca = new CellAddr(dgvc.ColumnIndex, dgvc.RowIndex);
886             deleteCells.Add(ca);
887             sheet.DeleteCell(ca);
888             sheet.workbook.Volatile xpressions.Remove(new FullCellAddr<Sheet>(ca.col, ca.row, sheet));
889             if (gui.ManualRecalculation)
890             {
891                 sheet.workbook.WaitingForManualUpdate.Add(new FullCellAddr<Sheet>(ca.col, ca.row, sheet));
892             }
893         }
894
895         RecomputeAndShow(deleteCells.ToArray());
896           stopwatch.Stop();
897           Logger.Log.Fatal(this, "Collective delete cells took: " + stopwatch. lapsed);
898     }
899
900     public void InsertRows(int N)
901     {
902       dgv.RowCount += N;
903       sheet.InsertRowCols(dgv.CurrentCellAddress.Y, N, true /* row */, gui.UseSupportGraph);
904       Reshow();
905     }
906
907     public void InsertColumns(int N)
908     {
909       dgv.ColumnCount += N;
910       sheet.InsertRowCols(dgv.CurrentCellAddress.X, N, false /* column */, gui.UseSupportGraph);
911       Reshow();
912     }
913
914     private void RecomputeAndShow(CellAddr[] addrs) {
915       sheet.workbook.Recalculate(this.sheet, addrs);
```

```
916       Reshow();
917     }
918
919     private void RecomputeAndShow(CellAddr addr)
920     {
921         RecomputeAndShow(new CellAddr[]{addr});
922 //        Reshow();
923     }
924
925     private void RecomputeAndShow()
926     {
927         sheet.workbook.RecalculateFull();
928         Reshow();
929     }
930
931     public String Reshow() {
932       sheet.ShowAll(delegate(int c, int r, String value) {
933         dgv[c, r].Value = value;
934       });
935       CellAddr ca = new CellAddr(dgv.CurrentCellAddress);
936       String text = null;
937       // FIXM : Not clear this is needed when distinguishing Value and FormattedValue
938       if (dgv.CurrentCell != null) {
939         text = sheet.Show(ca.col, ca.row);
940         dgv.CurrentCell.Value = text;
941       }
942       dgv.Focus();
943       return text;
944     }
945
946     public void ChangeCurrentText(String text) {
947       dgv.CurrentCell.Value = text;
948     }
949
950     public void SetCurrentCell(String text) {
951       CellAddr ca = new CellAddr(dgv.CurrentCellAddress);
952       SetCell(ca.col, ca.row, text);
953     }
954
955
956   }
957
958   // Package a copied cell and its cell address for holding in the Clipboard
959
960   [Serializable]
961   public class ClipboardCell {
962     public const String COPI D_C LL = "CopiedCell";
963     public const String CUT_C LL = "CutCell";
964     public readonly String FromSheet;
965     public readonly CellAddr FromCellAddr;
966
```

```
967     public ClipboardCell(String fromSheet, CellAddr fromCellAddr) {
968       this.FromSheet = fromSheet;
969       this.FromCellAddr = fromCellAddr;
970     }
971   }
972 }
973
```

CoreCalc.IO

WorkbookIO

```
// CoreCalc, a spreadsheet core implementation

// ------------------------------------------------------------------------
// Copyright (c) 2006 Thomas S. Iversen

// Permission is hereby granted, free of charge, to any person
// obtaining a copy of this software and associated documentation
// files (the "Software"), to deal in the Software without
// restriction, including without limitation the rights to use, copy,
// modify, merge, publish, distribute, sublicense, and/or sell copies
// of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:

//  * The above copyright notice and this permission notice shall be
//    included in all copies or substantial portions of the Software.

//  * The software is provided "as is", without warranty of any kind,
//    express or implied, including but not limited to the warranties of
//    merchantability, fitness for a particular purpose and
//    noninfringement.  In no event shall the authors or copyright
//    holders be liable for any claim, damages or other liability,
//    whether in an action of contract, tort or otherwise, arising from,
//    out of or in connection with the software or the use or other
//    dealings in the software.
// ------------------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Diagnostics;            // Stopwatch
using System.Globalization;          // CultureInfo
using System.IO;                           // MemoryStream, Stream
using System.Text;
using System.Xml;
using System.Windows.Forms;
using ICSharpCode.SharpZipLib.GZip;  // Gnumeric format is compressed

// TODO: Replace use of Scanner and Parser with Cell.Parse

// IOFormat is used to read XML files containing spreadsheets.
// Currently, Excel 2003 XMLSS format and Gnumeric format are supported.

namespace CoreCalc {
  abstract class IOFormat {
    public abstract Workbook Read(String filename);
    private String fileExtension;
    private String description;

    public Stream MakeStream(String s) {
      char[] cs = s.ToCharArray();
      byte[] bs = new byte[cs.Length];
      for (int i = 0; i < cs.Length; i++)
```

```
52          bs[i] = (byte)(cs[i]);
53        return new MemoryStream(bs);
54      }
55
56      public String GetFilter() {
57        return description + " (*." + fileExtension + ")|*." + fileExtension;
58      }
59
60      public IOFormat(String fileextension, String description) {
61        this.fileExtension = fileextension;
62        this.description = description;
63      }
64
65      public  oolean ValidExtension(String ext) {
66        return this.fileExtension == ext;
67      }
68    }
69
70    // ----------------------------------------
71
72    // Read and write Gnumeric format XML files
73
74    sealed class GnumericIOFormat : IOFormat {
75      public GnumericIOFormat() : base("gnumeric", "Gnumeric format") {
76      }
77
78      private  oolean isGnumeric(XmlDocument xmldoc) {
79        XmlElement docroot = xmldoc.DocumentElement;
80        return (docroot.Name == "gnm:Workbook");
81      }
82
83      private Workbook ParseGnumeric(XmlDocument xmldoc) {
84        Workbook wb = null;
85        Scanner scanner;
86        Parser parser;
87        Cell cell;
88
89        int i, row, col, valuetype, value, exprid, rows, cols;
90        String innertext;
91
92        // Is this a gnumeric spreadsheet?
93        if (isGnumeric(xmldoc))
94        {
95          wb = new Workbook();
96          XmlNodeList Xmlsheets = xmldoc.GetElements yTagName("gnm:Sheet");
97          XmlNamespaceManager xmlnsm = new XmlNamespaceManager(xmldoc.NameTable);
98          xmlnsm.AddNamespace("gnm", "http://www.gnumeric.org/v10.dtd");
99          XmlNode Xmlsheet, sheetnamenode, maxcolnode, maxrownode;
100         IDictionary<int, Expr> exprids = new Dictionary<int, Expr>();
101
102         String sheetname;
```

```
103         int maxcol, maxrow;
104         for (i = 0; i < Xmlsheets.Count; i++) {
105           Xmlsheet = Xmlsheets[i];
106           sheetnamenode = Xmlsheet.SelectSingleNode("gnm:Name", xmlnsm);
107           maxcolnode = Xmlsheet.SelectSingleNode("gnm:MaxCol", xmlnsm);
108           maxrownode = Xmlsheet.SelectSingleNode("gnm:MaxRow", xmlnsm);
109
110           if (sheetnamenode != null && maxcolnode != null && maxrownode != null) {
111             sheetname = sheetnamenode.InnerText;
112             try { maxcol = int.Parse(maxcolnode.InnerText); }
113             catch (Exception) { maxcol = 5; }
114             try { maxrow = int.Parse(maxrownode.InnerText); }
115             catch (Exception) { maxrow = 5; }
116
117             // maxrow and maxcol is encoded starting at 0 whereas our Sheet constructor
118             // below expects values starting at 1.
119             maxrow++;
120             maxcol++;
121
122             // Absence of meaningful maxrow and maxcol means that the sheet is
123             // there, but that the size is undefined. It defaults to 5x5.
124             if (maxrow <= 0)
125               maxrow = 5;
126             if (maxcol <= 0)
127               maxcol = 5;
128
129             // Add Sheet
130             new Sheet(wb, sheetname, maxcol, maxrow);
131           }
132           else {
133             // If we can't parse values, bail out.
134             return null;
135           }
136         }
137
138         // The sheets are in place; now fill in values.
139         // Iterate over sheets in wb instead of XmlNodeList sheets
140
141         i = 0;
142         foreach (Sheet sheet in wb) {
143           Xmlsheet = Xmlsheets[i];
144           XmlNodeList Xmlcells = Xmlsheet.SelectNodes("gnm:Cells/gnm:Cell", xmlnsm);
145
146           XmlNode Xmlcell;
147           XmlAttribute xmla;
148
149           // Iterate over all cell definitions in this Sheet.
150           for (int j = 0; j < Xmlcells.Count; j++) {
151             Xmlcell = Xmlcells[j];
152             XmlAttributeCollection xmlac = Xmlcell.Attributes;
153
```

```
154             row = col = exprid = valuetype = rows = cols = -1;
155
156             // Foreach cell, find its attributes.
157             for (int k = 0; k < xmlac.Count; k++) {
158               xmla = xmlac[k];
159
160               if (xmla != null) {
161                 try { value = int.Parse(xmla.Value); }
162                 catch { value = -1; }
163
164                 switch (xmla.Name) {
165                   case "Row":
166                     row = value;
167                     break;
168                   case "Col":
169                     col = value;
170                     break;
171                   case "ValueType":
172                     valuetype = value;
173                     break;
174                   case "ExprID":
175                     exprid = value;
176                     break;
177                   case "Rows":
178                     rows = value;
179                     break;
180                   case "Cols":
181                     cols = value;
182                     break;
183                   default:
184                     // Other Names do exist. For instance ValueFormat d/m/yy.
185                     //  ut they are not support as of now.
186                     break;
187                 }
188               }
189             }
190
191             if (row != -1 && col != -1) {
192               innertext = Xmlcell.InnerText;
193
194               // valuetype=60 => string
195               if (valuetype == 60) {
196                 innertext = "'" + innertext;
197               }
198
199               // If innertext is missing and exprid is set,
200               // It is a cached formula
201               if ((innertext == null | innertext.Equals("")) && exprid != -1) {
202                 cell = new Formula(wb, exprids[exprid]);
203               }
204               else {
```

```
205                 scanner = new Scanner(MakeStream(innertext));
206                 parser = new Parser(scanner);
207                 cell = parser.ParseCell(wb, col, row);
208
209                 Formula f = cell as Formula;
210                 if (exprid != -1 && f != null) {
211                   exprids[exprid] = f.Expr;
212                 }
213               }
214               if (rows != -1 || cols != -1) {
215                 // Area
216                 CellAddr c1 = new CellAddr(col, row);
217                 CellAddr c2 = new CellAddr(col + cols - 1, row + rows - 1);
218
219                 sheet.InsertMatrixFormula(cell, col, row, c1, c2);
220               }
221               else {
222                 sheet[col, row] = cell;
223               }
224             }
225           }
226           i++;
227         }
228       }
229       return wb;
230     }
231
232     public override Workbook Read(String filename) {
233       Workbook wb;
234       long membefore = System.GC.GetTotalMemory(true);
235
236       Stopwatch watch = new Stopwatch();
237       watch.Reset(); watch.Start();
238
239       Stream s = new GZipInputStream(File.OpenRead(filename));
240       XmlDocument xmldoc = new XmlDocument();
241       xmldoc.Load(s);
242       long memafter = System.GC.GetTotalMemory(true);
243       wb = ParseGnumeric(xmldoc);
244       watch.Stop();
245       Logger.Log.Debug(this, "Loaded gnumeric spreadsheet in {0} ms", watch.ElapsedMilliseconds.ToString());
246       Logger.Log.Debug(this, "Memory before: {0}\nMemory After: {1}\nMemory Consumed: {2}", membefore, memafter, memafter -
      membefore);
247       return wb;
248     }
249   }
250
251   // --------------------------------------------------
252
253   // Read and write XMLSS (Excel 2003 XML format) files
254
```

```
255   sealed class XMLSSIOFormat : IOFormat {
256
257     readonly Formats fo = new Formats();
258     IDictionary<String, Expr> expressionindex;
259
260     public XMLSSIOFormat()
261       : base("xml", "XMLSS") {
262       fo.RefFmt = Formats.RefType.R1C1;
263
264       // Used for finding identical formulas: same R1C1 representation
265       expressionindex = new Dictionary<String, Expr>();
266     }
267
268     private void ParseRow(XmlReader rowreader, Workbook wb, Sheet sheet, int row) {
269       /* XMLSS has origo at (1,1) = (A,1)
270        * CoreCalc internal representation has origo at (0,0) = (A,1)
271        */
272
273       int col = 0;
274       int colindex = 0;
275       XmlReader cellreader = rowreader.ReadSubtree();
276       Scanner scanner;
277       Parser parser;
278       while (cellreader.ReadToFollowing("Cell")) {
279         String colindexstr = cellreader.GetAttribute("ss:Index");
280         String arrayrangestr = cellreader.GetAttribute("ss:ArrayRange");
281         String formulastr = cellreader.GetAttribute("ss:Formula");
282         String typestr = "";
283         String dataval = "";
284
285         if (colindexstr != null) {
286           try { colindex = int.Parse(colindexstr); }
287           catch { colindex = 0; }
288           col = colindex;
289         }
290         else {
291           col++;
292         }
293
294         XmlReader datareader = cellreader.ReadSubtree();
295         if (datareader.ReadToFollowing("Data")) {
296           typestr = datareader.GetAttribute("ss:Type");
297           datareader.MoveToContent();
298           dataval = datareader.ReadElementContentAsString();
299
300         }
301
302         if (formulastr != null) {
303           scanner = new Scanner(MakeStream(formulastr));
304           //Logger.Log.Debug(this, "Trying to parse formula: {0}", formulastr);
305         }
```

```
306         else {
307           // Anything else than formulas are values.
308           // If XMLSS tells us it is a string we believe it to be so.
309
310           if (typestr == "String")
311             dataval = "'" + dataval;
312
313           scanner = new Scanner(MakeStream(dataval));
314         }
315
316         // If a Matrixresult occupies cells, do not overwrite
317         // the formula with precomputed and cached data from
318         // the XMLSS file. Instead skip the parsing and sheetupdate.
319         Cell existingcell = sheet[col - 1, row - 1];
320         if (existingcell != null)
321           continue;
322
323         parser = new Parser(scanner);
324         parser.SetNumberDecimalSeparator(".");
325         Cell cell = parser.ParseCell(wb, col, row);
326         //Cell cell = null;
327
328         if (arrayrangestr != null && cell is Formula) { // Matrix formula
329           string[] split = arrayrangestr.Split(":".ToCharArray());
330
331           RARef raref1 = new RARef(split[0]);
332           RARef raref2;
333           if (split.Length == 1) {
334             // FIXME: single cell result, but still matrix
335             raref2 = new RARef(split[0]);
336           }
337           else {
338             raref2 = new RARef(split[1]);
339           }
340
341           if (raref1 != null && raref2 != null) {
342             CellAddr celladdr1 = raref1.Addr(col - 1, row - 1);
343             CellAddr celladdr2 = raref2.Addr(col - 1, row - 1);
344
345             sheet.InsertMatrixFormula(cell, col - 1, row - 1, celladdr1, celladdr2);
346           }
347         }
348         else {   // One-cell formula, or constant
349           Formula f = cell as Formula;
350           if (f != null) {
351             String formulaexpression = f.Show(col, row, fo);
352             f.SetValue(sheet, col - 1, row - 1, Cell.ParseValue(dataval, typestr));
353             if (expressionindex.ContainsKey(formulaexpression))
354               f.Expr = expressionindex[formulaexpression];
355             else
356               expressionindex[formulaexpression] = f.Expr;
```

```
357           }
358           sheet[col - 1, row - 1] = cell;
359         }
360         datareader.Close();
361       }
362       cellreader.Close();
363     }
364
365
366
367     private void ParseSheet(XmlReader sheetreader, Workbook wb, Sheet sheet) {
368       int row;  // current row
369       if (sheetreader.ReadToFollowing("Table")) {
370         row = 0;
371         XmlReader rowreader = sheetreader.ReadSubtree();
372         while (rowreader.ReadToFollowing("Row")) {
373           String rowindexstr = rowreader.GetAttribute("ss:Index");
374           int rowindex;
375
376           if (rowindexstr != null) {
377             try { rowindex = int.Parse(rowindexstr); }
378             catch { rowindex = 0; }
379
380             row = rowindex;
381           }
382           else {
383             row++;
384           }
385           ParseRow(rowreader, wb, sheet, row);
386         }
387         rowreader.Close();
388       }
389     }
390
391     private void ParseSheets(XmlTextReader reader, Workbook wb) {
392       String sheetname;
393       Sheet sheet;
394       while (reader.ReadToFollowing("Worksheet")) {
395         sheetname = reader.GetAttribute("ss:Name");
396         XmlReader sheetreader = reader.ReadSubtree();
397         sheet = wb[sheetname];
398         ParseSheet(sheetreader, wb, sheet);
399         sheetreader.Close();
400       }
401     }
402
403     private  oolean isXMLSS(XmlTextReader reader) {
404       return reader.NodeType == XmlNodeType.Element &&
405              reader.Name == "Workbook" &&
406              reader.GetAttribute("xmlns") != null;
407     }
```

```
408
409     private void ParseWork ook(XmlTextReader reader, Workbook wb) {
410       while (reader.Read()) {
411         if (isXMLSS(reader)) {
412           ParseSheets(reader, wb);
413         }
414       }
415     }
416
417     private Workbook EmptyWorkbook(XmlTextReader reader) {
418       int cols = 0; // number of columns in sheet (parsed)
419       int rows = 0; // number of rows in sheet (parsed)
420       Workbook wb = null;
421       Sheet sheet = null;
422       String tmpstr;
423       while (reader.Read()) {
424         if (isXMLSS(reader)) {
425           wb = new Workbook();
426           String sheetname;
427           while (reader.ReadToFollowing("Worksheet")) {
428             sheetname = reader.GetAttribute("ss:Name");
429             XmlReader sheetreader = reader.ReadSubtree();
430
431             if (sheetreader.ReadToFollowing("Table")) {
432               // Assumption: ExpandedRowCound and ExpandedColumnCount exists
433               // EVEN THOUGH not required.
434
435               tmpstr = sheetreader.GetAttribute("ss:ExpandedRowCount");
436               try { rows = int.Parse(tmpstr); }
437               catch (Exception) { rows = 5; }
438               tmpstr = sheetreader.GetAttribute("ss:ExpandedColumnCount");
439               try { cols = int.Parse(tmpstr); }
440               catch (Exception) { cols = 5; }
441               sheet = new Sheet(wb, sheetname, cols, rows);
442             }
443             else {
444               // If no table exists, then the sheet does exist but is empty.
445               // FIXME, need to have app properties parsed onto here, so that we can use
446               // default userbased values instead of 5,5.
447               sheet = new Sheet(wb, sheetname, 5, 5);
448             }
449             sheetreader.Close();
450           }
451         }
452       }
453       return wb;
454     }
455
456     public override Workbook Read(String filename) {
457       XmlTextReader reader = null;
458       Workbook wb = null;
```

```
459       Stopwatch watch = new Stopwatch();
460
461       watch.Reset(); watch.Start();
462       try {
463         reader = new XmlTextReader(filename);
464         reader.WhitespaceHandling = WhitespaceHandling.None;
465         wb = EmptyWorkbook(reader);
466         if (reader != null)
467           reader.Close();
468         if (wb != null) {
469           reader = new XmlTextReader(filename);
470           reader.WhitespaceHandling = WhitespaceHandling.None;
471           try {
472             ParseWork ook(reader, wb);
473           }
474           catch ( adFormatException str) {
475             DialogResult result = Message ox.Show(" ad Format: " + str,
476                                   " ad Format",
477                                   Message ox uttons.OK,
478                                   Message oxIcon.Error);
479             return null;
480           }
481         }
482       }
483       finally {
484         if (reader != null)
485           reader.Close();
486       }
487       watch.Stop();
488       Logger.Log.Debug(this, "Loaded xmlss spreadsheet in {0} ms",
489                        watch.ElapsedMilliseconds.ToString());
490       return wb;
491     }
492   }
493
494   public class Work ookIO {
495     private List<IOFormat> formats;
496     private IOFormat defaultformat;
497     public Work ookIO() {
498       formats = new List<IOFormat>();
499       AddFormat(new XMLSSIOFormat(), true);
500       AddFormat(new GnumericIOFormat(), false);
501     }
502
503     private void AddFormat(IOFormat format, bool def) {
504       formats.Add(format);
505       if (def)
506         defaultformat = format;
507     }
508
509     private IOFormat FindFormat(String filename) {
```

```
510       String[] fields = filename.Split(("."").ToCharArray());
511       String ext = fields[fields.Length - 1];
512       foreach (IOFormat format in formats) {
513         if (format.ValidExtension(ext))
514           return format;
515       }
516       return null;
517     }
518
519     // Attempt to read a files using a supported format.
520     // The algorithm is as follows: First we try a supported format using
521     // a filename extension. If no method for that extension is found, or if the
522     // method found raises an exception, this method tries the supported formats
523     // one by one. If that does not succeed either, a null value is returned.
524
525     public Workbook Read(String filename) {
526       Workbook wb;
527       IOFormat format = FindFormat(filename);
528
529       // If we found one, try that format
530       if (format != null) {
531         wb = format.Read(filename);
532         if (wb != null)
533           return wb;
534       }
535
536       // Else try them all.
537       foreach (IOFormat format1 in formats) {
538         wb = format1.Read(filename);
539         if (wb != null)
540           return wb;
541       }
542
543       // give up.
544       return null;
545     }
546
547     public String SupportedFormatFilter() {
548       String uilder sb = new String uilder();
549       foreach (IOFormat ioformat in formats) {
550         sb.Append(ioformat.GetFilter());
551         sb.Append("|");
552       }
553
554       sb.Append("All files (*.*)|*.*");
555       return sb.ToString();
556     }
557
558     public int DefaultFormatIndex() {
559       return formats.IndexOf(defaultformat) + 1;
560     }
```

```
561     }
562 }
563
```

CoreCalc

Cells

```
 1 // CoreCalc, a spreadsheet core implementation
 2
 3 // ------------------------------------------------------------------------
 4 // Copyright (c) 2006 Peter Sestoft
 5
 6 // Permission is hereby granted, free of charge, to any person
 7 // obtaining a copy of this software and associated documentation
 8 // files (the "Software"), to deal in the Software without
 9 // restriction, including without limitation the rights to use, copy,
10 // modify, merge, publish, distribute, sublicense, and/or sell copies
11 // of the Software, and to permit persons to whom the Software is
12 // furnished to do so, subject to the following conditions:
13
14 //  * The above copyright notice and this permission notice shall be
15 //    included in all copies or substantial portions of the Software.
16
17 //  * The software is provided "as is", without warranty of any kind,
18 //    express or implied, including but not limited to the warranties of
19 //    merchantability, fitness for a particular purpose and
20 //    noninfringement.  In no event shall the authors or copyright
21 //    holders be liable for any claim, damages or other liability,
22 //    whether in an action of contract, tort or otherwise, arising from,
23 //    out of or in connection with the software or the use or other
24 //    dealings in the software.
25 // ------------------------------------------------------------------------
26
27 using System;
28 using System.Collections.Generic;
29 using System.Diagnostics;
30 using System.IO;
31 using C5;
32 using FAP;
33 // MemoryStream, Stream
34
35 // The Cell class and its subclasses represent the possible contents
36 // of a spreadsheet cell.
37
38 namespace CoreCalc {
39
40   // A Cell is what populates one cell of a Sheet (if anything)
41
42   public abstract class Cell {
43     public abstract void Eval(Sheet sheet, int col, int row);
44
45     public abstract Value GetValue(Sheet sheet, int col, int row);
46
47     public abstract void Apply(Sheet sheet, int col, int row, Act<Value> act);
48
49     public abstract Cell MoveContents(int deltaCol, int deltaRow);
50
51     public abstract void InsertRowCols(Dictionary<Expr, System.Collections.Generic.KeyValuePair<Adjusted<Expr>, bool>> adjusted,  ↙
```

```
    Sheet modSheet, bool thisSheet, int R, int N, int r, int c, bool doRows, HashDictionary<Expr, FAPList> currentExpression);

    public abstract void Reset();

    // Show computed value
    public String ShowValue(Sheet sheet, int col, int row) {
      Value v   GetValue(sheet, col, row);
      return v !  null   v.ToString() : "";
    }

    // Show constant or formula or matrix formula
    public abstract String Show(int col, int row, Formats fo);

    // Parse string to cell contents at (col, row) in given workbook
    public static Cell Parse(String text, Workbook workbook, int col, int row) {
      Scanner scanner   new Scanner(MakeStream(text));
      Parser parser   new Parser(scanner);
      Cell cell   parser.ParseCell(workbook, col, row);
      return cell;
    }

    private static Stream MakeStream(String s) {
      char[] cs   s.ToCharArray();
      byte[] bs   new byte[cs.Length];
      for (int i   0; i < cs.Length; i++)
        bs[i]   (byte)(cs[i]);
      return new MemoryStream(bs);
    }

      public static Value ParseValue(string value, string type)
      {
          try
          {
              switch (type)
              {
                  case "Number":
                      return new NumberValue(double.Parse(value.Replace('.', ',')));
                  case "String":
                      return new TextValue(value);
                  case "Boolean":
                      if (value    "1" || value    "TRUE")
                      {
                          return new NumberValue(1);
                      }
                      else if (value    "0" || value    "FALSE")
                      {
                          return new NumberValue(0);
                      }
                      return new ErrorValue("Error parsing boolean value from xml sheet");
                  case "Error":
                      return new ErrorValue(value);
```

```
102                 default:
103                     return new ErrorValue("Value type from xml sheet not implemented (only type \"Number\", \"String\" and \
    "Boolean\" are implemented)");
104             }
105         }
106         catch (Exception)
107         {
108             return new ErrorValue("Exception occured when parsing datatypes from xml file");
109         }
110     }
111 }
112
113 // ------------------------------------------------------------------
114 // A ConstCell is a cell that contains a constant only.
115 // In contrast to general cells, it is immutable and can be shared
116
117 abstract class ConstCell : Cell {
118
119
120     public override Cell MoveContents(int deltaCol, int deltaRow) {
121     return this;
122   }
123
124     public override void InsertRowCols(Dictionary<Expr, System.Collections.Generic.KeyValuePair<Adjusted<Expr>, bool>> adjusted,
    Sheet modSheet, bool thisSheet, int R, int N, int r, int c, bool doRows, HashDictionary<Expr, FAPList> currentExpression) { }
125
126   public override void Reset() { }
127 }
128
129 // ------------------------------------------------------------------
130 // A NumberCell is a cell containing a floating-point constant;
131 // not a constant number-valued expression
132
133 sealed class NumberCell : ConstCell {
134   public readonly NumberValue value;        // Non-null
135
136   public NumberCell(double d) {
137     this.value   new NumberValue(d);
138   }
139
140   public override void Eval(Sheet sheet, int col, int row) {
141
142   }
143
144   public override void Apply(Sheet sheet, int col, int row, Act<Value> act) {
145     act(value);
146   }
147
148   public override String Show(int col, int row, Formats fo) {
149     return value.value.ToString();
150   }
```

```
151
152       public override Value GetValue(Sheet sheet, int col, int row)
153       {
154           return value;
155       }
156   }
157
158   // ---------------------------------------------------------------
159   // A TextCell is a call containing a string constant;
160   // not a constant string-valued expression
161
162   sealed class TextCell : ConstCell {
163     public readonly TextValue value;          // Non-null
164
165     public TextCell(String s) {
166       this.value   new TextValue(s);
167     }
168
169     public override void Eval(Sheet sheet, int col, int row) {
170       }
171
172     public override void Apply(Sheet sheet, int col, int row, Act<Value> act) {
173       act(value);
174     }
175
176     public override String Show(int col, int row, Formats fo) {
177       return "'" + value.value;
178     }
179
180       public override Value GetValue(Sheet sheet, int col, int row)
181       {
182           return value;
183       }
184   }
185
186   // ---------------------------------------------------------------
187   // A Formula is a non-null caching expression contained in one cell.
188
189   sealed class Formula : Cell {
190     public readonly Workbook workbook;  // Non-null
191     private Expr e;                     // Non-null
192     private bool visited, uptodate;
193     private Value v;                  // Up to date if uptodate  workbook.Set
194
195     public Formula(Workbook workbook, Expr e) {
196       if (workbook    null || e    null)
197         throw new Exception("Formula arguments");
198       else {
199         this.workbook   workbook;
200         this.e   e;
201         this.visited   this.uptodate   workbook.Set;
```

```
202       }
203     }
204
205     public Formula(Formula f) : this(f.workbook, f.E) { }
206
207     // FIXME: Adequate for moving one cell, but block moves and row/column
208     // inserts should avoid the duplication and unsharing of expressions.
209     public override Cell MoveContents(int deltaCol, int deltaRow) {
210       return new Formula(workbook, E.Move(deltaCol, deltaRow));
211     }
212
213     // Evaluate expression if necessary, and cache its value
214     public override void Eval(Sheet sheet, int col, int row)
215     {
216         if (!sheet.workbook.UseSupportGraph)
217             EvalWithoutSupportGraph(col, row, sheet);
218         else
219             EvalWithSupportGraph(col, row, sheet);
220     }
221
222       private void EvalWithSupportGraph(int col, int row, Sheet sheet)
223       {
224           try
225           {
226               v   E.Eval(sheet, col, row);
227           }
228           catch (ErrorValueException exn)
229           {
230               v   exn.errorValue;
231           }
232       }
233
234       private void EvalWithoutSupportGraph(int col, int row, Sheet sheet)
235       {
236           if (uptodate !  workbook.Set)
237           {
238               if (visited    workbook.Set)
239                   throw new CyclicException("Cyclic cell reference: "
240                                             + Show(col, row, workbook.Format));
241
242               else
243               {
244                   visited   workbook.Set;
245                   //try
246                   //{
247                       v   E.Eval(sheet, col, row);
248                       uptodate   workbook.Set;
249                   //}
250                   //catch (ErrorValueException exn)
251                   //{
252                   //    v   exn.errorValue;
```

```
253                 //}
254             }
255         }
256     }
257
258
259     public override void Apply(Sheet sheet, int col, int row, Act<Value> act) {
260     act(GetValue(sheet, col, row));
261   }
262
263   public override void InsertRowCols(Dictionary<Expr, System.Collections.Generic.KeyValuePair<Adjusted<Expr>, bool>> adjusted,
      Sheet modSheet, bool thisSheet, int R, int N, int r, int c, bool doRows, HashDictionary<Expr, FAPList> currentExpression)
264   {
265       Adjusted<Expr> ae;
266       if (adjusted.ContainsKey(E) && (doRows   r : c) < adjusted[E].Key.upper)
267       {
268           // There is a valid cached adjusted expression
269           System.Collections.Generic.KeyValuePair<Adjusted<Expr>, bool> value   adjusted[E];
270           ae   value.Key;
271           if(value.Value)
272           {
273               int number   doRows   c : r;
274               FAPList current;
275               if(currentExpression.Find(ae.e, out current))
276                   current.Union(number, 1, 1);
277               else
278                   currentExpression.Add(ae.e, new FAPList(number, 1, 1, true));
279
280           }
281       }
282       else
283       {
284           bool coverRows   false;
285           // Compute a new adjusted expression and insert into the cache
286               ae   E.InsertRowCols(modSheet, thisSheet, R, N, doRows  r : c, doRows, ref coverRows);
287           Logger.Log.Debug(this, "Making new adjusted at rowcol " + r
288                           + "; upper   " + ae.upper);
289           if (ae.same)
290           { // For better sharing, reuse unadjusted e if same
291               ae   new Adjusted<Expr>(E, ae.upper, ae.same);
292               Logger.Log.Debug(this, "Reusing expression");
293           }
294           if (coverRows)
295           {
296               //This expression will have to generate a new
297               int number   doRows   c : r;
298               currentExpression.Add(ae.e, new FAPList(number, 1, 1, true));
299           }
300
301           adjusted[E]   new System.Collections.Generic.KeyValuePair<Adjusted<Expr>, bool>(ae, coverRows);
302       }
```

```
303         Debug.Assert((doRows   r : c) < ae.upper, "Formula.InsertRowCols");
304         e   ae.e;
305     }
306
307     // Reset recomputation flags after a circularity has been found
308     public override void Reset() {
309       visited   uptodate   workbook.Set;
310     }
311
312     public override String Show(int col, int row, Formats fo) {
313       return " " + E.Show(col, row, 0, fo);
314     }
315
316
317     public Expr Expr {
318       get { return E; }
319       set { e   value; }
320     }
321
322       public Expr E
323       {
324           get { return e; }
325       }
326
327       public override Value GetValue(Sheet sheet, int col, int row)
328       {
329           return v;
330       }
331
332       public void SetValue(Sheet sheet, int col, int row, Value value)
333       {
334           this.v   value;
335       }
336   }
337
338   // ------------------------------------------------------------------
339   // A MatrixFormula is a cached matrix formula shared among several
340   // cells, each cell accessing one part of the result matrix.  Several
341   // MatrixFormula cells share one CachedMatrixFormula cell; evaluation
342   // of one cell will evaluate the formula and cache its (matrix) value
343   // for the other cells to use.
344
345   sealed class MatrixFormula : Cell {
346     private readonly CachedMatrixFormula cmf;  // Non-null
347     private readonly CellAddr ca;  // Cell's location within matrix value
348     private MatrixValue matrix;
349
350     public MatrixFormula(CachedMatrixFormula cmf, CellAddr ca) {
351       this.cmf   cmf; this.ca   ca;
352     }
353
```

```
354     public MatrixFormula(CachedMatrixFormula cmf, int col, int row)
355       : this(cmf, new CellAddr(col, row)) { }
356
357     public override void Eval(Sheet sheet, int col, int row) {
358       matrix   cmf.Eval(sheet, col, row);
359
360     }
361
362     public override void Apply(Sheet sheet, int col, int row, Act<Value> act) {
363       cmf.Apply(sheet, col, row, act);
364     }
365
366     public bool Contains(int col, int row) {
367       return cmf.ulCa.col <  col && col <  cmf.lrCa.col
368           && cmf.ulCa.row <  row && row <  cmf.lrCa.row;
369     }
370
371     public override Cell MoveContents(int deltaCol, int deltaRow) {
372       // FIXME: loses sharing of the CachedMatrixFormula; but then again
373       // a matrix formula should never be moved cell by cell, but in its
374       // entirety, and in one go.
375       return new MatrixFormula(cmf.MoveContents(deltaCol, deltaRow), ca);
376     }
377
378       public override void InsertRowCols(Dictionary<Expr, System.Collections.Generic.KeyValuePair<Adjusted<Expr>, bool>> adjusted,↙
        Sheet modSheet, bool thisSheet, int R, int N, int r, int c, bool doRows, HashDictionary<Expr, FAPList> currentExpression)
379       {
380       // FIXME: Update underlying formula only once!
381     }
382
383     public override void Reset() {
384       cmf.Reset();
385     }
386
387     public override String Show(int col, int row, Formats fo) {
388       return "{" + cmf.Show(col, row, fo) + "}";
389     }
390
391       public override Value GetValue(Sheet sheet, int col, int row)
392       {
393           if (matrix !  null)
394               return matrix[ca];
395           else
396               return new ErrorValue("NONMATRIX");
397       }
398   }
399
400   // ------------------------------------------------------------------
401   // A cached matrix formula is shared between multiple matrix cells.
402   // It contains a (caching) matrix-valued formula, the original cell
403   // address of that formula, and the upper left and lower right corners
```

```
404   // of the matrix.
405
406   sealed class CachedMatrixFormula {
407     public readonly Formula formula;             // Non-null
408     private readonly int formulaCol, formulaRow; // Location of formula entry
409     public readonly CellAddr ulCa, lrCa;         // Corners of matrix formula
410     // Invariant: Every cell within sheet[ulCa, lrCa] is a MatrixFormula
411     // whose CachedMatrixFormula instance is this one.
412
413     public CachedMatrixFormula(Formula formula, int formulaCol, int formulaRow,
414                                CellAddr ulCa, CellAddr lrCa) {
415       if (formula    null)
416         throw new Exception("CachedMatrixFormula arguments");
417       else {
418         this.formula   formula;
419         this.formulaCol   formulaCol;
420         this.formulaRow   formulaRow;
421         this.ulCa   ulCa;
422         this.lrCa   lrCa;
423       }
424     }
425
426     // Evaluate expression if necessary, cast, and return
427     public MatrixValue Eval(Sheet sheet, int col, int row) {
428       return formula.GetValue(sheet, formulaCol, formulaRow) as MatrixValue;
429     }
430
431     public void Apply(Sheet sheet, int col, int row, Act<Value> act) {
432       formula.Apply(sheet, formulaCol, formulaRow, act);
433     }
434
435     public CachedMatrixFormula MoveContents(int deltaCol, int deltaRow) {
436       // FIXME: Unshares the formula, shouldn't ...
437       return new CachedMatrixFormula((Formula)formula.MoveContents(deltaCol, deltaRow),
438                                      formulaCol, formulaRow, ulCa, lrCa);
439     }
440
441     // Reset recomputation flags after a circularity has been found
442     public void Reset() {
443       formula.Reset();
444     }
445
446     public String Show(int col, int row, Formats fo) {
447       return formula.Show(col, row, fo);
448     }
449   }
450 }
451
```

CoreCalc

Functions

```
// CoreCalc, a spreadsheet core implementation

// ------------------------------------------------------------------------
// Copyright (c) 2006 Peter Sestoft and Thomas Iversen

// Permission is hereby granted, free of charge, to any person
// obtaining a copy of this software and associated documentation
// files (the "Software"), to deal in the Software without
// restriction, including without limitation the rights to use, copy,
// modify, merge, publish, distribute, sublicense, and/or sell copies
// of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:

//  * The above copyright notice and this permission notice shall be
//    included in all copies or substantial portions of the Software.

//  * The software is provided "as is", without warranty of any kind,
//    express or implied, including but not limited to the warranties of
//    merchantability, fitness for a particular purpose and
//    noninfringement.  In no event shall the authors or copyright
//    holders be liable for any claim, damages or other liability,
//    whether in an action of contract, tort or otherwise, arising from,
//    out of or in connection with the software or the use or other
//    dealings in the software.
// ------------------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Text;
using CoreCalc.SupportGraf;

// Machinery to define built-in functions and operators

namespace CoreCalc {
  class Function {
      private const int NO_VALUE = -1;

    public readonly String name;
    public readonly Applier applier;
    public readonly int fixity; // Non-zero: precedence of operator
    private static readonly IDictionary String, Function> table;
    private static readonly Random rnd = new Random();
    private static readonly long basedate = new DateTime(1899, 12, 30).Ticks;

    // Get a Function by name
    public static Function Get(String name)
    {
        if (!table.ContainsKey(name.ToUpper()))
        {
            Logger.Log.Info(typeof(Function), "Function does not exist: " + name.ToUpper());
            return table["PI"];
```

```
52          }
53        return table[name.ToUpper()];
54      }
55      private enum criteriaType {largerThan, largerThanOrEquals, smallerThan, smallerThanOrEquals, equals, notEquals, }
56
57      static Function() {
58        table = new Dictionary String, Function>();
59        //  fun> : unit -> number
60        new Function("NOW",
61                     MakeFunction(delegate() {
62                       return (DateTime.Now.Ticks - basedate) * 100E-9 / 60 / 60 / 24;
63        }));
64        new Function("PI",
65                     MakeConstant(3.14159265358979));
66        new Function("RAND",
67                     MakeFunction(rnd.NextDouble));
68        //  fun> : number -> number
69        new Function("SIN",
70            MakeFunction(Math.Sin));
71        new Function("COS",
72            MakeFunction(Math.Cos));
73        new Function("TAN",
74            MakeFunction(Math.Tan));
75        new Function("LN",
76            MakeFunction((Fun double,double>)Math.Log));
77        new Function("LOG10",
78            MakeFunction(Math.Log10));
79        new Function("EXP",
80            MakeFunction(Math.Exp));
81        new Function("SQRT",
82            MakeFunction(Math.Sqrt));
83        //  fun> : number * number -> number
84        new Function("CEILING",
85            MakeFunction(delegate(double d, double signif) {
86              return signif * Math.Ceiling(d / signif);
87            }));
88        new Function("FLOOR",
89            MakeFunction(delegate(double d, double signif) {
90              return signif * Math.Floor(d / signif);
91            }));
92        new Function("ROUND",
93            MakeFunction(delegate(double d, double digits) {
94              return Math.Round(d, (int)digits);
95            }));
96        new Function("^", 8,
97            MakeFunction(Math.Pow));
98        new Function("*", 7,
99            MakeFunction(delegate(double x, double y) { return x * y; }));
100       new Function("/", 7,
101           MakeFunction(delegate(double x, double y) { return x / y; }));
102       new Function("+", 6,
```

```
103             MakeFunction(delegate(double x, double y) { return x + y; }));
104         new Function("-", 6, delegate(Sheet sheet, Expr[] es, int col, int row)
105                                 {
106                                     if (es.Length == 2 || es.Length == 1)
107                                     {
108                                         NumberValue v0 = es[0].Eval(sheet, col, row) as NumberValue;
109                                         if (v0 == null)
110                                             return ArgTypeErr();
111 
112                                         if(es.Length == 1)
113                                         {
114                                             return new NumberValue(-v0.value);
115                                         }
116                                         else
117                                         {
118                                             NumberValue v1 = es[1].Eval(sheet, col, row) as NumberValue;
119                                             if(v1 == null)
120                                                 return new NumberValue(v0.value);
121 
122                                             return new NumberValue(v0.value - v1.value);
123                                         }
124                                     }
125                                     else
126                                         return ArgCountErr();
127                                 });
128 
129 
130 
131         new Function("&", 7,
132             MakeFunction(delegate(String x, String y) { return x + y; }));
133         new Function(" ", 5,
134             MakePredicate(delegate(double x, double y) { return x   y; }));
135         new Function(" =", 5,
136             MakePredicate(delegate(double x, double y) { return x  = y; }));
137         new Function(">=", 5,
138             MakePredicate(delegate(double x, double y) { return x >= y; }));
139         new Function(">", 5,
140             MakePredicate(delegate(double x, double y) { return x > y; }));
141         new Function("==", 4,
142             delegate(Sheet sheet, Expr[] es, int col, int row)
143                 {
144                     return Equals(sheet, es, col, row);
145                 });
146         new Function(" >", 4,
147             MakePredicate(delegate(double x, double y) { return x != y; }));
148         //  SUM : { number, matrix } * -> number
149         new Function("SUM",
150             MakeFunction(delegate(Value[] vs) {
151               double sum = 0.0;
152               Apply(vs, delegate(double x) { sum += x; });
153               return sum;
```

```
154           }));
155       new Function("FASTSUM",
156           delegate(Sheet sheet, Expr[] es, int col, int row) {
157             double sum = 0.0;
158             foreach (Expr e in es) {
159               e.Apply(sheet, col, row,
160                       delegate(Value v) {
161                         NumberValue number = v as NumberValue;
162                         if (number != null)
163                           sum += number.value;
164                         //else do nothing
165                           //throw new ArgumentException();
166                       });
167             }
168             return new NumberValue(sum);
169           });
170       //  AVG : (number | matrix)*  -> number
171       new Function("AVG",
172           MakeFunction(delegate(Value[] vs) {
173             double sum = 0.0;
174             int count = 0;
175             Apply(vs, delegate(double x) { sum += x; count++; });
176             return sum / count;
177           }));
178       // TRANSPOSE : matrix -> matrix
179       new Function("TRANSPOSE",
180           MakeFunction(delegate(Value v0) {
181             MatrixValue v0mat = v0 as MatrixValue;
182             if (v0mat != null) {
183               int cols = v0mat.Rows, rows = v0mat.Cols;
184               Value[,] res = new Value[cols, rows];
185               for (int c = 0; c   cols; c++)
186                 for (int r = 0; r   rows; r++)
187                   res[c, r] = v0mat[r, c];
188               return new MatrixValue(res);
189             } else
190               return ArgTypeErr();
191           }));
192       // COLS : matrix -> number  // Too strict?  If arg is CellArea
193       new Function("COLS",
194           MakeFunction(delegate(Value v0) {
195             MatrixValue v0mat = v0 as MatrixValue;
196             if (v0mat != null)
197               return new NumberValue(v0mat.Cols);
198             else
199               return ArgTypeErr();
200           }));
201       // ROWS : matrix -> number  // Too strict?  If arg is CellArea
202       new Function("ROWS",
203           MakeFunction(delegate(Value v0) {
204             MatrixValue v0mat = v0 as MatrixValue;
```

```
205             if (v0mat != null)
206               return new NumberValue(v0mat.Rows);
207             else
208               return ArgTypeErr();
209         }));
210       // ISMATRIX : matrix -> number  // Too strict?  If arg is CellArea
211       new Function("ISMATRIX",
212           MakeFunction(delegate(Value v0) {
213             if (v0 != null)
214               return new NumberValue(v0 is MatrixValue ? 1.0 : 0.0);
215             else
216               return new ErrorValue("ARGNULL");
217         }));
218       // IF : number any any -> any,
219       new Function("IF",                    // Note: non-strict in arg 2 and 3
220           delegate(Sheet sheet, Expr[] es, int col, int row) {
221             if (es.Length == 3 || es.Length == 2) {
222               NumberValue v0 = es[0].Eval(sheet, col, row) as NumberValue;
223               if (v0 != null)
224               {
225                   if (v0.value != 0)
226                       return es[1].Eval(sheet, col, row);
227                   else if (es.Length == 3)
228                       return es[2].Eval(sheet, col, row);
229                   else
230                       return new NumberValue(0.0);
231               }
232               else
233                 return ArgTypeErr();
234             }
235             else
236               return ArgCountErr();
237         });
238       // CHOOSE number * any* -> any
239       new Function("CHOOSE",                // Note: non-strict in arg 2...
240           delegate(Sheet sheet, Expr[] es, int col, int row) {
241             if (es.Length >= 1) {
242               NumberValue v0 = es[0].Eval(sheet, col, row) as NumberValue;
243               if (v0 != null && 1  = v0.value && v0.value   es.Length)
244                 return es[(int)(v0.value)].Eval(sheet, col, row);
245               else
246                 return ArgTypeErr();
247             }
248             else
249               return ArgCountErr();
250         });
251       new Function("MAX",
252                    MakeFunction(delegate(Value[] vs)
253                                 {
254                                     double highest = 0.0;
255
```

```
256                                     Apply(vs, delegate(double x)
257                                                 {
258                                                     if(x>highest)
259                                                         highest = x;
260                                                 }
261                                         );
262                                     return highest;
263                                 }
264                     )
265         );
266     new Function("MIN",
267                 MakeFunction(delegate(Value[] vs)
268                                 {
269                                     double lowest = int.MaxValue;
270                                     bool foundOne = false;
271 
272                                     Apply(vs, delegate (double x)
273                                                 {
274                                                     foundOne = true;
275                                                     if (x   lowest) lowest = x;
276                                                 });
277                                     if (foundOne)
278                                         return lowest;
279                                     else
280                                         return 0.0;
281                                 }
282                     )
283         );
284       new Function("MID",
285           delegate(Sheet sheet, Expr[] es, int col, int row) {
286           if (es.Length == 3) {
287             TextValue v0 = es[0].Eval(sheet, col, row) as TextValue;
288             NumberValue start = es[1].Eval(sheet, col, row) as NumberValue;
289             NumberValue numb = es[2].Eval(sheet, col, row) as NumberValue;
290             int startValue = ((int) start.value)-1;
291             int numbValue = (int) numb.value;
292             if (v0 != null && start != null && numb != null)
293             {
294                 if (startValue >= v0.value.Length) return new TextValue("");
295                 else if (numbValue + startValue > v0.value.Length) numbValue = v0.value.Length-startValue;
296                 return new TextValue(v0.value.Substring(startValue, numbValue));
297             }
298             else
299               return ArgTypeErr();
300           }
301           else
302             return ArgCountErr();
303         });
304       new Function("LEFT",
305           delegate(Sheet sheet, Expr[] es, int col, int row)
306           {
```

```
307                 if (es.Length == 2)
308                 {
309                     TextValue v0 = es[0].Eval(sheet, col, row) as TextValue;
310                     NumberValue numb = es[1].Eval(sheet, col, row) as NumberValue;
311                     if (v0 != null && numb != null)
312                     {
313                         if(numb.value  = v0.value.Length)
314                             return new TextValue(v0.value.Substring(0, (int)numb.value));
315                         else
316                             return new ErrorValue("ERROR IN FORMULA");
317                     }
318                     else
319                         return ArgTypeErr();
320                 }
321                 else
322                     return ArgCountErr();
323             });
324         new Function("RIGHT",
325              delegate(Sheet sheet, Expr[] es, int col, int row)
326             {
327                 if (es.Length == 2)
328                 {
329                     TextValue v0 = es[0].Eval(sheet, col, row) as TextValue;
330                     NumberValue numb = es[1].Eval(sheet, col, row) as NumberValue;
331                     if (v0 != null && numb != null)
332                     {
333                         int lastCh = v0.value.Length - 1;
334                         if (numb.value  = lastCh)
335                         {
336                             int startAt = lastCh - (int)numb.value;
337                             return new TextValue(v0.value.Substring(startAt, (int)numb.value));
338                         }
339                         else
340                             return new TextValue(v0.value);
341                     }
342                     else
343                         return ArgTypeErr();
344                 }
345                 else
346                     return ArgCountErr();
347             });
348 
349         new Function("OR", delegate(Sheet sheet, Expr[] es, int col, int row)
350                     {
351                      foreach (Expr e in es)
352                                     {
353                                         NumberValue nVal = e.Eval(sheet, col, row) as NumberValue;
354                                         if(nVal != null && nVal.value == 1.0)
355                                         {
356                                             return new NumberValue(1.0);
357                                         }
```

```
358                                 }
359                                 return new NumberValue(0.0);
360                             });
361         new Function("AND", delegate(Sheet sheet, Expr[] es, int col, int row)
362                             {
363                                 foreach (Expr e in es)
364                                 {
365                                     NumberValue nVal = e.Eval(sheet, col, row) as NumberValue;
366                                     if(nVal == null || nVal.value != 1.0)
367                                     {
368                                         return new NumberValue(0.0);
369                                     }
370                                 }
371                                 return new NumberValue(1.0);
372                             });
373 
374         new Function("TRUNC", delegate(Sheet sheet, Expr[] es, int col, int row)
375                               {
376                                   NumberValue numb = es[0].Eval(sheet, col, row) as NumberValue;
377 
378 
379                                   NumberValue tr;
380                                   if (es.Length == 2)
381                                       tr = es[1].Eval(sheet, col, row) as NumberValue;
382                                   else if (es.Length == 1)
383                                       tr = new NumberValue(0.0);
384                                   else
385                                       return ArgCountErr();
386                                   if (numb != null && tr != null)
387                                   {
388 
389                                       double val = numb.value;
390                                       int tenPowTr = (int)(Math.Pow(10.0, Math.Floor(tr.value)));
391                                       val = val * tenPowTr;
392                                       val = Math.Floor(val);
393                                       val = val / tenPowTr;
394 
395                                       return new NumberValue(val);
396                                   }
397                                   else
398                                       return ArgTypeErr();
399                               });
400         new Function("HLOOKUP", MakeLookupFunction(false));
401         new Function("VLOOKUP", MakeLookupFunction(true));
402         new Function("LEN", delegate(Sheet sheet, Expr[] es, int col, int row)
403                             {
404                                 if (es.Length == 1)
405                                 {
406                                     TextValue value = es[0].Eval(sheet, col, row) as TextValue;
407                                     if (value != null)
408                                     {
```

```
409                                             return new NumberValue(value.value.Length);
410                                         }
411                                         else return new NumberValue(0.0);
412                                     }
413                                     return ArgCountErr();
414                                 });
415         new Function("TRUE", MakeConstant(1.0));
416         new Function("FALSE", MakeConstant(0.0));
417         new Function("NOT", MakeFunction(delegate(double boolVal)
418                                             {
419                                                 return boolVal == 0.0 ? 1.0 : 0.0;
420                                             }));
421
422         new Function("OFFSET", delegate(Sheet sheet, Expr[] es, int col, int row)
423                                     {
424                                         if(es.Length >= 3 && es.Length  =5)
425                                         {
426                                             int width = NO_VALUE;
427                                             int height = NO_VALUE;
428
429                                             NumberValue rowOffSetNumbValue = es[1].Eval(sheet, col, row) as NumberValue;
430                                             NumberValue colOffsetNumbValue = es[2].Eval(sheet, col, row) as NumberValue;
431
432
433                                             if(es.Length >= 4)
434                                             {
435                                                 NumberValue heightVal = es[3].Eval(sheet, col, row) as NumberValue;
436                                                 if (heightVal != null)
437                                                     height = (int)heightVal.value;
438                                                 else
439                                                     return ArgTypeErr();
440
441                                             }
442
443                                             if(es.Length == 5)
444                                             {
445                                                 NumberValue widthVal = es[4].Eval(sheet, col, row) as NumberValue;
446                                                 if (widthVal != null)
447                                                     width = (int)widthVal.value;
448                                                 else
449                                                     return ArgTypeErr();
450                                             }
451
452                                             if (rowOffSetNumbValue != null && colOffsetNumbValue != null && (es[0] is CellRef ||  ↙
    es[0] is CellArea))
453                                             {
454                                                 int rowOffSet = (int)rowOffSetNumbValue.value;
455                                                 int colOffSet = (int)colOffsetNumbValue.value;
456
457                                                 CellRef reference;
458                                                 if (es[0] is CellRef)
```

```
459                                             {
460                                                  reference = es[0] as CellRef;
461                                                  if((width == NO_VALUE || width == 1) && (width == NO_VALUE || height == 1))
462                                                  {
463                                                      width = 1;
464                                                      height = 1;
465                                                  }
466
467                                             }
468                                             else
469                                             {
470                                                 CellArea cArea = es[0] as CellArea;
471                                                 Area area = new Area(cArea, sheet, new CellAddr(col, row));
472                                                 if(width == NO_VALUE)
473                                                  width = area.colEnd - area.colStart;
474                                                 if(height == NO_VALUE)
475                                                  height = area.rowEnd - area.rowStart;
476                                                 reference = new CellRef(cArea.sheet, cArea.ul);
477                                             }
478
479                                             RARef addr = reference.raref;
480
481                                             int colVal = addr.colRef + colOffSet;
482                                             int rowVal = addr.rowRef + rowOffSet;
483
484                                             Sheet actualSheet = reference.sheet == null ? sheet : reference.sheet;
485
486                                             RARef upperLeft = new RARef(addr.colAbs, colVal, addr.rowAbs, rowVal);
487                                             if(width == 1 && height == 1)
488                                                 return new CellRef(reference.sheet, upperLeft).Eval(sheet, col, row);
489                                             else
490                                             {
491                                                 RARef lowerRight = new RARef(addr.colAbs, colVal+width, addr.rowAbs, rowVal+
    height);
492                                                 return new CellArea(reference.sheet, upperLeft, lowerRight).Eval(sheet, col,
    row);
493                                             }
494
495                                         }
496                                         else return ArgTypeErr();
497
498                                     }
499                                     else
500                                         return ArgCountErr();
501                                 });
502
503         new Function("CONCATENATE", MakeFunction(delegate(Value[] vs) {
504             StringBuilder builder = new StringBuilder();
505             Apply(vs, delegate(string x) { builder.Append(x); });
506             return builder.ToString();
507           }));
```

```
508
509         new Function("LOOKUP", delegate(Sheet sheet, Expr[] es, int col, int row)
510                                     {
511                                         if(es.Length == 2 || es.Length == 3)
512                                         {
513                                             Value searchValue = es[0].Eval(sheet, col, row);
514                                             MatrixValue lookup;
515                                             MatrixValue values;
516
517                                             try
518                                             {
519                                                 lookup = GetMatrixValue(sheet, es[1], col, row);
520                                             }
521                                             catch(ArgumentException)
522                                             {
523                                                 return ArgTypeErr();
524                                             }
525
526
527                                             if (es.Length == 2)
528                                                 values = lookup;
529                                             else
530                                             {
531                                                 try
532                                                 {
533                                                     values = GetMatrixValue(sheet, es[2], col, row);
534                                                 }
535                                                 catch(ArgumentException)
536                                                 {
537                                                     return ArgTypeErr();
538                                                 }
539                                             }
540
541                                             Value currentValue = null;
542                                             for (int c = 0; c   lookup.Cols; c++)
543                                             {
544                                                 for (int r = 0; r   lookup.Rows; r++)
545                                                 {
546
547                                                     if (lookup[c, r] == null)
548                                                     {
549                                                         if (searchValue == null)
550                                                             return new NullValue();
551                                                     }
552                                                     else
553                                                     {
554                                                         CompareResult result = lookup[c, r].CompareToOther(searchValue);
555
556                                                         if (result == CompareResult.ThisIsSmallest || result == CompareResult.     ↙
    Equal)
557                                                         {
```

```
                                                    if (c   values.Cols && r   values.Rows)
                                                    {
                                                        currentValue = values[c, r];
                                                    }
                                                    else
                                                        return ArgNotAvail();
                                                }

                                                if (result == CompareResult.OtherIsSmallest || result == CompareResult.
Equal)
                                                {
                                                    if (currentValue != null)
                                                        return currentValue;
                                                    else
                                                        return ArgNotAvail();
                                                }
                                            }
                                        }
                                    }
                                    //we would have quitted the loop if we found something suitable
                                    //TODO: find out if we should return currentvalue if it's not null
                                    return ArgNotAvail();


                                }
                                return ArgCountErr();

                            });

        new Function("COUNTIF", delegate(Sheet sheet, Expr[] es, int col, int row)
                                {
                                    if (es.Length == 2)
                                    {
                                        MatrixValue values;
                                        try
                                        {
                                            values = GetMatrixValue(sheet, es[0], col, row);
                                        }
                                        catch (ArgumentException)
                                        {
                                            return ArgTypeErr();
                                        }

                                        Value value = es[1].Eval(sheet, col, row);
                                        Value refValue;
                                        criteriaType criteria;
                                        if (value is NumberValue)
                                        {
                                            refValue = (value as NumberValue);
                                            criteria = criteriaType.equals;
                                        }
```

```
                                    else if (value is TextValue)
                                    {
                                        string text = (value as TextValue).value;
                                        GetCriteriaAndNumber(text, out refValue, out criteria);

                                    }
                                    else
                                        return ArgTypeErr();

                                    double returnVal = 0.0;
                                    foreach (Value val in values)
                                    {
                                        if(CriteriaHolds(val, refValue, criteria))
                                                returnVal++;

                                    }
                                    return new NumberValue(returnVal);

                                }
                                else return ArgCountErr();
                            });


        //new Function("SUMIF", delegate(Sheet sheet, Expr[] es, int col, int row)
        //                        {
        //                            if (es.Length == 2 || es.Length == 3)
        //                            {
        //                                Value rangeValue = es[0].Eval(sheet, col, row);
        //                                MatrixValue range;
        //                                range = GetMatrixValue(rangeValue);

        //                                Value criteria = es[1].Eval(sheet, col, row);
        //                                MatrixValue sumRange;

        //                                if(es.Length == 2)
        //                                    sumRange = range;
        //                                else
        //                                    sumRange = GetMatrixValue(es[2].Eval(sheet, col, row));

        //                                if(criteria is TextValue || criteria is NumberValue)
        //                                {
        //                                    TextValue textCriteria = criteria as TextValue;
        //                                    NumberValue numberCriteria = criteria as NumberValue;
        //                                    if(textCriteria != null)
        //                                    {
        //                                        string cr = criteria.value;
        //                                        criteriaType type;
        //                                        if (cr.StartsWith(" "))
        //                                            type = criteriaType.smallerThan;
        //                                        else if (cr.StartsWith(">"))
```

```
659         //                                          type = criteriaType.largerThan;
660         //                                      else if (cr.StartsWith("="))// || cr.StartsWith("=="))
661         //                                          type = criteriaType.equals;
662         //                                      else if (cr.StartsWith(" >"))
663         //                                          type = criteriaType.notEquals;
664         //                                      else
665         //                                          return ArgTypeErr();
666 
667         //                                      if (type == criteriaType.notEquals || (type == criteriaType.equals && cr.StartsWith("==")))
668         //                                          cr = cr.Remove(2);
669         //                                      else
670         //                                          cr = cr.Remove(1);
671 
672         //                                      double number;
673         //                                      try
674         //                                      {
675         //                                          number = Convert.ToDouble(cr);
676         //                                      }
677         //                                      catch(InvalidCastException e)
678         //                                      {
679         //                                          Logger.Log.Info(typeof(Function), "The criteria for function ISSUM was not correct format" +
680         //                                              "the criteria was {0}. The exception was {1}", new object[]{textCriteria.value, e});
681         //                                          return ArgTypeErr();
682         //                                      }
683 
684         //                                  }
685         //                                  else if(numberCriteria != null)
686         //                                  {}
687         //                              }
688         //                              else return ArgTypeErr();
689         //                          }
690         //                          ArgCountErr()
691         //                      });
692 
693         //METHOD FOR NEGATING (ex. =-A1) (could do this via the "-" function by looking at number of expressions but the "-" looks so nice right now :) )
694         new Function("ABS", MakeFunction(delegate(double input)
695                                          {
696                                              return Math.Abs(input);
697                                          }));
698 
699 
700 
701         //DUMMY METHODS: These methods are not correct, they just represent the real methods so tests can be run
702 
703         new Function("PROBABILITYRANGE", delegate(Sheet sheet, Expr[] es, int col, int row)
704                                          {
705                                              if(es.Length >= 2)
```

```
706                                                 {
707                                                     Value value = es[1].Eval(sheet, col, row);
708                                                     if(value is MatrixValue)
709                                                     {
710                                                         MatrixValue mValue = value as MatrixValue;
711                                                         Random random = new Random();
712                                                         int c = random.Next(0, mValue.Cols);
713                                                         int r = random.Next(0, mValue.Rows);
714                                                         return mValue[c, r];
715                                                     }
716
717                                                     return value;
718                                                 }
719                                                 return ArgCountErr();
720                                             }
721             );
722
723
724     }
725
726       private static Value Equals(Sheet sheet, Expr[] es, int col, int row)
727       {
728           if (es.Length == 2)
729           {
730               Value v0 = es[0].Eval(sheet, col, row);
731               Value v1 = es[1].Eval(sheet, col, row);
732               if (v0 != null)
733               {
734                   bool equals = v0.CompareToOther(v1) == CompareResult.Equal;
735                   return new NumberValue(equals ? 1.0 : 0.0);
736               }
737               else if (v1 != null)
738               {
739                   bool equals = v1.CompareToOther(v0) == CompareResult.Equal;
740                   return new NumberValue(equals ? 1.0 : 0.0);
741               }
742               else
743                 return new NumberValue(0.0);
744           }
745           else
746               return ArgCountErr();
747       }
748
749       private static bool CriteriaHolds(Value value, Value referenceValue, criteriaType criteria)
750       {
751           if(value == null)
752               return referenceValue == null || referenceValue.CompareToOther(value) == CompareResult.Equal;
753
754           CompareResult result = value.CompareToOther(referenceValue);
755
756           if (criteria == criteriaType.equals)
```

```
757             return result == CompareResult.Equal;
758         else if (criteria == criteriaType.largerThan)
759             return result == CompareResult.OtherIsSmallest;
760         else if (criteria == criteriaType.largerThanOrEquals)
761             return result == CompareResult.OtherIsSmallest || result == CompareResult.Equal;
762         else if (criteria == criteriaType.notEquals)
763             return result == CompareResult.DifferentOrIncompatibleTypes;
764         else if (criteria == criteriaType.smallerThan)
765             return result == CompareResult.ThisIsSmallest;
766         else if (criteria == criteriaType.smallerThanOrEquals)
767             return result == CompareResult.ThisIsSmallest || result == CompareResult.Equal;
768         else
769             throw new ArgumentException("The criteria has an unexpected value. Criteria was: " + criteria);
770     }
771     private static void GetCriteriaAndNumber(string cr, out Value compareVal, out criteriaType criteria)
772     {
773
774         if (cr.StartsWith(" "))
775             criteria = criteriaType.smallerThan;
776         else if (cr.StartsWith(" ="))
777             criteria = criteriaType.smallerThanOrEquals;
778         else if (cr.StartsWith(">"))
779             criteria = criteriaType.largerThan;
780         else if (cr.StartsWith(">="))
781             criteria = criteriaType.largerThanOrEquals;
782         else if (cr.StartsWith("="))// || cr.StartsWith("=="))
783             criteria = criteriaType.equals;
784         else if (cr.StartsWith(" >"))
785             criteria = criteriaType.notEquals;
786         else
787         {
788             criteria = criteriaType.equals;
789             compareVal = new TextValue(cr);
790             return;
791         }
792
793         if (criteria == criteriaType.notEquals || criteria == criteriaType.smallerThanOrEquals || criteria == criteriaType.     ↙
    largerThanOrEquals
794             || (criteria == criteriaType.equals && cr.StartsWith("==")))
795             cr = cr.Remove(2);
796         else
797             cr = cr.Remove(1);
798
799         try
800         {
801             compareVal = new NumberValue(Convert.ToDouble(cr));
802         }
803         catch(FormatException)
804         {
805             compareVal = new TextValue(cr);
806         }
```

```
807
808
809         }
810
811         private static ErrorValue ArgNotAvail()
812         {
813             return new ErrorValue("N/A");
814         }
815
816         private static MatrixValue GetMatrixValue(Sheet sheet, Expr es, int col, int row)
817         {
818             MatrixValue lookup;
819             CellArea lookupArea = es as CellArea;
820             CellRef lookupRef = es as CellRef;
821
822             if (lookupArea == null)
823             {
824                 if (lookupRef != null)
825                     lookupArea = new CellArea(lookupRef.sheet, lookupRef.raref, lookupRef.raref);
826                 else
827                     throw new ArgumentException("ARGTYPE");
828             }
829
830             lookup = lookupArea.Eval(sheet, col, row) as MatrixValue;
831
832             if (lookup == null) //As time of writing the above would always return Matrixvalue
833                 throw new ArgumentException("ARGTYPE");
834             return lookup;
835         }
836
837         private static ErrorValue ArgTypeErr()
838         {
839             return new ErrorValue("ARGTYPE");
840         }
841
842         private static ErrorValue ArgCountErr()
843         {
844             return new ErrorValue("ARGCOUNT");
845         }
846
847
848         private static MatrixValue GetMatrixValue(Value rangeValue)
849         {
850             MatrixValue range;
851             if (rangeValue is MatrixValue)
852                 range = rangeValue as MatrixValue;
853             else
854             {
855                 Value[,] values = new Value[1, 1];
856                 values[0, 0] = rangeValue;
857                 range = new MatrixValue(values);
```

```
858             }
859             return range;
860         }
861         private Function(String name, Applier applier)
862             : this(name, 0, applier) { }
863 
864         private Function(String name, int fixity, Applier applier) {
865             this.name = name;
866             this.applier = applier;
867             this.fixity = fixity;
868             table.Add(name, this);
869         }
870 
871         // Make number-valued nullary function, eg RAND()
872         private static Applier MakeFunction(Fun double> dlg) {
873             return
874                 delegate(Sheet sheet, Expr[] es, int col, int row) {
875                         if (es.Length == 0)
876                             return new NumberValue(dlg());
877                         else
878                             return ArgCountErr();
879                 };
880         }
881 
882         // Make number-valued constant nullary function, eg PI()
883         private static Applier MakeConstant(double d) {
884             NumberValue value = new NumberValue(d);
885             return
886                 delegate(Sheet sheet, Expr[] es, int col, int row) {
887                         if (es.Length == 0)
888                             return value;
889                         else
890                             return ArgCountErr();
891                 };
892         }
893 
894         // Make number-valued strict unary function, eg SIN(x)
895         private static Applier MakeFunction(Fun double, double> dlg) {
896             return
897                 delegate(Sheet sheet, Expr[] es, int col, int row) {
898                         if (es.Length == 1) {
899                             NumberValue v0 = es[0].Eval(sheet, col, row) as NumberValue;
900                             if (v0 != null)
901                                 return new NumberValue(dlg(v0.value));
902                             else
903                                 return ArgTypeErr();
904                         }
905                         else
906                             return ArgCountErr();
907                 };
908         }
```

```
909
910         // Make number-valued strict binary function, eg +
911         private static Applier MakeFunction(Fun double, double, double> dlg) {
912             return
913                 delegate(Sheet sheet, Expr[] es, int col, int row) {
914                         if (es.Length == 2) {
915                             NumberValue v0 = es[0].Eval(sheet, col, row) as NumberValue,
916                                         v1 = es[1].Eval(sheet, col, row) as NumberValue;
917                             //TODO: This is made as a part of testing
918                             //TEST: this will not function correctly when using logical functions such as
919                             if(v0 == null)
920                                 v0 = new NumberValue(0.0);
921                             if (v1 == null)
922                                 v1 = new NumberValue(0.0);
923
924                             return new NumberValue(dlg(v0.value, v1.value));
925
926                         }
927                         else
928                             return ArgCountErr();
929                 };
930         }
931
932         // Make string-valued strict binary function, eg &
933         private static Applier MakeFunction(Fun String, String, String> dlg) {
934             return
935                 delegate(Sheet sheet, Expr[] es, int col, int row) {
936                         if (es.Length == 2) {
937                             Value v0 = es[0].Eval(sheet, col, row),
938                                   v1 = es[1].Eval(sheet, col, row);
939                             if (v0 == null)
940                                 v0 = new NullValue();
941                             if (v1 == null)
942                                 v1 = new NullValue();
943                             //TODO: TEST: Hack to make this always work
944                             string val1 = v0.ToString();
945                             string val2 = v1.ToString();
946                             return new TextValue(dlg(val1, val2));
947
948                         }
949                         else
950                             return ArgCountErr();
951                 };
952         }
953
954         // Make boolean-valued strict binary function, eg ==
955         private static Applier MakePredicate(Fun double, double, bool> dlg) {
956             return
957                 MakeFunction(delegate(double x, double y) {
958                                 return dlg(x, y) ? 1.0 : 0.0;
959                 });
```

```
960         }
961
962         // Make number-valued strict variadic function, eg SUM(A1; B2:B7)
963         private static Applier MakeFunction(Fun Value[], double> dlg) {
964             return
965                 delegate(Sheet sheet, Expr[] es, int col, int row) {
966                        // try {
967                             return new NumberValue(dlg(Eval(es, sheet, col, row)));
968                         //} catch (ArgumentException e) {
969                          //   return ArgTypeErr();
970                         //}
971                 };
972         }
973
974         // Make string-valued strict variadic function, eg CONCATENATE(A1; B2:B7)
975         private static Applier MakeFunction(Fun Value[], string> dlg)
976         {
977             return
978                 delegate(Sheet sheet, Expr[] es, int col, int row)
979                 {
980                     try
981                     {
982                         return new TextValue(dlg(Eval(es, sheet, col, row)));
983                     }
984                     catch (ArgumentException)
985                     {
986                         return ArgTypeErr();
987                     }
988                 };
989         }
990
991         // Make number-valued strict monadic function, eg ROWS(A1:B7)
992         private static Applier MakeFunction(Fun Value, Value> dlg) {
993             return
994                 delegate(Sheet sheet, Expr[] es, int col, int row) {
995                         if (es.Length == 1)
996                             return dlg(es[0].Eval(sheet, col, row));
997                         else
998                             return ArgCountErr();
999                 };
1000        }
1001
1002        // Make general strict function, eg MMULT(A1:B3, E1:G2)
1003        private static Applier MakeFunction(Fun Value[], Value> dlg) {
1004            return
1005                delegate(Sheet sheet, Expr[] es, int col, int row) {
1006                        return dlg(Eval(es, sheet, col, row));
1007                };
1008        }
1009
1010        // Make function for H and V lookup
```

```
1011        private static Applier MakeLookupFunction(bool directionRows)
1012        {
1013            return delegate(Sheet sheet, Expr[] es, int col, int row)
1014                        {
1015                            if (es.Length == 4 || es.Length == 3)
1016                            {
1017                                NumberValue not_exact_match_number;
1018
1019                                if(es.Length == 4)
1020                                    not_exact_match_number = es[3].Eval(sheet, col, row) as NumberValue;
1021                                else
1022                                    not_exact_match_number = new NumberValue(1.0);
1023
1024                                Value value = es[0].Eval(sheet, col, row);
1025                                NumberValue indexNumber = es[2].Eval(sheet, col, row) as NumberValue;
1026                                bool es1CellArea = es[1] is CellArea;
1027                                bool es1CellRef = es[1] is CellRef;
1028                                if (value == null || value is ErrorValue)
1029                                    value = new NullValue();
1030
1031                                if (indexNumber != null && not_exact_match_number != null && (es1CellArea || es1CellRef))
1032                                {
1033                                    int index = ((int) indexNumber.value) - 1;
1034                                    bool not_exact_match = 1.0 == not_exact_match_number.value;
1035                                    if(es1CellArea)
1036                                    {
1037                                        //TODO: do not eval es[1], but eval each cell in the area one at a time and stop
1038                                        //when the correct one has been found (CaseCellAreaRowDirection --> look at it)
1039                                        //MatrixValue values = es[1].Eval(sheet, col, row) as MatrixValue;
1040
1041                                        CellArea matrixArea = es[1] as CellArea;
1042                                        Sheet currentSheet = matrixArea.sheet == null ? sheet : matrixArea.sheet;
1043                                        int sCol = matrixArea.ul.colAbs ? matrixArea.ul.colRef : matrixArea.ul.colRef + col;
1044                                        int eCol = matrixArea.lr.colAbs ? matrixArea.lr.colRef : matrixArea.lr.colRef + col;
1045                                        int sRow = matrixArea.ul.rowAbs ? matrixArea.ul.rowRef : matrixArea.ul.rowRef + row;
1046                                        int eRow = matrixArea.lr.rowAbs ? matrixArea.lr.rowRef : matrixArea.lr.rowRef + row;
1047
1048                                        //
1049                                        //es[1].GetAdresses
1050                                        if(directionRows)
1051                                            return CaseCellAreaRowDirection(col, value, index, not_exact_match, currentSheet,
1052                                                sCol, sRow, eCol, eRow);
1053                                        else
1054                                            return CaseCellAreaColDirection(row, value, index, not_exact_match, currentSheet,
1055                                                sCol, sRow, eCol, eRow);
1056
1057                                    }
1058                                    else //doesn't matter if this is Horisontal or Vertical there is only one value
1059                                    {
1060                                        return CaseCellRef(sheet, es, col, row, value, index, not_exact_match);
1061                                    }
```

```
1062                            }
1063                            else
1064                                return ArgTypeErr();
1065                        }
1066                        else
1067                            return ArgCountErr();
1068                    };
1069        }
1070
1071        private static Value CaseCellAreaRowDirection(int col, Value value, int index, bool not_exact_match,
1072            Sheet sheet, int sCol, int sRow, int eCol, int eRow)
1073        {
1074            index = index + sRow;
1075            int Mcol = -1;
1076            bool run = true;
1077            for (int c = sCol; c  = eCol && run; c++)
1078            {
1079                for (int r = sRow; r  = eRow && run; r++)
1080                {
1081                    if(sheet[c, r] != null)
1082                    {
1083                        if(!sheet.workbook.UseSupportGraph)
1084                          sheet[c, r].Eval(sheet, c, r);
1085                        Value val = sheet[c, r].GetValue(sheet, c, r);
1086                        val = val == null ? new NullValue() : val;
1087                        CompareResult result = val.CompareToOther(value);
1088                        if (not_exact_match)
1089                        {
1090                            if (result == CompareResult.ThisIsSmallest)
1091                                run = false;
1092                            else
1093                                Mcol = c;
1094                        }
1095                        if (result == CompareResult.Equal)
1096                        {
1097                            Mcol = c;
1098                            run = false;
1099                        }
1100                    }
1101                }
1102            }
1103            if (Mcol != -1 && (int)index   eRow-sRow && sheet[Mcol, index] != null)
1104            {
1105                if (!sheet.workbook.UseSupportGraph)
1106                  sheet[Mcol, index].Eval(sheet, Mcol, index);
1107                return sheet[Mcol, index].GetValue(sheet, Mcol, index);
1108            }
1109            else
1110                return new NullValue();
1111        }
1112
```

```
1113        private static Value CaseCellAreaColDirection(int row, Value value, int index, bool not_exact_match,
1114            Sheet sheet, int sCol, int sRow, int eCol, int eRow)
1115        {
1116            index = index + sCol;
1117            int MRow = -1;
1118            bool run = true;
1119            for (int r = sRow; r  = eRow && run; r++)
1120            {
1121                for (int c = sCol; c  = eCol && run; c++)
1122                {
1123                    if (sheet[c, r] != null)
1124                    {
1125                        if (!sheet.workbook.UseSupportGraph)
1126                          sheet[c, r].Eval(sheet, c, r);
1127
1128                        Value val = sheet[c, r].GetValue(sheet, c, r);
1129                        val = val == null ? new NullValue() : val;
1130                        CompareResult result = val.CompareToOther(value);
1131                        if (not_exact_match)
1132                        {
1133                            if (result == CompareResult.ThisIsSmallest)
1134                                run = false;
1135                            else
1136                                MRow = r;
1137                        }
1138                        if (result == CompareResult.Equal)
1139                        {
1140                            MRow = r;
1141                            run = false;
1142                        }
1143                    }
1144                }
1145            }
1146            if (MRow != -1 && index   eCol-sCol && sheet[index, MRow] != null)
1147            {
1148                if (!sheet.workbook.UseSupportGraph)
1149                  sheet[index, MRow].Eval(sheet, index, MRow);
1150                return sheet[index, MRow].GetValue(sheet, index, MRow);
1151            }
1152            else
1153                return new ErrorValue("REFERENCE");
1154        }
1155
1156
1157
1158        private static Value CaseCellRef(Sheet sheet, Expr[] es, int col, int row, Value value, int index, bool not_exact_match)
1159        {
1160            Value onlyValue = es[2].Eval(sheet, col, row);
1161
1162            if (index == 1)
1163            {
```

```
1164                if (onlyValue != null && (!not_exact_match || value.CompareToOther(onlyValue) == CompareResult.Equal))
1165                    return onlyValue;
1166                else
1167                    return new ErrorValue("N/A");
1168            }
1169            else
1170                return new ErrorValue("REFERENCE");
1171        }
1172
1173        // Evaluate expression array
1174        private static Value[] Eval(Expr[] es, Sheet sheet, int col, int row) {
1175            Value[] vs = new Value[es.Length];
1176            for (int i = 0; i   es.Length; i++)
1177                vs[i] = es[i].Eval(sheet, col, row);
1178            return vs;
1179        }
1180
1181        // Apply act to numbers in vs, and recursively to matrix values
1182        private static void Apply(Value[] vs, Act double> act) {
1183            foreach (Value v in vs) {
1184                if (v is NumberValue)
1185                    act((v as NumberValue).value);
1186                else if (v is MatrixValue)
1187                    (v as MatrixValue).Apply(act);
1188                //else do nothing
1189                  //  throw new ArgumentException();
1190            }
1191        }
1192
1193        // Apply act to numbers in vs, and recursively to matrix values
1194        private static void Apply(Value[] vs, Act string> act) {
1195            foreach (Value v in vs) {
1196                if (v is TextValue)
1197                    act((v as TextValue).value);
1198                else if (v is MatrixValue)
1199                    (v as MatrixValue).Apply(act);
1200                //else do nothing
1201                  //  throw new ArgumentException();
1202            }
1203        }
1204    }
1205 }
1206
```

CoreCalc

Sheet

```
1  // CoreCalc, a spreadsheet core implementation
2
3  // ------------------------------------------------------------------------
4  // Copyright (c) 2006 Peter Sestoft and Thomas S. Iversen
5
6  // Permission is hereby granted, free of charge, to any person
7  // obtaining a copy of this software and associated documentation
8  // files (the "Software"), to deal in the Software without
9  // restriction, including without limitation the rights to use, copy,
10 // modify, merge, publish, distribute, sublicense, and/or sell copies
11 // of the Software, and to permit persons to whom the Software is
12 // furnished to do so, subject to the following conditions:
13
14 //  * The above copyright notice and this permission notice shall be
15 //    included in all copies or substantial portions of the Software.
16
17 //  * The software is provided "as is", without warranty of any kind,
18 //    express or implied, including but not limited to the warranties of
19 //    merchantability, fitness for a particular purpose and
20 //    noninfringement.  In no event shall the authors or copyright
21 //    holders be liable for any claim, damages or other liability,
22 //    whether in an action of contract, tort or otherwise, arising from,
23 //    out of or in connection with the software or the use or other
24 //    dealings in the software.
25 // ------------------------------------------------------------------------
26
27 using System;
28 using System.Collections.Generic;
29 using System.Diagnostics;
30 using C5;
31 using CoreCalc.SupportGraf;
32 using Utilities;
33 using FAP;
34 // A Sheet contains Cells and belongs to a Workbook
35
36 namespace CoreCalc {
37     public sealed class Sheet :ISheet<Sheet>
38     {
39         public const int cols = 20, rows = 20;   // Default sheet size
40         private bool[,] inTopologicalSort;
41         private String name;
42         public readonly Workbook workbook;       // Non-null
43         //TODO: made this public, should just make sheet implement enumerable interface
44         public Cell[,] cells;   // Indexed by (col, row); unused cells are null
45         private SupportGraph supportGraph;  // Corresponds to the cells;
46         //a cells which does not contain support graph is null
47
48         internal SupportGraph SupportGraph
49         {
50             get { return supportGraph; }
51             set { supportGraph = value; }
```

```
52          }
53
54
55
56
57          public Sheet(Workbook workbook, String name)
58              : this(workbook, name, cols, rows)
59          {
60          }
61
62          public Sheet(Workbook workbook, String name, int cols, int rows)
63          {
64              this.workbook = workbook;
65              this.name = name;
66              this.cells = new Cell[cols, rows];
67              this.SupportGraph = new SupportGraph(cols, rows, this);
68              workbook.AddSheet(this);
69          }
70
71          // Recalculate all cells by evaluating their contents
72          public int Recalculate()
73          {
74              int back = 0;
75              for (int col = 0; col < Cols; col++)
76                  for (int row = 0; row < Rows; row++)
77                  {
78                      Cell cell = cells[col, row];
79                      if (cell != null)
80                      {
81                          cell.Eval(this, col, row);
82                          back++;
83                      }
84                  }
85
86              return back;
87          }
88
89          public void Recalculate(CellAddr addr)
90          {
91              Cell cell = cells[addr.col, addr.row];
92              if (cell != null)
93              {
94                  cell.Eval(this, addr.col, addr.row);
95              }
96          }
97
98
99
100         // Show the contents of all non-null cells
101         public void ShowAll(Shower show)
102         {
```

```
103             for (int col = 0; col < Cols; col++)
104                 for (int row = 0; row < Rows; row++)
105                 {
106                     Cell cell = cells[col, row];
107                     show(col, row, cell != null ? cell.ShowValue(this, col, row) : null);
108                 }
109         }
110 
111         // Reset recomputation flags after a circularity has been found
112         public void Reset()
113         {
114             foreach (Cell cell in cells)
115                 if (cell != null)
116                     cell.Reset();
117         }
118 
119         // Insert cell from parsed constant or formula in sheet[col, row]
120         public Cell InsertCell(String text, CellAddr ca)
121         {
122             Stopwatch stopwatch = new Stopwatch();
123             stopwatch.Start();
124             Cell newCell = Cell.Parse(text, workbook, ca.col, ca.row);
125             if(workbook.UseSupportGraph)
126                 supportGraph.InsertCellToGraph(this[ca], newCell, ca);
127 
128             this[ca] = newCell;
129             stopwatch.Stop();
130             Logger.Log.Info(this, "Insert cell took (without recalculation): " + stopwatch.Elapsed);
131             return this[ca];
132         }
133 
134         // Insert cell, which must be Formula, as matrix formula
135         // in area ((ulCol,ulRow), (lrCol, lrRow))
136         public void InsertMatrixFormula(Cell cell, int col, int row,
137                                         CellAddr ulCa, CellAddr lrCa)
138         {
139             Logger.Log.Debug(this, "Insert formula as matrix formula");
140             Formula formula = cell as Formula;
141             if (cell == null)
142                 throw new Exception("Invalid matrix formula");
143             else
144             {
145                 CachedMatrixFormula cmf
146                   = new CachedMatrixFormula(formula, col, row, ulCa, lrCa);
147                 int cols = lrCa.col - ulCa.col + 1, rows = lrCa.row - ulCa.row + 1;
148                 for (int c = 0; c < cols; c++)
149                     for (int r = 0; r < rows; r++)
150                         cells[ulCa.col + c, ulCa.row + r] = new MatrixFormula(cmf, c, r);
151             }
152         }
153 
```

```
154         //TODO: not used
155         // Copy cell to cell address ca
156         public void PasteCell(Cell cell, CellAddr ca)
157         {
158             PasteCell(cell, ca, 1, 1);
159         }
160 
161         // Copy cell to area ((col,row), (col+cols-1,row+rows-1))
162         public void PasteCell(Cell cell, CellAddr ca, int cols, int rows)
163         {
164             Stopwatch stopwatch = new Stopwatch();
165             stopwatch.Start();
166             //Delete cells from support graph first
167             if(workbook.UseSupportGraph)
168                 supportGraph.DeleteAreaFromGraph(ca.col, ca.col+cols-1, ca.row, ca.row+rows-1);
169 
170             Logger.Log.Debug(this, "Copy paste cell to cell area");
171             //if (cells[ca.col, ca.row] == null) Logger.Log.Debug(this, "  Cell is null, no update needed");
172             if (cell is Formula)
173             {
174                 // Clone cache but share expression f.e between all target cells
175                 Formula f = (Formula)cell;
176                 for (int c = ca.col; c < ca.col+cols; c++)
177                     for (int r = ca.row; r < ca.row+rows; r++)
178                     {
179                         cells[c, r] = new Formula(f);
180                     }
181                 if (workbook.UseSupportGraph)
182                     supportGraph.CopyToCells(f.Expr, ca, cols, rows);
183                 stopwatch.Stop();
184                 Logger.Log.Info(this, "Copy cell took (without recalculation): " + stopwatch.Elapsed);
185             }
186             else if (cell is ConstCell)
187             {
188                 // Share constant cell between all target cells
189                 for (int c = ca.col; c < ca.col + cols; c++)
190                     for (int r = ca.row; r < ca.row + rows; r++)
191                     {
192                        cells[c, r] = cell;
193                     }
194                 stopwatch.Stop();
195                 Logger.Log.Info(this, "Copy cell took (without recalculation): " + stopwatch.Elapsed);
196             }
197             else
198                 throw new Exception("Cannot copy cell: " + cell);
199         }
200 
201         //TODO: not used
202         // Move cell (fromCol, fromRow) to cell (col,row)
203         public void MoveCell(int fromCol, int fromRow, int col, int row, Sheet sheet)
204         {
```

```
205             Cell cell = cells[fromCol, fromRow];
206             cells[col, row] = cell.MoveContents(col - fromCol, row - fromRow);
207         }
208 
209         public void DeleteCell(CellAddr ca)
210         {
211             Stopwatch stopwatch = new Stopwatch();
212             stopwatch.Start();
213             if(workbook.UseSupportGraph)
214                 supportGraph.DeleteCellFromGraph(this[ca], ca);
215             this[ca] = null;
216             stopwatch.Stop();
217             Logger.Log.Info(this, "Delete cell took (without recalculation): " + stopwatch.Elapsed);
218         }
219 
220         // Insert N new rows just before row R >= 0
221         public void InsertRowCols(int R, int N, bool doRows, bool useSupportGraph)
222         {
223             Stopwatch stopwatch = new Stopwatch();
224             stopwatch.Start();
225             Logger.Log.Debug(this, "Inserting rows or cols");
226             // Check that this will not split a matrix formula
227             if (R >= 1)
228                 if (doRows)
229                 {
230                     for (int col = 0; col < Cols; col++)
231                     {
232                         Cell cell = cells[col, R - 1];
233                         MatrixFormula mf = cell as MatrixFormula;
234                         if (mf != null && mf.Contains(col, R))
235                             throw new Exception("Row insert would split matrix formula");
236                     }
237                 }
238                 else
239                 {
240                     for (int row = 0; row < Rows; row++)
241                     {
242                         Cell cell = cells[R - 1, row];
243                         MatrixFormula mf = cell as MatrixFormula;
244                         if (mf != null && mf.Contains(R, row))
245                             throw new Exception("Column insert would split matrix formula");
246                     }
247                 }
248 
249             //Make a new cell array to contain the new cells
250             Cell[,] newCells;
251             if (doRows)
252             {
253                 newCells = new Cell[this.Cols, this.Rows + N];
254             }
255             else
```

```
256             {
257                 newCells = new Cell[this.Cols + N, this.Rows];
258             }
259 
260             // Adjust formulas in all sheets.  The dictionary records adjusted
261             // expressions to preserve sharing of expressions where possible.
262             Dictionary<Expr, System.Collections.Generic.KeyValuePair<Adjusted<Expr>, bool>> adjusted
263                     = new Dictionary<Expr, System.Collections.Generic.KeyValuePair<Adjusted<Expr>, bool>>();
264              ashDictionary<Sheet, List< ashDictionary<Expr, FAPList>>> expressionOccurrenceMapForSheets = new  ashDictionary<Sheet↙
      , List< ashDictionary<Expr, FAPList>>>();
265             foreach (Sheet sheet in workbook)
266             {
267                 expressionOccurrenceMapForSheets.Add(sheet, new List< ashDictionary<Expr, FAPList>>());
268                 if (doRows)
269                 {
270                     Cell[,] cs = sheet.cells;
271                     for (int r = 0; r < sheet.Rows; r++)
272                     {
273                          ashDictionary<Expr, FAPList> currentRow = new  ashDictionary<Expr, FAPList>();
274                         if(r == R-1)
275                             for (int i = 0; i < N; i++)
276                                 expressionOccurrenceMapForSheets[sheet].Add(null);
277 
278                         for (int c = 0; c < sheet.Cols; c++)
279                         {
280                             Cell cell = cs[c, r];
281                             if (cell != null)
282                             {
283                                 cell.InsertRowCols(adjusted, this, sheet == this, R, N, r, c, doRows, currentRow);
284                             }
285                         }
286                         expressionOccurrenceMapForSheets[sheet].Add(currentRow);
287                     }
288                 }
289                 else
290                 {
291                     Cell[,] cs = sheet.cells;
292                     for (int c = 0; c < sheet.Cols; c++)
293                     {
294                          ashDictionary<Expr, FAPList> currentColumn = new  ashDictionary<Expr, FAPList>();
295                         if (c == R-1)
296                             for (int i = 0; i < N; i++)
297                                 expressionOccurrenceMapForSheets[sheet].Add(null);
298                         for (int r = 0; r < sheet.Rows; r++)
299                         {
300                             Cell cell = cs[c, r];
301                             if (cell != null)
302                             {
303                                 cell.InsertRowCols(adjusted, this, sheet == this, R, N, r, c, doRows, currentColumn);
304                             }
305                         }
```

```
306                     expressionOccurrenceMapForSheets[sheet].Add(currentColumn);
307                 }
308             }
309         }
310         if (doRows)
311         {
312             //copy normal cells
313             for (int r = 0; r < R; r++)
314             {
315                 for (int c = 0; c < Cols; c++)
316                 {
317                     newCells[c, r] = cells[c, r];
318                 }
319 
320             }
321 
322             // Move the rows R, R+1, ... later by N rows in current sheet
323             for (int r = Rows - 1; r >= R + N; r--)
324                 for (int c = 0; c < Cols; c++)
325                     newCells[c, r] = cells[c, r - N];
326             //// Finally, null out the fresh rows
327             //for (int r = 0; r < N; r++)
328             //    for (int c = 0; c < Cols; c++)
329             //        cells[c, r + R] = null;
330 
331         }
332         else
333         {
334             //Copy normal cells
335             for (int c = 0; c < R; c++)
336                 for (int r = 0; r < Rows; r++)
337                 {
338                     newCells[c, r] = cells[c, r];
339                 }
340 
341             // Move the columns R, R+1, ... later by N columns in current sheet
342             for (int c = Cols - 1; c >= R + N; c--)
343                 for (int r = 0; r < Rows; r++)
344                     newCells[c, r] = cells[c - N, r];
345             //// Finally, null out the fresh columns
346             //for (int c = 0; c < N; c++)
347             //    for (int r = 0; r < Rows; r++)
348             //        cells[c + R, r] = null;
349         }
350 
351         cells = newCells;
352 
353         if(useSupportGraph)
354             supportGraph.AddLinesToGraph(R, N, doRows, expressionOccurrenceMapForSheets, this);
355         stopwatch.Stop();
356         Logger.Log.Info(this, "Insert Rows/Cols took (without recalculation): " + stopwatch.Elapsed);
```

```
357
358         }
359
360
361
362         // Show contents, if any, of cell at (col, row)
363         public String Show(int col, int row)
364         {
365             if (0 <= col && col < Cols && 0 <= row && row < Rows)
366             {
367                 Cell cell = cells[col, row];
368                 if (cell != null)
369                     return cell.Show(col, row, workbook.Format);
370             }
371             return null;
372         }
373
374         // Show value, if any, of cell (col, row)
375         public String ShowValue(int col, int row)
376         {
377             if (0 <= col && col < Cols && 0 <= row && row < Rows)
378             {
379                 Cell cell = cells[col, row];
380                 if (cell != null)
381                     return cell.ShowValue(this, col, row);
382             }
383             return null;
384         }
385
386         // Get and set cell contents
387         public Cell this[int col, int row]
388         {
389             get
390             {
391                 return cells[col, row];
392             }
393             set
394             {
395                 cells[col, row] = value;
396             }
397         }
398
399         public Cell this[CellAddr ca]
400         {
401             get
402             {
403                 return cells[ca.col, ca.row];
404             }
405             set
406             {
407                 cells[ca.col, ca.row] = value;
```

```
408             }
409         }
410 
411         public int Cols
412         {
413             get { return cells.GetLength(0); }
414         }
415 
416         public int Rows
417         {
418             get { return cells.GetLength(1); }
419         }
420 
421         public String Name
422         {
423             get { return name; }
424             set { name = value; }
425         }
426 
427         public bool[,] InTopologicalSort
428         {
429             get { return inTopologicalSort; }
430             set { inTopologicalSort = value; }
431         }
432 
433         public override string ToString()
434         {
435             return name;
436         }
437 
438         internal C5.IList<FullCellAddr<Sheet>> GetSupportGraph(CellAddr[] alteredCells, FullCellAddr<Sheet>[] altered,
    TopologicalSortType topsort)
439         {
440             return supportGraph.GetSupportGraph(alteredCells, altered,  topsort);
441         }
442 
443         public bool ContainsVolatileFunction(Cell c)
444         {
445             if (c is Formula)
446             {
447                 Formula f = c as Formula;
448                 if (f.Expr is FunCall)
449                 {
450                     string[] volatileFunctionNames = { "RAND", "NOW" };
451                     FunCall fc = f.Expr as FunCall;
452                     return ContainsVolatileFunction(fc, volatileFunctionNames);
453                 }
454             }
455             return false;
456         }
457 
```

```
458         public bool ContainsVolatileFunction(Expr e)
459         {
460             string[] volatileFuncNames = {"RAND", "NOW"};
461             if (e is FunCall) return ContainsVolatileFunction((FunCall)e, volatileFuncNames);
462             return false;
463         }
464 
465         private bool ContainsVolatileFunction(FunCall fc, string[] volatileFunctionNames)
466         {
467             foreach (string s in volatileFunctionNames){
468                 if (s.ToLower() == fc.function.name.ToLower())
469                 {
470                     Logger.Log.Info(this, "This cell should always be recalculated since it contains one or more volatile functions");
471                     return true;
472                 }
473             }
474 
475             foreach (Expr e in fc.es)
476             {
477                 if (e is FunCall)
478                 {
479                     if (ContainsVolatileFunction((FunCall)e, volatileFunctionNames))
480                     {
481                         return true;
482                     }
483                 }
484             }
485 
486             return false;
487         }
488 
489 
490         #region IComparable<Sheet> Members
491 
492         public int CompareTo(Sheet other)
493         {
494             return Name.CompareTo(other.Name);
495         }
496 
497         #endregion
498     }
499 }
500 
```

CoreCalc

Types

```
// CoreCalc, a spreadsheet core implementation

// ------------------------------------------------------------------------
// Copyright (c) 2006 Peter Sestoft and Thomas S. Iversen

// Permission is hereby granted, free of charge, to any person
// obtaining a copy of this software and associated documentation
// files (the "Software"), to deal in the Software without
// restriction, including without limitation the rights to use, copy,
// modify, merge, publish, distribute, sublicense, and/or sell copies
// of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:

//  * The above copyright notice and this permission notice shall be
//    included in all copies or substantial portions of the Software.

//  * The software is provided "as is", without warranty of any kind,
//    express or implied, including but not limited to the warranties of
//    merchantability, fitness for a particular purpose and
//    noninfringement.  In no event shall the authors or copyright
//    holders be liable for any claim, damages or other liability,
//    whether in an action of contract, tort or otherwise, arising from,
//    out of or in connection with the software or the use or other
//    dealings in the software.
// ------------------------------------------------------------------------

using System;
using System.Text;

// Delegate types, exception classes and formula formatting options

namespace CoreCalc {

  // ---------------------------------------------------------------
  // Delegate types used in the recalculation engine

  public delegate void Shower(int col, int row, String s);
  public delegate Value Applier(Sheet sheet, Expr[] es, int col, int row);

  public delegate R Fun<R>();
  public delegate R Fun<A1, R>(A1 x1);
  public delegate R Fun<A1, A2, R>(A1 x1, A2 x2);
  public delegate void Act();
  public delegate void Act<A1>(A1 x1);
  public delegate void Act<A1, A2>(A1 x1, A2 x2);

  // ---------------------------------------------------------------
  // Exception classes

  class BadFormatException : Exception {
    public BadFormatException(String msg) : base(msg) { }
```

```
52    }
53
54    class CyclicException : Exception {
55      public CyclicException(String msg) : base(msg) { }
56    }
57
58    class ErrorValueException : Exception {
59      public readonly ErrorValue errorValue;
60
61      public ErrorValueException(ErrorValue errorValue) {
62        this.errorValue = errorValue;
63      }
64    }
65
66    class ImpossibleException : Exception {
67      public ImpossibleException(String msg) : base(msg) { }
68    }
69
70    class NotImplementedException : Exception {
71      public NotImplementedException(String msg) : base(msg) { }
72    }
73
74    // ------------------------------------------------------------------
75    // Formula formatting options
76
77    public class Formats {
78      public enum RefType { A1, C0R0, R1C1 }
79
80      private RefType refFmt = RefType.A1;
81      private char argDelim = ',';
82      private char rangeDelim = ':';
83
84      public RefType RefFmt {
85        get { return refFmt; }
86        set { refFmt = value; }
87      }
88
89      public char RangeDelim {
90        get { return rangeDelim; }
91        set { rangeDelim = value; }
92      }
93
94      public char ArgDelim {
95        get { return argDelim; }
96        set { argDelim = value; }
97      }
98    }
99  }
100
```

CoreCalc

Workbook

```
// CoreCalc, a spreadsheet core implementation

// ------------------------------------------------------------------------
// Copyright (c) 2006 Peter Sestoft and Thomas S. Iversen

// Permission is hereby granted, free of charge, to any person
// obtaining a copy of this software and associated documentation
// files (the "Software"), to deal in the Software without
// restriction, including without limitation the rights to use, copy,
// modify, merge, publish, distribute, sublicense, and/or sell copies
// of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:

//  * The above copyright notice and this permission notice shall be
//    included in all copies or substantial portions of the Software.

//  * The software is provided "as is", without warranty of any kind,
//    express or implied, including but not limited to the warranties of
//    merchantability, fitness for a particular purpose and
//    noninfringement.  In no event shall the authors or copyright
//    holders be liable for any claim, damages or other liability,
//    whether in an action of contract, tort or otherwise, arising from,
//    out of or in connection with the software or the use or other
//    dealings in the software.
// ------------------------------------------------------------------------

using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;
using C5;
using CoreCalc.SupportGraf;
using Logger;
using FAP;
using System.Diagnostics;

// A Workbook is a collection of named Sheets

namespace CoreCalc {
  public sealed class Workbook : IEnumerable<Sheet> {
    internal List<Sheet> sheets; // All non-null and distinct
    public Formats Format;      // Formula formatting options
    private bool set;           // For controlling recomputation
    private bool cyclic;        // If true, workbook may be inconsistent
    //TODO: Maybe make this into fapdoublelists
    internal HashSet<FullCellAddr<Sheet>> volatileExpressions;
    internal HashSet<FullCellAddr<Sheet>> waitingForManualUpdate;
    private WorkbookForm gui;
    private Stopwatch stopwatch;
    private string workbookName;

```

```
52      public string WorkbookName
53      {
54          get { return workbookName; }
55          set { workbookName   value; }
56      }
57      internal  int sizeOfLastTopologicalSort; //TODO: refactor this
58
59      private NormalizerThread normalizeThread;
60        private TopologicalSortType topsort;
61        private readonly TopologicalSortType defaultTopoSort   TopologicalSortType.NoloopsAllowed;
62
63        public Workbook()
64      {
65          topsort   defaultTopoSort;
66        volatileExpressions   new HashSet<FullCellAddr<Sheet>>();
67        waitingForManualUpdate   new HashSet<FullCellAddr<Sheet>>();
68        this.sheets   new List<Sheet>();
69        Format   new Formats();
70        normalizeThread   new NormalizerThread(this);
71        sizeOfLastTopologicalSort   0;
72        stopwatch   new Stopwatch();
73        Log.Info(typeof(Workbook), "Using high resolution timer: " + Stopwatch.IsHighResolution);
74        long frequency   Stopwatch.Frequency;
75        Log.Info(typeof(Workbook), "Timer frequency in ticks per second   "+frequency);
76        double nanosecPerTick   (1000.0 * 1000.0 * 1000.0) / frequency;
77        Log.Fatal(typeof(Workbook), string.Format("Timer is accurate within {0} nanoseconds", nanosecPerTick));
78
79      }
80
81        public void attachGui(WorkbookForm gui)
82        {
83           this.gui   gui;
84        }
85
86      public bool UseSupportGraph
87      {
88          get { return gui.UseSupportGraph; }
89          set { gui.UseSupportGraph   value; }
90      }
91
92      public bool RecalculateVolatileFunctions
93      {
94          get { return gui.RecalculateVolatileFunctions; }
95          set { gui.RecalculateVolatileFunctions   value; }
96      }
97
98      public bool ManualRecalculation
99      {
100         get { return gui.ManualRecalculation; }
101         set { gui.ManualRecalculation   value; }
102     }
```

```
103
104     public void Dispose()
105     {
106         StopNormalizer();
107     }
108
109     public void AddSheet(Sheet sheet) {
110       sheets.Add(sheet);
111     }
112
113     public Sheet this[String name] {
114       get {
115         foreach (Sheet sheet in sheets)
116           if (sheet.Name    name)
117             return sheet;
118         throw new Exception("No sheet called " + name);
119       }
120     }
121
122       public void Recalculate(Sheet altered, CellAddr[] alteredCells)
123       {
124           Recalculate(gui.UseSupportGraph, gui.RecalculateVolatileFunctions, gui.ManualRecalculation, false, altered,
    alteredCells);
125       }
126
127     public TimeSpan Recalculate(bool useSupportGraph, bool recalculateVolatileFunctions, bool manualRecalculation, bool
    fullRecalculation, Sheet altered, CellAddr[] alteredCells)
128     {
129         if (manualRecalculation && !fullRecalculation) return new TimeSpan();
130         Log.Info(this, "Recalculation called");
131         if (useSupportGraph)
132         {
133             sizeOfLastTopologicalSort   0;
134             stopwatch.Reset();
135             stopwatch.Start();
136             Log.Info(this, "Using supportgraph in recalculation");
137             cyclic   false;
138             //set   !set;
139             C5.LinkedList<FullCellAddr<Sheet>> volatileAndQuededCells   new C5.LinkedList<FullCellAddr<Sheet>>();
140             if(recalculateVolatileFunctions) volatileAndQuededCells.AddAll(volatileExpressions.ToArray());
141             if (manualRecalculation && fullRecalculation)
142             {
143                 volatileAndQuededCells.AddAll(waitingForManualUpdate);
144                 waitingForManualUpdate.Clear();
145             }
146             TimeSpan beforeSup   stopwatch.Elapsed;
147             C5.IList<FullCellAddr<Sheet>> supportGraph   altered.GetSupportGraph(alteredCells, volatileAndQuededCells.ToArray(),
    Topsort);
148             TimeSpan afterSup   stopwatch.Elapsed;
149             TimeSpan resultSup   afterSup.Subtract(beforeSup);
150             Log.Fatal(this, "Doing the topological sort and thereby finding the support graph took: " + resultSup);
```

```
151             //TODO: make updates on all sheets
152             TimeSpan beforeRecalc   stopwatch.Elapsed;
153             foreach (FullCellAddr<Sheet> addr in supportGraph)
154             {
155                 if (addr.sheet[addr.col, addr.row] !  null)
156                     addr.sheet[addr.col, addr.row].Eval(addr.sheet, addr.col, addr.row);
157             }
158             TimeSpan afterRecalc   stopwatch.Elapsed;
159             TimeSpan resultRecalc   afterRecalc.Subtract(beforeRecalc);
160             Log.Fatal(this, "Recalcing took: " + resultRecalc);
161             sizeOfLastTopologicalSort   supportGraph.Count;
162             stopwatch.Stop();
163             TimeSpan duration   stopwatch.Elapsed;
164 
165             Log.Fatal(this, "Recalculation with supportgraph took " + duration);
166             return duration;
167         }
168         else
169         {
170             stopwatch.Reset();
171             stopwatch.Start();
172             Log.Info(this, "Doing full recalculation");
173             cyclic   false;
174             set   !set;
175             // Now for all cached expressions, ce.visited !  set and ce.uptodate !  set
176 
177             //TEST: In the below area the try catch block have been disabled as this makes it
178             //Easier to debug using VS2005
179             //try
180             //{
181             int recalcCells   0;
182 
183             foreach (Sheet sheet in sheets)
184                     recalcCells +  sheet.Recalculate();
185             //}
186             //catch (Exception exn)
187             //{
188             //    foreach (Sheet sheet in sheets)
189             //        sheet.Reset();
190             //    if (exn is CyclicException)
191             //        cyclic   true;
192             //    throw;
193             //}
194             stopwatch.Stop();
195             TimeSpan duration   stopwatch.Elapsed;
196             Log.Fatal(this, "Recalculation without supportgraph took " + duration);
197             Log.Fatal(this, "The number of eval cells was:" + recalcCells);
198             return duration;
199         }
200       // Now for all cached expressions, ce.visited    set and ce.uptodate    set
201     }
```

```
202
203     internal TimeSpan RecalculateFull()
204     {
205         if (sheets.Count !  0)
206         {
207             return Recalculate(gui.UseSupportGraph, gui.RecalculateVolatileFunctions, gui.ManualRecalculation, true, sheets[0],
    new CellAddr[0]);
208             //Recalculate(false, false, false, true, null, null);
209         }
210         return new TimeSpan();
211     }
212
213     public int SheetCount {
214       get { return sheets.Count; }
215     }
216
217     public bool Set {
218       get { return set; }
219     }
220
221     public bool Cyclic {
222       get { return cyclic; }
223     }
224
225       internal HashSet<FullCellAddr<Sheet>> VolatileExpressions
226       {
227           get { return volatileExpressions; }
228       }
229
230       internal HashSet<FullCellAddr<Sheet>> WaitingForManualUpdate
231       {
232           get { return waitingForManualUpdate; }
233       }
234
235       public TopologicalSortType Topsort
236       {
237           get { return topsort; }
238           set { topsort   value; }
239       }
240
241       IEnumerator<Sheet> IEnumerable<Sheet>.GetEnumerator() {
242       foreach (Sheet sheet in sheets)
243         yield return sheet;
244     }
245
246     IEnumerator IEnumerable.GetEnumerator() {
247       foreach (Sheet sheet in sheets)
248         yield return sheet;
249     }
250
251     public void Clear() {
```

```
252       sheets.Clear();
253     }
254
255     public bool StartNormalizer()
256     {
257         return normalizeThread.StartNormalizerThread();
258     }
259
260     public bool StopNormalizer()
261     {
262         return normalizeThread.StopNormalizerThread();
263     }
264
265     public void ThreadLogging(bool logThread)
266     {
267         normalizeThread.LogDuringThread   logThread;
268     }
269
270     public void PrintListOfVolatileFunctions()
271     {
272         StringBuilder sb   new StringBuilder();
273         sb.Append("Current list of volatile functions: \r\n");
274         foreach (FullCellAddr<Sheet> fca in VolatileExpressions)
275         {
276             sb.Append(fca.sheet.Name + "!" + CellAddr.ColumnName(fca.col) + (fca.row+1) + "\r\n");
277         }
278         if (VolatileExpressions.Count    0) sb.Append("None");
279         Log.Fatal(this, sb);
280     }
281
282       public void PrintListOfQeuedCellsWaitingForManualUpdate()
283       {
284           StringBuilder sb   new StringBuilder();
285           sb.Append("Current list of queued cells waiting for an update: \r\n");
286           foreach (FullCellAddr<Sheet> fca in WaitingForManualUpdate)
287           {
288               sb.Append(fca.sheet.Name + "!" + CellAddr.ColumnName(fca.col) + (fca.row + 1) + "\r\n");
289           }
290           if (WaitingForManualUpdate.Count    0) sb.Append("None");
291           Log.Fatal(this, sb);
292       }
293
294
295     internal void BuildSupportGraph()
296     {
297         bool directionRows   true;
298         HashDictionary<Sheet, SupportGraph> realSupportGraphs   SupportGraphBuilder.BuildSupportGraph(this, directionRows);
299         SupportGraphBuilder.ApplySupportGraphs(realSupportGraphs);
300     }
301
302
```

```
303 }
304 }
305
```

CoreCalc

CellAddressing

```
// CoreCalc, a spreadsheet core implementation

// ------------------------------------------------------------------------
// Copyright (c) 2006 Peter Sestoft

// Permission is hereby granted, free of charge, to any person
// obtaining a copy of this software and associated documentation
// files (the "Software"), to deal in the Software without
// restriction, including without limitation the rights to use, copy,
// modify, merge, publish, distribute, sublicense, and/or sell copies
// of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:

//  * The above copyright notice and this permission notice shall be
//    included in all copies or substantial portions of the Software.

//  * The software is provided "as is", without warranty of any kind,
//    express or implied, including but not limited to the warranties of
//    merchantability, fitness for a particular purpose and
//    noninfringement.  In no event shall the authors or copyright
//    holders be liable for any claim, damages or other liability,
//    whether in an action of contract, tort or otherwise, arising from,
//    out of or in connection with the software or the use or other
//    dealings in the software.
// ------------------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Text;
using Logger;

// Classes for representing absolute cell addresses,
// relative/absolute cell references, and
// adjusted expressions and refererences

namespace CoreCalc {

  // ---------------------------------------------------------------
  // A CellAddr is an absolute, zero-based (col, row) location

  [Serializable]
  public struct CellAddr
  {
    public readonly int col, row;
    public static readonly CellAddr A1 = new CellAddr(0, 0);

    public CellAddr(int col, int row) {
        //Log.Debug(typeof(CellAddr), "--- " + ColumnName(col) + (row+1)+ " selected");
      this.col = col;
      this.row = row;
    }
```

```
52
53      public CellAddr(RARef cr, int col, int row) {
54        this.col = cr.colAbs ? cr.colRef : cr.colRef + col;
55        this.row = cr.rowAbs ? cr.rowRef : cr.rowRef + row;
56          //TODO: THIS SHOULD BE REMOVED
57          StringBuilder sb = new StringBuilder();
58          sb.Append("References: ");
59          if (cr.colAbs) sb.Append("$" + ColumnName(cr.colRef));
60          else sb.Append(ColumnName(cr.colRef + col));
61          if (cr.rowAbs) sb.Append("$" + (cr.rowRef + 1));
62          else sb.Append((cr.rowRef + row + 1));
63      }
64
65      // Turn an A1-format string into an absolute cell address
66      public CellAddr(String a1Ref) : this(new RARef(a1Ref, 0, 0), 0, 0) { }
67
68      // Return ulCa as upper left and lrCa as lower right of (ca1, ca2)
69      public static void NormalizeArea(CellAddr ca1, CellAddr ca2,
70                                       out CellAddr ulCa, out CellAddr lrCa) {
71        int minCol = ca1.col, minRow = ca1.row,
72            maxCol = ca2.col, maxRow = ca2.row;
73        if (ca1.col > ca2.col) {
74          minCol = ca2.col; maxCol = ca1.col;
75        }
76        if (ca1.row > ca2.row) {
77          minRow = ca2.row; maxRow = ca1.row;
78        }
79        ulCa = new CellAddr(minCol, minRow);
80        lrCa = new CellAddr(maxCol, maxRow);
81      }
82
83      public CellAddr(System.Drawing.Point p) {
84        this.col = p.X;
85        this.row = p.Y;
86      }
87
88      public override String ToString() {
89        return ColumnName(col) + (row + 1);
90      }
91
92      public static String ColumnName(int col) {
93        String name = "";
94        while (col >= 26) {
95          name = (char)('A' + col % 26) + name;
96          col = col / 26 - 1;
97        }
98        return (char)('A' + col) + name;
99      }
100
101
102       public void Set(int c, int r)
```

```
103       {
104           throw new System.NotImplementedException();
105       }
106   }
107
108   // ------------------------------------------------------------
109   // A RARef represents a relative or absolute cell reference
110
111   public sealed class RARef {
112     public readonly bool colAbs, rowAbs;  // True=absolute, False=relative
113     public readonly int colRef, rowRef;
114
115     public RARef(bool colAbs, int colRef, bool rowAbs, int rowRef) {
116       this.colAbs = colAbs; this.colRef = colRef;
117       this.rowAbs = rowAbs; this.rowRef = rowRef;
118     }
119
120     // Parse "$A$3" to (true,0,true,2) and so on; relative references
121     // must be adjusted to the (col,row) in which the reference occurs.
122
123     public RARef(String a1Ref, int col, int row) {
124       /* RC denotes "this" cell in MS XMLSS format; needed for area support */
125       if (a1Ref.Equals("rc", StringComparison.CurrentCultureIgnoreCase)) {
126         colAbs = false; rowAbs = false; colRef = 0; rowRef = 0;
127       } else {
128         int i = 0;
129         if (i < a1Ref.Length && a1Ref[i] == '$') {
130           colAbs = true;
131           i++;
132         }
133         int val = -1;
134         while (i < a1Ref.Length && IsAToZ(a1Ref[i])) {
135           val = (val + 1) * 26 + AToZValue(a1Ref[i]);
136           i++;
137         }
138         colRef = colAbs ? val : val - col;
139         if (i < a1Ref.Length && a1Ref[i] == '$') {
140           rowAbs = true;
141           i++;
142         }
143         val = ParseInt(a1Ref, ref i);
144         rowRef = (rowAbs ? val : val - row) - 1;
145       }
146     }
147
148     // Parse XMLSS R1C1 notation from well-formed s (checked by parser)
149
150     public RARef(String r1c1) {
151       int i = 0;
152       rowAbs = true;
153       colAbs = true;
```

```
      if (i < r1c1.Length && r1c1[i] == 'R')
        i++;
      if (i < r1c1.Length && r1c1[i] == '[') {
        rowAbs = false;
        i++;
      }
      int val = ParseInt(r1c1, ref i);
      if (rowAbs && val == 0)
        rowAbs = false;
      this.rowRef = rowAbs ? val - 1 : val;
      if (i < r1c1.Length && r1c1[i] == ']')
        i++;
      if (i < r1c1.Length && r1c1[i] == 'C')
        i++;
      if (i < r1c1.Length && r1c1[i] == '[') {
        colAbs = false;
        i++;
      }
      val = ParseInt(r1c1, ref i);
      if (i < r1c1.Length && r1c1[i] == ']')
        i++;
      if (colAbs && val == 0)
        colAbs = false;
      this.colRef = colAbs ? val - 1 : val;
      Formats fo = new Formats();
      fo.RefFmt = Formats.RefType.R1C1;
    }

    // Parse possibly signed decimal integer
    private static int ParseInt(String s, ref int i) {
      int val = 0;
      bool negative = false;
      if (i < s.Length && (s[i] == '-' || s[i] == '+')) {
        negative = s[i] == '-';
        i++;
      }
      val = 0;
      while (i < s.Length && Char.IsDigit(s[i])) {
        val = val * 10 + (s[i] - '0');
        i++;
      }
      return negative ? -val : val;
    }

    private static bool IsAToZ(char c) {
      return 'a' <= c && c <= 'z' || 'A' <= c && c <= 'Z';
    }

    private static int AToZValue(char c) {
      return (c - 'A') % 32;
    }
```

```
205
206     // Absolute address of ref
207
208     public CellAddr Addr(int col, int row) {
209       return new CellAddr(this, col, row);
210     }
211
212     // Insert N new rowcols before rowcol R>=0, when we're at rowcol r
213     public Adjusted<RARef> InsertRowCols(int R, int N, int r, bool insertRow, out bool isBeforeInserted) {
214       int newRef;
215       int upper;
216       if (insertRow) {   // Insert N rows before row R; we're at row r
217         InsertRowCols(R, N, r, rowAbs, rowRef, out newRef, out upper, out isBeforeInserted);
218         RARef rarefNew = new RARef(colAbs, colRef, rowAbs, newRef);
219         return new Adjusted<RARef>(rarefNew, upper, rowRef == newRef);
220       } else {           // Insert N columns before column R; we're at column r
221         InsertRowCols(R, N, r, colAbs, colRef, out newRef, out upper, out isBeforeInserted);
222         RARef rarefNew = new RARef(colAbs, newRef, rowAbs, rowRef);
223         return new Adjusted<RARef>(rarefNew, upper, colRef == newRef);
224       }
225     }
226
227     // Insert N new rowcols before rowcol R>=0, when we're at rowcol r
228     private static void InsertRowCols(int R, int N, int r,
229                                       bool rcAbs, int rcRef,
230                                       out int newRc, out int upper, out bool isBeforeInserted)
231     {
232       if (rcAbs) {
233         if (rcRef >= R) {
234           // Absolute ref to cell after inserted          // Case (Ab)
235           newRc = rcRef + N;
236           upper = int.MaxValue;
237           isBeforeInserted = false;
238         } else {
239           // Absolute ref to cell before inserted       // Case (Aa)
240           newRc = rcRef;
241           upper = int.MaxValue;
242           isBeforeInserted = true;
243         }
244       } else // Relative reference
245           if (r >= R)
246           {
247               if (r + rcRef < R)
248               {
249                   // Relative ref from after inserted rowcols to cell before them
250                   newRc = rcRef - N;                          // Case (Rab)
251                   upper = R - rcRef;
252                   isBeforeInserted = true;
253               }
254               else
255               {
```

```
                    // Relative ref from after inserted rowcols to cell after them
                    newRc = rcRef;                                  // Case (Rbb)
                    upper = int.MaxValue;
                    isBeforeInserted = false;
                }

            }
            else // r < R
            {
                if (r + rcRef >= R)
                {
                    // Relative ref from before inserted rowcols to cell after them
                    newRc = rcRef + N;                              // Case (Rba)
                    upper = R;
                    isBeforeInserted = false;
                }
                else
                {
                    // Relative ref from before inserted rowcols to cell before them
                    newRc = rcRef;                                  // Case (Raa)
                    upper = Math.Min(R, R - rcRef);
                    isBeforeInserted = true;
                }

            }
        }

        // Clone and move (when the containing formula is moved, not copied)

        public RARef Move(int deltaCol, int deltaRow) {
          return new RARef(colAbs, colAbs ? colRef : colRef + deltaCol,
                           rowAbs, rowAbs ? rowRef : rowRef + deltaRow);
        }

        public String Show(int col, int row, Formats fo) {
          switch (fo.RefFmt) {
            case Formats.RefType.A1:
              CellAddr ca = new CellAddr(this, col, row);
              return (colAbs ? "$" : "") + CellAddr.ColumnName(ca.col)
                   + (rowAbs ? "$" : "") + (ca.row + 1);
            case Formats.RefType.R1C1:
              return "R" + RelAbsFormat(rowAbs, rowRef, 1)
                   + "C" + RelAbsFormat(colAbs, colRef, 1);
            case Formats.RefType.C0R0:
              return "C" + RelAbsFormat(colAbs, colRef, 0)
                   + "R" + RelAbsFormat(rowAbs, rowRef, 0);
            default:
              throw new ImpossibleException("Unknown reference format");
          }
        }

```

```
307     private static String RelAbsFormat(bool abs, int offset, int origo) {
308       if (abs)
309         return (offset + origo).ToString();
310       else if (offset == 0)
311         return "";
312       else
313         return "[" + (offset > 0 ? "+" : "") + offset.ToString() + "]";
314     }
315   }
316
317   // ------------------------------------------------------------------
318   // An Adjusted<T> represents an adjusted expression or
319   // a relative/absolute ref, for InsertRowCols
320
321   public struct Adjusted<T> {
322     public readonly T e;          // The adjusted Expr or RaRef
323     public readonly int upper;    // ... invalid for rows >= upper
324     public readonly bool same;    // Adjusted is identical to original
325
326     public Adjusted(T e, int upper, bool same) {
327       this.e = e;
328       this.upper = upper;
329       this.same = same;
330     }
331
332     public Adjusted(T e) : this(e, int.MaxValue, true) { }
333   }
334
335
336 }
337
```

CoreCalc

Expressions

```
// CoreCalc, a spreadsheet core implementation

// ------------------------------------------------------------------------
// Copyright (c) 2006 Peter Sestoft

// Permission is hereby granted, free of charge, to any person
// obtaining a copy of this software and associated documentation
// files (the "Software"), to deal in the Software without
// restriction, including without limitation the rights to use, copy,
// modify, merge, publish, distribute, sublicense, and/or sell copies
// of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:

//  * The above copyright notice and this permission notice shall be
//    included in all copies or substantial portions of the Software.

//  * The software is provided "as is", without warranty of any kind,
//    express or implied, including but not limited to the warranties of
//    merchantability, fitness for a particular purpose and
//    noninfringement.  In no event shall the authors or copyright
//    holders be liable for any claim, damages or other liability,
//    whether in an action of contract, tort or otherwise, arising from,
//    out of or in connection with the software or the use or other
//    dealings in the software.
// ------------------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Text;
using C5;
using CoreCalc.SupportGraf;

// Class Expr and its subclasses are used to recursively build formulas

namespace CoreCalc {
  // ---------------------------------------------------------------
  // Expressions are used to build formulas

  public abstract class Expr {
    // Update cell references when containing cell is moved (not copied)
    public abstract Expr Move(int deltaCol, int deltaRow);

    // Evaluate expression as if at cell address sheet[col, row]
    public abstract Value Eval(Sheet sheet, int col, int row);

    // Apply act to result of evaluating expression at sheet[col, row]
    public abstract void Apply(Sheet sheet, int col, int row, Act<Value> act);

    // Insert N new rows before row R
    public abstract Adjusted<Expr> InsertRowCols(Sheet modSheet, bool thisSheet, int R, int N, int r, bool doRows, ref bool
    coveringInsertedRows);
```

```
51
52        internal abstract void GetAdresses<T, Y>(IAdressCollector<Y> collector, ref T adresses) where T : C5.ICollection<Y>;
53
54      // Show contents as expression
55      public abstract String Show(int col, int row, int ctxpre, Formats fo);
56    }
57
58    // ------------------------------------------------------------------
59    // A Const expression is a constant (immutable, sharable)
60
61    abstract class Const : Expr {
62      public override Expr Move(int deltaCol, int deltaRow) {
63        return this;
64      }
65
66      public override Adjusted<Expr> InsertRowCols(Sheet modSheet, bool thisSheet, int R, int N, int r, bool doRows, ref bool
        coveringInsertedRows)
67      {
68        coveringInsertedRows   false;
69        return new Adjusted<Expr>(this);
70      }
71    }
72
73    // ------------------------------------------------------------------
74    // A NumberConst is a constant expression
75
76    class NumberConst : Const {
77      private readonly NumberValue value;
78
79      public NumberConst(double d) {
80        value   new NumberValue(d);
81      }
82
83      public override Value Eval(Sheet sheet, int col, int row) {
84        return value;
85      }
86
87      public override void Apply(Sheet sheet, int col, int row, Act<Value> act) {
88        act(value);
89      }
90
91      public override String Show(int col, int row, int ctxpre, Formats fo) {
92        return value.ToString();
93      }
94
95        internal override void GetAdresses<T,Y>(IAdressCollector<Y> collector, ref T adresses)
96        {
97            collector.GetAdresses(this, ref  adresses);
98        }
99    }
100
```

```
101   // ------------------------------------------------------------------
102   // A TextConst is a constant string-valued expression
103
104   class TextConst : Const {
105     private readonly TextValue value;
106
107     public TextConst(String s) {
108       value   new TextValue(s);
109     }
110
111     public override Value Eval(Sheet sheet, int col, int row) {
112       return value;
113     }
114
115     public override void Apply(Sheet sheet, int col, int row, Act<Value> act) {
116       act(value);
117     }
118
119     public override String Show(int col, int row, int ctxpre, Formats fo) {
120       return "\"" + value.ToString() + "\"";
121     }
122
123       internal override void GetAdresses<T, Y>(IAdressCollector<Y> collector, ref T adresses)
124       {
125           collector.GetAdresses(this, ref adresses);
126       }
127   }
128
129   // ------------------------------------------------------------------
130   // An Error expression represents a static error, e.g. invalid reference
131
132   class Error : Expr {
133     private readonly String error;
134     private readonly ErrorValue value;
135
136     public Error(String s) {
137       value   new ErrorValue(s);
138       error   "#" + s;
139     }
140
141     public override Value Eval(Sheet sheet, int col, int row) {
142       return value;
143     }
144
145     public override void Apply(Sheet sheet, int col, int row, Act<Value> act) {
146       act(value);
147     }
148
149     public override String Show(int col, int row, int ctxpre, Formats fo) {
150       return error;
151     }
```

```
152
153     public override Expr Move(int deltaCol, int deltaRow) {
154       return this;
155     }
156
157     public override Adjusted<Expr> InsertRowCols(Sheet modSheet, bool thisSheet, int R, int N, int r, bool doRows, ref bool
        coveringInsertedRows)
158     {
159        coveringInsertedRows   false;
160       return new Adjusted<Expr>(this);
161     }
162
163       internal override void GetAdresses<T, Y>(IAdressCollector<Y> collector, ref T adresses)
164       {
165           collector.GetAdresses(this, ref adresses);
166       }
167   }
168
169   // ------------------------------------------------------------------
170   // A FunCall is an operator application such as 1+$A$4 or a function
171   // call such as RAN () or SIN(4*A$7) or SUM(B4:B52; 3) or IF(A1; A2; 1/A1)
172
173   class FunCall : Expr {
174     public readonly Function function;   // Non-null
175     public readonly Expr[] es;           // Non-null, elements non-null
176
177     public FunCall(Function function, Expr[] es) {
178       if (function    null || es    null
179           || Array.Exists(es, delegate(Expr e) { return e    null; }))
180         throw new Exception("Function or argument null");
181       else {
182         this.function   function;
183         this.es   es;
184       }
185     }
186
187     public FunCall(String name, Expr[] es)
188       : this(Function.Get(name), es) { }
189
190     // Arguments are passed unevaluated to cater for non-strict IF
191
192     public override Value Eval(Sheet sheet, int col, int row) {
193       return function.applier(sheet, es, col, row);
194     }
195
196     public override void Apply(Sheet sheet, int col, int row, Act<Value> act) {
197       Value v   function.applier(sheet, es, col, row);
198       if (v is NumberValue || v is TextValue)
199         act(v);
200       else if (v is MatrixValue)
201         (v as MatrixValue).Apply(act);
```

```
202       else if (v !  null)
203         throw new ArgumentException();   // Non-null but wrong type
204     }
205
206     public override Expr Move(int deltaCol, int deltaRow) {
207       Expr[] newEs   new Expr[es.Length];
208       for (int i   0; i < es.Length; i++)
209         newEs[i]   es[i].Move(deltaCol, deltaRow);
210       return new FunCall(function, newEs);
211     }
212
213     public override Adjusted<Expr> InsertRowCols(Sheet modSheet, bool thisSheet, int R, int N, int r, bool doRows, ref bool
      coveringInsertedRows)
214     {
215       Expr[] newEs   new Expr[es.Length];
216       int upper   int.MaxValue;
217       bool same   true;
218       coveringInsertedRows   false;
219       for (int i   0; i < es.Length; i++)
220       {
221         Adjusted<Expr> ae   es[i].InsertRowCols(modSheet, thisSheet, R, N, r, doRows, ref coveringInsertedRows);
222         upper   Math.Min(upper, ae.upper);
223         same   same && ae.same;
224         newEs[i]   ae.e;
225       }
226       return new Adjusted<Expr>(new FunCall(function, newEs), upper, same);
227     }
228
229     // Show infixed operators as infix and without excess parentheses
230
231     public override String Show(int col, int row, int ctxpre, Formats fo) {
232       StringBuilder sb   new StringBuilder();
233       int pre   function.fixity;
234       if (pre    0) { // Not operator
235         sb.Append(function.name).Append("(");
236         for (int i   0; i < es.Length; i++) {
237           if (i > 0)
238             sb.Append(";");
239           sb.Append(es[i].Show(col, row, 0, fo));
240         }
241         sb.Append(")");
242       }
243       else { // Operator.  Assume es.Length is 1 or 2
244         if (es.Length    2) {
245           // If precedence lower than context, add parens
246           if (pre < ctxpre)
247             sb.Append("(");
248           sb.Append(es[0].Show(col, row, pre, fo));
249           sb.Append(function.name);
250           // Only higher precedence right operands avoid parentheses
251           sb.Append(es[1].Show(col, row, pre + 1, fo));
```

```
252           if (pre < ctxpre)
253             sb.Append(")");
254         }
255         else if (es.Length    1) {
256           sb.Append(function.name);
257           sb.Append(es[0].Show(col, row, pre, fo));
258         }
259         else
260           throw new ImpossibleException("Operator not unary or binary");
261       }
262       return sb.ToString();
263     }
264
265       internal override void GetAdresses<T, Y>(IAdressCollector<Y> collector, ref T adresses)
266       {
267           collector.GetAdresses(this, ref adresses);
268       }
269   }
270
271   // ------------------------------------------------------------
272   // A CellRef expression is A1 or $A1 or A$1 or $A$1 or Sheet1!A1
273
274   class CellRef : Expr {
275     public readonly RARef raref;
276     internal readonly Sheet sheet;    // non-null if sheet-absolute
277
278     public CellRef(Sheet sheet, RARef raref) {
279       this.sheet   sheet;
280       this.raref   raref;
281     }
282
283     public CellRef(Sheet sheet, bool colAbs, int colRef, bool rowAbs, int rowRef)
284       : this(sheet, new RARef(colAbs, colRef, rowAbs, rowRef)) {
285     }
286
287     // Evaluate cell ref by evaluating the cell referred to
288
289     public override Value Eval(Sheet sheet, int col, int row) {
290         if (this.sheet !  null)
291             sheet   this.sheet;
292         CellAddr ca   raref.Addr(col, row);
293         if (sheet.Rows > ca.row && sheet.Cols > ca.col)
294         {
295             Cell cell   sheet[ca];
296             if (cell    null)
297                 return null;
298             if (sheet.workbook.UseSupportGraph)
299                 return cell.GetValue(sheet, ca.col, ca.row);
300             else
301             {
302                 cell.Eval(sheet, ca.col, ca.row);
```

```
303                 return cell.GetValue(sheet, ca.col, ca.row);
304             }
305         }
306         else return null;
307
308     }
309
310     // Apply to cell ref by applying to the cell referred to
311
312     public override void Apply(Sheet sheet, int col, int row, Act<Value> act) {
313       if (this.sheet !  null)
314         sheet   this.sheet;
315       CellAddr ca   raref.Addr(col, row);
316       Cell cell   sheet[ca];
317       if (cell !  null)
318         cell.Apply(sheet, ca.col, ca.row, act);
319     }
320
321     // Clone and move (when the containing formula is moved, not copied!)
322
323     public override Expr Move(int deltaCol, int deltaRow) {
324       return new CellRef(sheet, raref.Move(deltaCol, deltaRow));
325     }
326
327     public override Adjusted<Expr> InsertRowCols(Sheet modSheet, bool thisSheet, int R, int N, int r, bool doRows, ref bool
      coveringInsertedRows)
328     {
329         if (sheet    modSheet || sheet    null && thisSheet)
330         {
331             bool noUse;
332             Adjusted<RARef> adj   raref.InsertRowCols(R, N, r, doRows, out noUse);
333             return new Adjusted<Expr>(new CellRef(sheet, adj.e),
334                                       adj.upper, adj.same);
335         }
336         else
337         {
338             coveringInsertedRows   false;
339             return new Adjusted<Expr>(this);
340         }
341     }
342
343     public override String Show(int col, int row, int ctxpre, Formats fo) {
344       String s   raref.Show(col, row, fo);
345       return sheet    null ? s : sheet.Name + "!" + s;
346     }
347
348       internal override void GetAdresses<T, Y>(IAdressCollector<Y> collector, ref T adresses)
349       {
350           collector.GetAdresses(this, ref adresses);
351       }
352   }
```

```
353
354   // ---------------------------------------------------------------
355   // A CellArea is an expression such as
356   // A1:C4 or $A$1:C4 or A1:$C4 or Sheet1!A1:C4
357
358   class CellArea : Expr {
359     public readonly RARef ul, lr;  // upper left, lower right
360     public Sheet sheet;   // non-null if sheet-absolute
361
362     public CellArea(Sheet sheet,
363                     bool ulColAbs, int ulColRef, bool ulRowAbs, int ulRowRef,
364                     bool lrColAbs, int lrColRef, bool lrRowAbs, int lrRowRef)
365       : this(sheet,
366              new RARef(ulColAbs, ulColRef, ulRowAbs, ulRowRef),
367              new RARef(lrColAbs, lrColRef, lrRowAbs, lrRowRef)) {
368     }
369
370     public CellArea(Sheet sheet, RARef ul, RARef lr) {
371       this.sheet   sheet;
372       this.ul   ul;
373       this.lr   lr;
374     }
375
376     // Evaluate cell area by evaluating every cell in the area
377
378     public override Value Eval(Sheet defaultSheet, int col, int row) {
379       CellAddr ulCa   ul.Addr(col, row), lrCa   lr.Addr(col, row);
380       CellAddr.NormalizeArea(ulCa, lrCa, out ulCa, out lrCa);
381       int cols   lrCa.col - ulCa.col + 1, rows   lrCa.row - ulCa.row + 1;
382       int col0   ulCa.col, row0   ulCa.row;
383       Value[,] values   new Value[cols, rows];
384       Sheet useSheet   this.sheet    null ? defaultSheet : this.sheet;
385       for (int c   0; c < cols; c++)
386         for (int r   0; r < rows; r++) {
387             if (useSheet.Cols > col0 + c && useSheet.Rows > row0 + r)//Made it put null in values outside the sheet
388             {
389                 Cell cell   useSheet[col0 + c, row0 + r];
390                 if (cell !  null)
391                 {
392                     if (defaultSheet.workbook.UseSupportGraph)
393                         values[c, r]   cell.GetValue(useSheet, col0 + c, row0 + r);
394                     else
395                     {
396                         cell.Eval(useSheet, col0 + c, row0 + r);
397                         values[c, r]   cell.GetValue(useSheet, col0 + c, row0 + r);
398                     }
399                 }
400             }
401         }
402       return new MatrixValue(values);
403     }
```

```
404
405     // Apply to cell area by applying to every cell in the area
406
407     public override void Apply(Sheet sheet, int col, int row, Act<Value> act) {
408       CellAddr ulCa   ul.Addr(col, row), lrCa   lr.Addr(col, row);
409       CellAddr.NormalizeArea(ulCa, lrCa, out ulCa, out lrCa);
410       int cols   lrCa.col - ulCa.col + 1, rows   lrCa.row - ulCa.row + 1;
411       int col0   ulCa.col, row0   ulCa.row;
412       for (int c   0; c < cols; c++)
413         for (int r   0; r < rows; r++) {
414           Cell cell   sheet[col0 + c, row0 + r];
415           if (cell !  null)
416             cell.Apply(sheet, col0 + c, row0 + r, act);
417         }
418     }
419
420     // Clone and move (when the containing formula is moved, not copied)
421
422     public override Expr Move(int deltaCol, int deltaRow) {
423       return new CellArea(sheet,
424                           ul.Move(deltaCol, deltaRow),
425                           lr.Move(deltaCol, deltaRow));
426     }
427
428     public override Adjusted<Expr> InsertRowCols(Sheet modSheet, bool thisSheet, int R, int N, int r, bool doRows, ref bool
        coveringInsertedRows)
429     {
430         if (sheet    modSheet || sheet    null && thisSheet)
431         {
432             bool ulBefore;
433             bool lrBefore;
434             Adjusted<RARef> ulNew   ul.InsertRowCols(R, N, r, doRows, out ulBefore),
435                             lrNew   lr.InsertRowCols(R, N, r, doRows, out lrBefore);
436             //if ul is before and lr is after this covers the inserted rows
437             coveringInsertedRows   ulBefore && !lrBefore;
438             int upper   Math.Min(ulNew.upper, lrNew.upper);
439             return new Adjusted<Expr>(new CellArea(sheet, ulNew.e, lrNew.e),
440                                       upper, ulNew.same && lrNew.same);
441         }
442         else
443         {
444             coveringInsertedRows   false;
445             return new Adjusted<Expr>(this);
446         }
447     }
448
449     public override String Show(int col, int row, int ctxpre, Formats fo) {
450       String s   ul.Show(col, row, fo) + ":" + lr.Show(col, row, fo);
451       return sheet    null ? s : sheet.Name + "!" + s;
452     }
453
```

```
454         internal override void GetAdresses<T, Y>(IAdressCollector<Y> collector, ref T adresses)
455         {
456             collector.GetAdresses(this, ref adresses);
457         }
458     }
459 }
460
```

CoreCalc

Program

```
// CoreCalc, a spreadsheet core implementation

// ------------------------------------------------------------------------
// Copyright (c) 2006 Peter Sestoft and Thomas S. Iversen

// Permission is hereby granted, free of charge, to any person
// obtaining a copy of this software and associated documentation
// files (the "Software"), to deal in the Software without
// restriction, including without limitation the rights to use, copy,
// modify, merge, publish, distribute, sublicense, and/or sell copies
// of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:

//  * The above copyright notice and this permission notice shall be
//    included in all copies or substantial portions of the Software.

//  * The software is provided "as is", without warranty of any kind,
//    express or implied, including but not limited to the warranties of
//    merchantability, fitness for a particular purpose and
//    noninfringement.  In no event shall the authors or copyright
//    holders be liable for any claim, damages or other liability,
//    whether in an action of contract, tort or otherwise, arising from,
//    out of or in connection with the software or the use or other
//    dealings in the software.
// ------------------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.IO;

// The main program; start the command line version or the GUI version

namespace CoreCalc {
  class Program {
    [STAThread]
    public static void Main(String[] args) {
        Console.SetWindowSize(70, 5 );
        Console.SetWindowPosition(0, 0);
        Console.SetBufferSize(70, 5000);
      if (args.Length == 0) {
        // GUI, with new empty workbook
        WorkbookForm gui = new WorkbookForm(new Workbook());

        return;
      } else if (args.Length == 1) {
        FileInfo fi = new FileInfo(args[0]);
        Logger.Log.Debug(typeof(Program), fi);
        switch (fi.Extension) {
          case ".xml":            // Open existing workbook in GUI
            WorkBookIO workbookio = new WorkBookIO();
            WorkbookForm gui = new WorkbookForm(workbookio.Read(fi.Name));
```

```
52              return;
53            case ".cs":             // Run script
54              TinyScript tinyscript = new TinyScript(fi.Name, true);
55              tinyscript.Run(new String[0]);
56              return;
57            default:
58              break;
59          }
60        }
61        Logger.Log.Debug(typeof(Program), "Usage: CoreCalc [workbook.xml|script.cs]\n");
62      }
63    }
64  }
65
```

CoreCalc

Values

```
// CoreCalc, a spreadsheet core implementation

// ------------------------------------------------------------------------
// Copyright (c) 2006 Peter Sestoft

// Permission is hereby granted, free of charge, to any person
// obtaining a copy of this software and associated documentation
// files (the "Software"), to deal in the Software without
// restriction, including without limitation the rights to use, copy,
// modify, merge, publish, distribute, sublicense, and/or sell copies
// of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:

//  * The above copyright notice and this permission notice shall be
//    included in all copies or substantial portions of the Software.

//  * The software is provided "as is", without warranty of any kind,
//    express or implied, including but not limited to the warranties of
//    merchantability, fitness for a particular purpose and
//    noninfringement.  In no event shall the authors or copyright
//    holders be liable for any claim, damages or other liability,
//    whether in an action of contract, tort or otherwise, arising from,
//    out of or in connection with the software or the use or other
//    dealings in the software.
// ------------------------------------------------------------------------

using System;
using System.Collections;
using System.Text;

// Class Value and its subclasses are used to represent the possible
// values of a spreadsheet cell

namespace CoreCalc {

  // A Value is the result of evaluating an expression
  public enum CompareResult {ThisIsSmallest, OtherIsSmallest, Equal, DifferentOrIncompatibleTypes}

  public abstract class Value
  {
      public CompareResult CompareToOther(Value other)
      {

          NumberValue thisNumber = this as NumberValue;
          if (thisNumber != null)
          {
              NumberValue otherNumber;
              if(other!= null)
                otherNumber = other as NumberValue;
              else
                  otherNumber = new NumberValue(0.0);
```

```
52              if (otherNumber != null)
53              {
54                  if (thisNumber.value < otherNumber.value)
55                      return CompareResult.ThisIsSmallest;
56                  else if (thisNumber.value > otherNumber.value)
57                      return CompareResult.OtherIsSmallest;
58                  else
59                      return CompareResult.Equal;
60              }
61              else
62                  return CompareResult.DifferentOrIncompatibleTypes;
63          }
64          else
65          {
66              TextValue thisText = this as TextValue;
67              if (thisText != null)
68              {
69                  TextValue otherText;
70                  if(other != null)
71                    otherText = other as TextValue;
72                  else
73                      otherText = new TextValue(string.Empty);
74                  if (otherText != null)
75                  {
76                      int compareResult = thisText.value.CompareTo(otherText.value);
77                      if (compareResult < 0)
78                          return CompareResult.ThisIsSmallest;
79                      else if (compareResult > 0)
80                          return CompareResult.OtherIsSmallest;
81                      else
82                          return CompareResult.Equal;
83
84                  }
85                  else
86                      return CompareResult.DifferentOrIncompatibleTypes;
87              }
88              else if(this is NullValue && other is NullValue)
89              {
90                  return CompareResult.Equal;
91              }
92              else
93                  return CompareResult.DifferentOrIncompatibleTypes;
94          }
95      }
96  }
97
98    class NullValue : Value
99    {
100         public NullValue()
101         {}
102
```

```
103         public override String ToString()
104         {
105             return "Null Value";
106         }
107     }
108
109   // ------------------------------------------------------------------
110   // A NumberValue is a floating-point number
111
112   class NumberValue : Value {
113     public readonly double value;
114
115     public NumberValue(double value) {
116       this.value = value;
117     }
118
119     public override String ToString() {
120       return value.ToString();
121     }
122   }
123
124   // ------------------------------------------------------------------
125   // A TextValue is a string
126
127   class TextValue : Value {
128     public readonly String value;
129
130     public TextValue(String value) {
131       this.value = value;
132     }
133
134     public override String ToString() {
135       return value;
136     }
137   }
138
139   // ------------------------------------------------------------------
140   // An ErrorValue is a value indicating failure of evaluation
141
142   class ErrorValue : Value {
143     private readonly String msg;
144
145     public ErrorValue(String msg) {
146       this.msg = "#ERR: " + msg;
147     }
148
149     public override String ToString() {
150       return msg;
151     }
152   }
153
```

```
154   // ------------------------------------------------------------------
155   // A MatrixValue is a rectangle of Values
156
157   class MatrixValue : Value, IEnumerable {
158     private readonly Value[,] values;   // Non-null; elements may be null
159
160     public MatrixValue(Value[,] values) {
161       this.values = values;
162     }
163
164     public int Cols {
165       get { return values.GetLength(0); }
166     }
167
168     public int Rows {
169       get { return values.GetLength(1); }
170     }
171
172     // Get cell value from matrix value
173     public Value this[CellAddr ca] {
174       get {
175         //    0 <= ca.col && ca.col < Cols
176         // && 0 <= ca.row && ca.row < Rows
177         return values[ca.col, ca.row];
178       }
179     }
180
181     public Value this[int col, int row] {
182       get {
183         //    0 <= col && col < Cols
184         // && 0 <= row && row < Rows
185         return values[col, row];
186       }
187     }
188
189     public void Apply(Act<double> act) {
190         foreach (Value v in values)
191             if (v != null) // Only active cells contribute
192                 if (v is NumberValue)
193                     act((v as NumberValue).value);
194                 else if (v is MatrixValue)
195                     (v as MatrixValue).Apply(act);
196                 else if (v is ErrorValue)
197                     throw new ErrorValueException(v as ErrorValue);
198                 else if (!(v is NullValue))//if nullvalue just ignore
199                     throw new Exception("MatrixValue.Apply");
200     }
201
202       public void Apply(Act<string> act)
203       {
204           foreach (Value v in values)
```

```
205                 if (v != null) // Only active cells contribute
206                     if (v is TextValue)
207                         act((v as TextValue).value);
208                     else if (v is MatrixValue)
209                         (v as MatrixValue).Apply(act);
210                     else if (v is ErrorValue)
211                         throw new ErrorValueException(v as ErrorValue);
212                     else
213                         throw new Exception("MatrixValue.Apply");
214         }
215 
216         public void Apply(Act<Value> act) {
217             foreach (Value v in values)
218                 if (v != null) // Only active cells contribute
219                     if (v is NumberValue || v is TextValue)
220                         act(v);
221                     else if (v is MatrixValue)
222                         (v as MatrixValue).Apply(act);
223                     else if (v is ErrorValue)
224                         throw new ErrorValueException(v as ErrorValue);
225                     else
226                         throw new Exception("MatrixValue.Apply");
227         }
228 
229         public override String ToString() {
230             StringBuilder sb = new StringBuilder();
231             for (int c = 0; c < Cols; c++) {
232                 for (int r = 0; r < Rows; r++) {
233                     Value v = values[c, r];
234                     sb.Append(v == null ? "[none]" : v.ToString());
235                 }
236                 sb.Append("\n");
237             }
238             return sb.ToString();
239         }
240 
241         public IEnumerator GetEnumerator()
242         {
243             return values.GetEnumerator();
244         }
245     }
246 }
247 
```