Handin for
A Derivational Approach to Calculi, Evaluation, and
Abstract Machines

Mikkel Bundgaard

August 18, 2006

Contents

1 Problem Description 2
  1.1 Simplifications and Assumptions ........................................... 3

2 Solution 3

3 The algorithm 5
  3.1 Direct Version ................................................................. 5
  3.2 CPS Version ................................................................. 8
  3.3 Defunctionalised Version .................................................. 10
  3.4 As a Transition System .................................................. 14
1 Problem Description

In this report we examine the transformation of an big-step relation, for finding matches in place graphs, into a small-step transition relation which solves the same task.

Static Structure of Bigraphs  For brevity we will not present the full definition of bigraphs but instead refer the reader to [2, 1] for the full definition and examples.

We will however briefly (and somewhat informally) introduce one constituent of a bigraph: the place graph. In Fig. 1 (top) we see a bigraph. A bigraph can be thought of as a multi-hole context and it has two constituents: the place graph and the link graph, representing the topology and the connectivity of the bigraph, respectively. The bigraph in Fig. 1 (top) is represented by these two structures (depicted below) which shares the same set of nodes. The place graph represents the nesting of nodes, and the link graph represents the linkage of nodes. The place graph is essentially an ordered forest of unordered trees. The gray holes in bigraph represents holes (also called sites) in the bigraph in which we can place other bigraphs.

Every node in a bigraph is assigned a control which can be though of as the type of the node. A given control induce several properties on the nodes of its kind. However in this presentation we will only distinguish between non-atomic and atomic controls, controlling whether a node can contain other nodes or not. We will often abuse notation and confuse the node with its control and vice versa, but note that nodes in a bigraph are pairwise distinct whereas in general this is not the case of controls.

![Bigraph Diagram](image)

Figure 1: Bigraphs
Dynamics The dynamics of bigraphs, i.e. the reconfigurations that may occur, depend upon both structural components; however as mentioned before we will only work with one component in this report. The reconfigurations are determined by reaction rules, and each reaction rule has a \textit{redex} and a \textit{reactum}. The redex is a precondition for a reaction, and is represented by a pattern of nesting (in the case of place graphs). We represent a \textit{reaction rule} as a pair \((r, r')\) of \textit{ground} place graphs\(^1\). Given a ground place graph \(a\) (called the \textit{agent}) and a reaction rule \((r, r')\) we try to calculate a context \(C\) such that \(a = C \circ r\) (that is \(a\) is equal to the composition of \(C\) and \(r\), i.e. that we can find the redex within the agent). If such a \(C\) exists we can then calculate the ground place graph \(a'\), the result of the rewriting of \(a\) using \((r, r')\), as \(a' = C \circ r'\). However we will only focus on the task to calculate \(C\) given an agent \(a\) and a redex \(r\).

1.1 Simplifications and Assumptions

In order to simplify the task (and to keep the implementation simple) we have chosen to make the following simplifications and assumptions:

- Both the \textit{agent} and the redex are \textit{ground}. This corresponds to the basic dynamics in place graphs [2]. This implies that the only place where holes can occur is in \(C\).
- Both agent and redex are \textit{prime}, meaning that we only consider place graphs which are unordered trees, hence forgetting the ordered forest on top-level. This greatly simplifies the function for finding matches.
- In the full presentation of bigraphs we distinguish between active and passive controls (controlling whether rewrites can occur within a node or not), we will not make such a distinction in this report. For simplicity we will assume that all controls are active, hence rewrites can occur everywhere in the place graph.
- We only return one match and not all possible matches (assuming that the agent and the redex matches at all).
- To simplify the implementation of matching considerably, we will not allow more than two children of any node. The unordered element of place graphs can then be handled by swapping the two children instead of permutations of arbitrary size.

2 Solution

First we begin by giving some simple examples of matches. Then we explain the actual representation of place graphs, the overview of the matching function, and finally the actual function.

Some Examples Let \(a\) be the following place graph with an outer node \(v0\), containing the node \(v1\), which in turn contains the node \(v2\), which contains the atomic node \(v3\). Let the redex \(r\) be a node \(v2\) containing a atomic node \(v3\). Since we need to find a \(C\) such that \(a = C \circ r\) the only possible solution is to let \(C\) be the place graph with an outer node \(v0\), containing the node \(v1\), which in turn contains a site, in which we can place \(r\).

\(^1\)A place graph is called \textit{ground} if it contains no holes.
The agent $a$

\[
\begin{array}{ccc}
v_0 & & v_1 \\
\mid & \mid & \mid \\
v_2 & & v_3 \\
\mid & & v_2
\end{array}
\]  

The result $C$

\[
\begin{array}{ccc}
\mid & v_0 \\
v_1 & \mid & v_1 \\
\mid & \mid & Site 0
\end{array}
\]

Let us consider a more complex example, where we have a node of control $v_0$ with two children of controls $v_1$ and $v_2$, respectively. We want to match this with the redex $r$ which is just two nodes of controls $v_1$ and $v_2$ in parallel (recall that the children are unordered). The result of this is $C$ which is just the node of control $v_0$ containing the hole.

\[
\begin{array}{ccc}
v_0 & & v_1 \\
\mid & \mid & \mid \\
v_2 & & v_3 \\
\mid & \mid & v_2
\end{array}
\]  

The result $C$

\[
\begin{array}{ccc}
\mid & v_0 \\
v_1 & \mid & v_1 \\
\mid & \mid & Site 0
\end{array}
\]

**Concrete Representation** We use the following datatype for representing a place graph. A place graph can be:

- a parallel composition of two place graphs;
- a control with a name containing a place graph;
- an atomic control with a name;
- a hole in the bigraph (which normally are numbered, but we will not use this feature in this report as only one hole will occur as result of a matching).

**Listing 1: The PlaceGraph Datatype**

```python
datatype PlaceGraph =
    Par of PlaceGraph * PlaceGraph
  | Control of string * PlaceGraph
  | Atomic of string
  | Site of int
```

For the result of a matching we need the following “extended” option type. Besides the need to know whether we have a match or not, we need to know whether we have matched the agent and redex directly, or if there is context surrounding redex in the match. We will need this knowledge when matching parallel compositions.

**Listing 2: The Value Datatype**

```python
datatype Value =
    None
  | Match of PlaceGraph
  | Context of PlaceGraph
```
3 The algorithm

Before we present the actual function we first explain the general idea. The function is going to take two place graphs as arguments and return a value. The function recursively descends the structure of the agent.

- If the agent is an atomic control we need that the redux is also an atomic control of same sort. If this is the case we return a match (i.e. `Site 0`) otherwise we return none.

- If the agent is a control there are several cases.

  If the redux is not a control we match below the control in the agent and put the control on top of the result of this matching.

  The same is the case if the redux is a control, but of different kind.

  If the redux is a control of the same sort we first try to match below the control as in the previous cases, as we search for the bottom-most match. If this does not succeed we then try to match where we have peeled off the top-level control in both the agent and the redux.

- If the agent is a parallel composition there are several cases.

  Either we can match the entire redux in either branch and then put the remaining branch in parallel when returning the result.

  If the redux is also a parallel composition we can also match one part of the redux in one part of the agent and match the remaining part of the redux in the remaining part. However we must match on top-level in both branches for this to work.

- We leave the case when the agent is a site undefined as this breaks with our assumption about the agent.

3.1 Direct Version

For readability we have chosen to split up the function in several parts. The function has the following signature.

```plaintext
matchGroundPGG : PlaceGraph -> PlaceGraph -> Value
```

Listing 3: Case Control

```plaintext
fun matchGroundPGG agent redux =
  case agent
  (*/ We the agent is a control on top-level */)
  of Control (ctag, pgag) => (case redux
    of Control (ctred, pgre) => (if ctag = ctred then (*/ First try to match the redux further down, even if */)
    (*/ this control matches the top of the redux */)
    case matchGroundPGG pgag redux of
      Match pgres => Context (Control (ctag, pgres])
      | Context pgres => Context (Control (ctag, pgres])
      | None => (*/ Try to match when this control in the agent */)
      (*/ should match this control in the redux *)
  ```
3 THE ALGORITHM

case matchGroundPGPG pgag pgred of
  Match _ => Match (Site 0)
  (/* This case can not happen */)  
  (/* | Context _ => None */)  
  | None => None

(* We have two controls but of different kinds *)
(* So we add the current control as context on *)
(* the possible match *)
else (  
  case matchGroundPGPG pgag redex of
    Match pgres => Context (Control (ctlag, pgres))
    | Context pgres => Context (Control (ctlag, pgres))
    | None => None

(* The redex is not a control on top-level, so we add *)
(* the agent control as context *)
| _ => case matchGroundPGPG pgag redex of
    Match pgres => Context (Control (ctlag, pgres))
    | Context pgres => Context (Control (ctlag, pgres))
    | None => None

Listing 4: Case Par

(* In a parallel composition in the redex we can match *)
(* the redex in either branch *)
| Par (pglag, pg2ag) => (  (* Match entire redex in left branch *)
  case matchGroundPGPG pflag redex
    of Match pgres => Context (Par (pgres, pg2ag))
    | Context pgres => Context (Par (pgres, pg2ag))
    | None => (  (* Match entire redex in right branch *)
  case matchGroundPGPG pg2ag redex
    of Match pgres => Context (Par (pglag, pgres))
    | Context pgres => Context (Par (pglag, pgres))
    | None => ( (* Match one part of the redex in one branch *)
    (* and the other in the other branch *)
  case redex
    of Par (pg1re, pg2re) => (  (* Permute matching in branches *)
    case matchGroundPGPG pflag pg1re of
      Match _ => (  
        case matchGroundPGPG pg2ag pg2re of
          Match _ => Match (Site 0)
          | _ => None
        )
        | (* Permute matching in branches *)
    case matchGroundPGPG pflag pg2re of
      Match _ => (  

```scala
3 THE ALGORITHM

```case``` matchGroundPGPG pg2ag pglre of
  Match _ => Match (Site 0)
  | _ => None
)
)
)
)
)
)

Listing 5: Case Atomic

```scala
(* An atomic control can only be matched by another atomic *)
(* control of same kind *)
| Atomic strag => ```case``` redux of Atomic sttre => (```case``` redux
  if strag = sttre then Match (Site 0)
  else None
)
  _ => None
)

Test Cases In order to check the correctness of the algorithm we use the following test cases. The test cases are far from complete, but due to time and space constraints we will not perform any more test.

Listing 6: Test Cases

```scala
val example = Control ("v0", Control ("v1", Control ("v2", Atomic("v3"))));
val redux = Control ("v2", Atomic("v3"));
val example2a = Par (Atomic("v1"), Atomic("v2"));
val example2b = Control ("v0", Par (Atomic("v1"), Atomic("v2")));
val redux2 = Par (Atomic("v2"), Atomic("v1"));
val example3 = Control ("v2", Control ("v1", Control ("v2", Atomic("v3"))));
val redux3 = Control ("v2", Atomic("v3"));
val example4 = Par (Control("v0", Atomic("v1")), Atomic("v2"));
val redux4 = Par (Atomic("v2"), Atomic("v1"));

val test1 = matchGroundPGPG example redux =
  Context (Control ("v0", Control ("v1", Site 0)));
val test2a = matchGroundPGPG example2a redux2 =
  Match (Site 0);
val test2b = matchGroundPGPG example2b redux2 =
  Context (Control ("v0", Site 0));
val test3 = matchGroundPGPG example3 redux3 =
  Context (Control ("v2", Control ("v1", Site 0)));
val test4 = matchGroundPGPG example4 redux4 = None;
...
```
3 THE ALGORITHM

val test1 = true : bool
val test2a = true : bool
val test2b = true : bool
val test3 = true : bool
val test4 = true : bool

3.2 CPS Version

We transform the function from Sec. 3.1 into continuation-passing style using the general template: we give the function an explicit continuation as an additional argument; we transform the function so instead of returning a value it is passed to the continuation argument; and when a recursive call is performed within the function we need to supply a function (the continuation) which will be invoked with the return value of the function. Again, for readability we have chosen to split up the function in several parts. The function has the following signature.

matchGroundPGPG : PlaceGraph -> PlaceGraph -> (Value -> Value) -> Value

Listing 7: Case Control

fun matchGroundPGPGCPS agent redux c =
  case agent
    of Control (ctlag, pgag) => ( redux
        of Control (ctlred, pgred) => ( redux
            if ctitlag = ctitlred then ( redux
                (* First try to match the redux further down, even if *) redux
                (* this control matches the top of the redux *) redux
                matchGroundPGPGCPS pgag redux ( redux
                    fn v => case v of redux
                        Match pgres => c (Context (Control (ctlag, pgres))) redux
                        | Context pgres => c (Context (Control (ctlag, pgres))) redux
                        | None => ( redux
                            (* Try to match when this control in the agent *) redux
                            (* should match this control in the redux *) redux
                            matchGroundPGPGCPS pgag pgred ( redux
                                fn v2 => case v2 of redux
                                    Match => c (Match (Site 0)) redux
                                    (* This case can not happen *) redux
                                    (* | Context => None *) redux
                                    | None => c (None) redux
                                ) redux
                            ) redux
                        ) redux
                    ) redux
                ) redux
            ) redux
        ) redux
    ) redux
  )

(* We have two controls but of different kinds *)
(* So we add the current control as context on the possible match *)
else ( redux
    matchGroundPGPGCPS pgag redux ( redux
        fn v => case v of redux
            Match pgres => c (Context (Control (ctlag, pgres))) redux
            | Context pgres => c (Context (Control (ctlag, pgres))) redux
            | None => c None redux
        ) redux
    )
)
3 THE ALGORITHM

(* The redux is not a control on top-level, so we add *)
(* the agent control as context *)
| _ ⇒ matchGroundPGPGCPSP pgag redux {
  fn v ⇒ case v of
  Match pgres ⇒ c (Context (Control (ctag, pgres)))
  | Context pgres ⇒ c (Context (Control (ctag, pgres)))
  | None ⇒ c None
|}

Listing 8: Case Par

(* In a parallel composition in the redux we can match *)
(* the redux in either branch *)
| Par (pglag, pg2ag) ⇒ {
  (* Match entire redux in left branch *)
  matchGroundPGPGCPSP pglag redux {
    fn v1 ⇒ case v1 of
    Match pgres ⇒ c (Context (Par (pgres, pg2ag))))
    | Context pgres ⇒ c (Context (Par (pgres, pg2ag))))
    | None ⇒ {
      (* Match entire redux in right branch *)
      matchGroundPGPGCPSP pg2ag redux {
        fn v2 ⇒ case v2 of
        Match pgres ⇒ c (Context (Par (pglag, pgres))))
        | Context pgres ⇒ c (Context (Par (pglag, pgres))))
        | None ⇒ {
          (* Match one part of the redux in one branch *)
          (* and the other in the other branch *)
          case redux of
          Par (pg1re, pg2re) ⇒ {
            matchGroundPGPGCPSP pglag pg1re {
              fn v3 ⇒ case v3 of
              Match _ ⇒ {
                matchGroundPGPGCPSP pg2ag pg2re {
                  fn v4 ⇒ case v4 of
                  Match _ ⇒ c (Match (Site 0))
                  | _ ⇒ c None
                }
              }
              | _ ⇒ {
                (* Permute matching in branches *)
                matchGroundPGPGCPSP pg2ag pg1re {
                  fn v5 ⇒ case v5 of
                  Match _ ⇒ {
                    matchGroundPGPGCPSP pg2ag pg1re {
                      fn v6 ⇒ case v6 of
                      Match _ ⇒ c (Match (Site 0))
                      | _ ⇒ c None
                    }
                  }
                  | _ ⇒ c None
                }
            }
          }
        }
      }
    }
  }
}

}}
3 THE ALGORITHM

Listing 9: Case Atomic

(* An atomic control can only be matched by another atomic *)
(* control of same kind *)
| Atomic_strag \Rightarrow (case redex
of Atomic_strre \Rightarrow (
  if strag = strre then c (Match (Site 0))
  else c None
)
| _ \Rightarrow c None
)

Listing 10: Driver Method

fun main1 agent redex = matchGroundPGPGPS agent redex (fn v => v)

Test Cases  The test cases are as in Sec. 3.1 so we only display the results.

Listing 11: Test Cases

val test1 = main1 example redex = Context ( Control ("v0", Control ("v1", Site 0)));
val test2a = main1 example2a redex2 = Match (Site 0);
val test2b = main1 example2b redex2 = Context ( Control ("v0", Site 0) );
val test3 = main1 example3 redex3 = Context ( Control ("v2", Control ("v1", Site 0)));
val test4 = main1 example4 redex4 = None;
OC #0.0.0.1.13.569: (0 ms)
  val test1 = true : bool
  val test2a = true : bool
  val test2b = true : bool
  val test3 = true : bool
  val test4 = true : bool

3.3 Defunctionalised Version

Basically defunctionalisation [3] is a program transformation which turns higher-order programs into first-order ones. For a given function space (the continuation is this setting) we find all the inhabitants and enumerate those (in our project we have just enumerated them as their appear textually in the code) and represent them as a sum type. We then represent every abstraction with a data constructor that will take as arguments parts of the environment needed for evaluating the body. Every application of the higher-order function is replaced with an apply function that will interpret the data structure.

The defunctionalised continuation (value -> value) is represented by the following sum type.
Listing 12: Defunctionalised Continuation

```
datatype Cont = C0
| C1 of Cont * string * PlaceGraph * PlaceGraph
| C2 of Cont
| C3 of Cont * string
| C4 of Cont * string
| C5 of Cont * PlaceGraph * PlaceGraph * PlaceGraph
| C6 of Cont * PlaceGraph * PlaceGraph * PlaceGraph
| C7 of Cont * PlaceGraph * PlaceGraph * PlaceGraph * PlaceGraph
| C8 of Cont
| C9 of Cont * PlaceGraph * PlaceGraph
| C10 of Cont
```

The matching algorithm (match2 : PlaceGraph -> PlaceGraph -> Cont -> Value) is then transformed into the following, where we have replace every declaration (of the continuation) into a sum construction and every application into a call to the apply function (apply : Cont -> Value -> Value).

Listing 13: The Match Function

```
fun match2 agent redex c =
  case agent
    (* We the agent is a control on top-level *)
  of Control (ctlred, pgred) => (c
    case redex
      of Control (ctlred, pgred) => (c
        if ctltag = ctlred then
          (* First try to match the redex further down, even if *)
          (* this control matches the top of the redex *)
          match2 pgag redex (C1 (c, ctltag, pgag, pgred))
          (* So we have two controls but of different kinds *)
        else match2 pgag redex (C3 (c, ctltag))
        )
          (* The redex is not a control on top-level, so we add *)
          (* the agent control as context *)
        | _ => match2 pgag redex (C4 (c, ctltag))
      )
      (* New case *)
      (* An atomic control can only be matched by another atomic *)
      (* control of same kind *)
    | Atomic strag => (c
      case redex
        of Atomic strre => (c
          if strag = strre then apply c (Match (Site 0))
          else apply c None
        )
      )
      (* New case *)
      (* In a parallel composition in the redex we can match *)
      (* the redex in either branch *)
    | Par (pg1ag, pg2ag) => (c
      match2 pg1ag redex (C5 (c, pg1ag, pg2ag, redex))
    )
  )
```
3 THE ALGORITHM

(* New case *)
(* Should not be reached as we only match *)
(* on ground bigraphs *)
(* / Site n => apply c None *)

The apply function (apply : Cont -> Value -> Value) is then responsible for representing
the bodies of the continuations. The apply function highlights a possible optimisation (which we
have not performed) as the cases for C3 and C4 are equal, so we could merge the two cases into
one.

Listing 14: The Apply Function

and apply (C0) v = v
  | apply (C1 (c, ctag, pgag, pgred)) v =
    |
    case v of
    Match pgres => apply c (Context (Control (ctag, pgres)))
    | Context pgres => apply c (Context (Control (ctag, pgres)))
    | None =>
        (* Try to match when this control in the agent *)
        (* should match this control in the redux *)
        match2 pgag pgred (C2 (c))
    
    | apply (C2 (c)) v =
      |
      case v of
      Match _ => apply c (Match (Site 0))
      (* This case can not happen *)
      (* / Context => None *)
      | None => apply c (None)
    
    | apply (C3 (c, ctag)) v =
      |
      case v of
      Match pgres => apply c (Context (Control (ctag, pgres)))
      | Context pgres => apply c (Context (Control (ctag, pgres)))
      | None => apply c None
    
    | apply (C4 (c, ctag)) v =
      |
      case v of
      Match pgres => apply c (Context (Control (ctag, pgres)))
      | Context pgres => apply c (Context (Control (ctag, pgres)))
      | None => apply c None
    
    | apply (C5 (c, pglag, pg2ag, redux)) v =
      |
      case v of
      Match pgres => apply c (Context (Par (pgres, pg2ag)])
      | Context pgres => apply c (Context (Par (pgres, pg2ag)])
      | None =>
          (* Match entire redux in right branch *)
          match2 pg2ag redux (C6 (c, pglag, pg2ag, redux))
    
    | apply (C6 (c, pglag, pg2ag, redux)) v =
3 THE ALGORITHM

\[
\begin{align*}
\text{case } v \text{ of} \\
\quad \text{Match } pgres &\Rightarrow \text{apply } c \left( \text{Context } \left( \text{Par} \left( \text{pg1ag, pgres} \right) \right) \right) \\
\quad \text{Context } pgres &\Rightarrow \text{apply } c \left( \text{Context } \left( \text{Par} \left( \text{pg1ag, pgres} \right) \right) \right) \\
\quad \text{None } &\Rightarrow \\
\quad \left( \ast \text{ Match one part of the redux in one branch } \ast \right) \\
\quad \left( \ast \text{ and the other in the other branch } \ast \right) \\
\quad \text{case } \text{redux of} \\
\quad \quad \text{Par} \left( \text{pg1re, pg2re} \right) &\Rightarrow \\
\quad \quad \text{match2 } \text{pg1ag pg1re } \left( C7 \left( c, \text{pg1ag, pg2ag, pg1re, pg2re} \right) \right) \\
\quad \quad \left\{ \begin{array}{ll}
\ast &\Rightarrow \text{apply } c \text{ None}
\end{array} \right.
\end{align*}
\]

The driver method is just changed by replacing the continuation with the proper constructor.

Listing 15: Driver Method

\[
\begin{align*}
\text{fun } \text{main2 agent redux } &= \text{match2 agent redux } C0
\end{align*}
\]

Test Cases The test cases are as in Sec. 3.1 so again we only display the results.

Listing 16: Test Cases

\[
\begin{align*}
\text{val } \text{test1 } &= \text{main2 example redux } = \text{Context} \left( \text{Control} \left( \text{"v0", Control } \left( \text{"v1", Site } 0 \right) \right) \right); \\
\text{val } \text{test2a } &= \text{main2 example2a redux2 } = \text{Match} \left( \text{Site } 0 \right); \\
\text{val } \text{test2b } &= \text{main2 example2b redux2 } = \text{Context} \left( \text{Control} \left( \text{"v0", Site } 0 \right) \right); \\
\text{val } \text{test3 } &= \text{main2 example3 redux3 } = \text{Context} \left( \text{Control} \left( \text{"v2", Control } \left( \text{"v1", Site } 0 \right) \right) \right); \\
\text{val } \text{test4 } &= \text{main2 example4 redux4 } = \text{None}; \\
\text{val } \text{test1 } &= \text{true } : \text{bool} \\
\quad - \text{val } \text{test2a } &= \text{true } : \text{bool} \\
\quad - \text{GC } \#\text{0.0.0.1.21.743: } \left( 0 \text{ ms} \right) \\
\text{val } \text{test2b } &= \text{true } : \text{bool} \\
\quad - \text{val } \text{test3 } &= \text{true } : \text{bool} \\
\quad - \text{val } \text{test4 } &= \text{true } : \text{bool}
\end{align*}
\]
3 THE ALGORITHM

3.4 As a Transition System

In order to transform the defunctionalised version into a transition system, where the combination of the name of a function together with its augments represents a configuration and each function clause represents a transition, we have placed all the pattern matching on top-level in both functions match2 and apply instead of the nested case-constructs used in the previous sections. We have also lifted the checks for string equality on controls (in the cases for Control and Atomic) to top-level, even though this is not supported by SML. We will in the following section for brevity abbreviate apply with app.

Transitions from an app configuration

\[
\begin{align*}
(C0, v)_{\text{app}} & \Rightarrow v \\
(C1(c, ct\text{lag}, pgag, pg\text{red}), \text{Match pgres})_{\text{app}} & \Rightarrow (c, (\text{Context}(\text{Control}(ct\text{lag}, pg\text{res}))))_{\text{app}} \\
(C1(c, ct\text{lag}, pgag, pg\text{red}), \text{Context pgres})_{\text{app}} & \Rightarrow (c, (\text{Context}(\text{Control}(ct\text{lag}, pg\text{res}))))_{\text{app}} \\
(C1(c, ct\text{lag}, pgag, pg\text{red}), \text{None})_{\text{app}} & \Rightarrow (pgag, pg\text{red}, C2(c))_{\text{match2}} \\
(C2(c), \text{Match Site})_{\text{app}} & \Rightarrow (c, \text{Match(Site 0)})_{\text{app}} \\
(C3(c, ct\text{lag}), \text{Match pgres})_{\text{app}} & \Rightarrow (c, (\text{Context}(\text{Control}(ct\text{lag}, pg\text{res}))))_{\text{app}} \\
(C3(c, ct\text{lag}), \text{Context pgres})_{\text{app}} & \Rightarrow (c, (\text{Context}(\text{Control}(ct\text{lag}, pg\text{res}))))_{\text{app}} \\
(C3(c, ct\text{lag}), \text{None})_{\text{app}} & \Rightarrow (c, \text{None})_{\text{app}} \\
(C4(c, ct\text{lag}), \text{Match pgres})_{\text{app}} & \Rightarrow (c, (\text{Context}(\text{Control}(ct\text{lag}, pg\text{res}))))_{\text{app}} \\
(C4(c, ct\text{lag}), \text{Context pgres})_{\text{app}} & \Rightarrow (c, (\text{Context}(\text{Control}(ct\text{lag}, pg\text{res}))))_{\text{app}} \\
(C4(c, ct\text{lag}), \text{None})_{\text{app}} & \Rightarrow (c, \text{None})_{\text{app}} \\
(C5(c, pg1ag, pg2ag, redex), \text{Match pgag})_{\text{app}} & \Rightarrow (c, (\text{Context}(\text{Par}(pg\text{ag}, pg2ag))))_{\text{app}} \\
(C5(c, pg1ag, pg2ag, redex), \text{Context pgag})_{\text{app}} & \Rightarrow (c, (\text{Context}(\text{Par}(pg\text{ag}, pg2ag))))_{\text{app}} \\
(C5(c, pg1ag, pg2ag, redex), \text{None})_{\text{app}} & \Rightarrow (pg2ag, redex, (C6(c, pg1ag, pg2ag, redex)))_{\text{match2}} \\
(C6(c, pg1ag, pg2ag, redex), \text{Match pgag})_{\text{app}} & \Rightarrow (c, (\text{Context}(\text{Par}(pg\text{ag}, pg\text{res}))))_{\text{app}} \\
(C6(c, pg1ag, pg2ag, redex), \text{Context pgag})_{\text{app}} & \Rightarrow (c, (\text{Context}(\text{Par}(pg\text{ag}, pg\text{res}))))_{\text{app}} \\
(C6(c, pg1ag, pg2ag, \text{Par(pg1re, pg2re)}), \text{None})_{\text{app}} & \Rightarrow (pg1ag, #pg\text{ag}, (C7(c, pg1ag, pg2ag, pg1re, pg2re)))_{\text{match2}} \\
(C6(c, pg1ag, pg2ag, \text{None})_{\text{app}} & \Rightarrow (c, \text{None})_{\text{app}} \\
(C7(c, pg1ag, pg2ag, pg1re, pg2re), \text{Match Site})_{\text{app}} & \Rightarrow (pg2ag, pg1re, (C8(c)))_{\text{match2}} \\
(C7(c, pg1ag, pg2ag, pg1re, pg2re), \text{None})_{\text{app}} & \Rightarrow (pg1ag, pg2re, (C9(c, pg2ag, pg1re)))_{\text{match2}} \\
(C8(c), \text{Match Site})_{\text{app}} & \Rightarrow (c, \text{Match(Site 0)})_{\text{app}} \\
(C8(c), \text{None})_{\text{app}} & \Rightarrow (c, \text{None})_{\text{app}} \\
(C9(c, pg2ag, pg1re), \text{Match Site})_{\text{app}} & \Rightarrow (pg2ag, pg1re, (C10(c)))_{\text{match2}} \\
(C9(c, pg2ag, pg1re), \text{None})_{\text{app}} & \Rightarrow (c, \text{None})_{\text{app}} \\
(C10(c), \text{Match Site})_{\text{app}} & \Rightarrow (c, \text{Match(Site 0)})_{\text{app}} \\
(C10(c), \text{None})_{\text{app}} & \Rightarrow (c, \text{None})_{\text{app}}
\end{align*}
\]

Transitions from a match2 configuration

\[
\begin{align*}
((\text{Control}(ct\text{lag}, pgag)), (\text{Control}(ct\text{lag}, pgag)), c)_{\text{match2}} & \Rightarrow (pgag, redex, (C1(c, ct\text{lag}, pgag, pg\text{red})))_{\text{match2}} \\
((\text{Control}(ct\text{lag}, pgag)), (\text{Control}(ct\text{lag}, pgag)), c)_{\text{match2}} & \Rightarrow (pgag, redex, (C3(c, ct\text{lag})))_{\text{match2}} \\
((\text{Control}(ct\text{lag}, pgag)), (\text{Control}(ct\text{lag}, pgag)), c)_{\text{match2}} & \Rightarrow (pgag, redex, (C4(c, ct\text{lag})))_{\text{match2}} \\
((\text{Atomic(strag)}), (\text{Atomic(strag)}), c)_{\text{match2}} & \Rightarrow (c, \text{Match(Site 0)})_{\text{app}} \\
((\text{Atomic(strag)}), (\text{Atomic(strag)}), c)_{\text{match2}} & \Rightarrow (c, \text{None})_{\text{app}} \\
((\text{Atomic(strag)}), (\text{Atomic(strag)}), c)_{\text{match2}} & \Rightarrow (c, \text{None})_{\text{app}} \\
((\text{Par}(pg1ag, pg2ag)), \text{redex}, c)_{\text{match2}} & \Rightarrow (pg1ag, redex, (C5(c, pg1ag, #pg2ag, redex)))_{\text{match2}}
\end{align*}
\]
The driver method.

\[ pg_1 pg_2 \Rightarrow (pg_1, pg_2, C_0)_{\text{match2}} \]

References

