

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 thought 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.

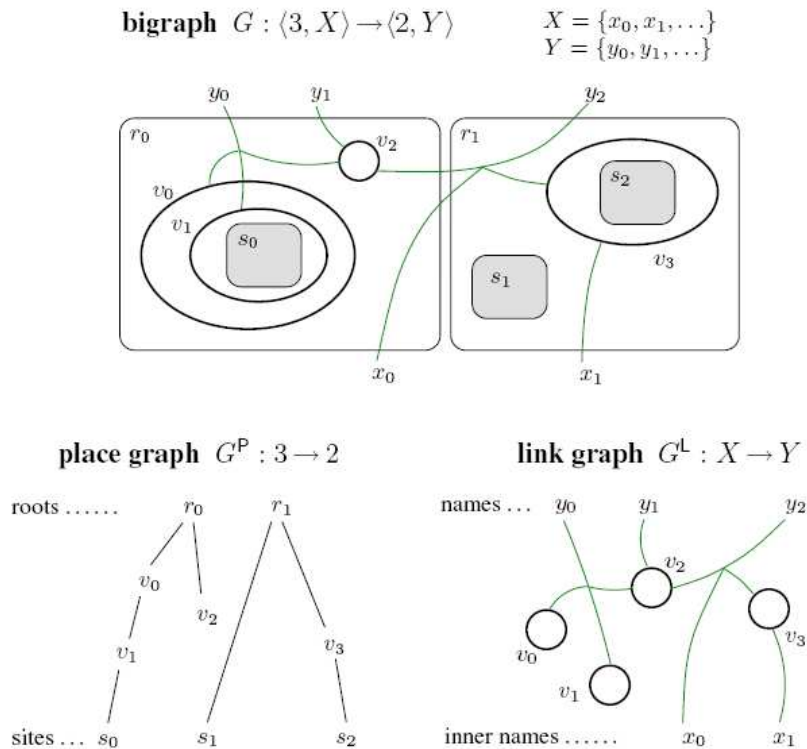


Figure 1: Bigraphs

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 *redex* and a *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 *reaction rule* as a pair (r, r') of *ground* place graphs¹. Given a ground place graph a (called the *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 *agent* and the redex are *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 *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 .

¹a place graph is called ground if it contains no holes.

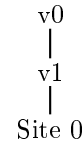
The agent a



The redex r

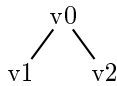


The result C



Let us consider a more complex example, where we have a node of control $v0$ with two children of controls $v1$ and $v2$, respectively. We want to match this with the redex r which is just two nodes of controls $v1$ and $v2$ in parallel (recall that the children are unordered). The result of this is C which is just the node of control $v0$ containing the hole.

The agent a



The redex r



The result C



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

```
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

```
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 redex 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 redex 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 redex is a control, but of different kind.
 - If the redex 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 toplevel control in both the agent and the redex.
- If the agent is a parallel composition there are several cases.
 - Either we can match the entire redex in either branch and then put the remaining branch in parallel when returning the result.
 - If the redex is also a parallel composition we can also match one part of the redex in one part of the agent and match the remaining part of the redex 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.

```
matchGroundPGPG : PlaceGraph -> PlaceGraph -> Value
```

Listing 3: Case Control

```

fun matchGroundPGPG agent redex =
  case agent
  (* We the agent is a control on top-level *)
  of Control (ctlag, pgag) => (
    case redex
    of Control (ctlred, pgreg) => (
      if ctlag = ctlred then (
        (* First try to match the redex further down, even if *)
        (* this control matches the top of the redex *)
        case matchGroundPGPG pgag redex of
          Match pgres => Context (Control (ctlag, pgres))
        | Context pgres => Context (Control (ctlag, pgres))
        | None => (
          (* Try to match when this control in the agent *)
          (* should match this control in the redex *)

```

```

    case matchGroundPGPG pgag pged of
      Match _ => Match (Site 0)
      (* This case can not happend *)
      (* / 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 pges => Context (Control (ctlag, pges))
      | Context pges => Context (Control (ctlag, pges))
      | 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 pges => Context (Control (ctlag, pges))
    | Context pges => Context (Control (ctlag, pges))
    | None => None
  )
)

```

Listing 4: Case Par

```

(* In a parallel composition in the redex we can match *)
(* the redex in either branch *)
| Par (pg1ag, pg2ag) => (
  (* Match entire redex in left branch *)
  case matchGroundPGPG pg1ag redex
  of Match pges => Context (Par (pgres, pg2ag))
  | Context pges => Context (Par (pgres, pg2ag))
  | None => (
    (* Match entire redex in right branch *)
    case matchGroundPGPG pg2ag redex
    of Match pges => Context (Par (pg1ag, pges))
    | Context pges => Context (Par (pg1ag, pges))
    | None => (
      (* Match one part of the redex in one branch *)
      (* and the other in the other branch *)
      case redex
      of Par (pg1re, pg2re) => (
        case matchGroundPGPG pg1ag pg1re of
          Match _ => (
            case matchGroundPGPG pg2ag pg2re of
              Match _ => Match (Site 0)
              | _ => None
            )
          )
        | _ => (
          (* Permute matching in branches *)
          case matchGroundPGPG pg1ag pg2re of
            Match _ => (

```



```

- 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 redex c =
  case agent
  (* We the agent is a control on top-level *)
  of Control (ctlag, pgag) => (
    case redex
    of Control (ctlred, pgrid) => (
      if ctlag = ctlred then (
        (* First try to match the redex further down, even if *)
        (* this control matches the top of the redex *)
        matchGroundPGPGCPS pgag redex (
          fn v => case v of
            Match pgres => c (Context (Control (ctlag, pgres)))
          | Context pgres => c (Context (Control (ctlag, pgres)))
          | None => (
              (* Try to match when this control in the agent *)
              (* should match this control in the redex *)
              matchGroundPGPGCPS pgag pgrid (
                fn v2 => case v2 of
                  Match _ => c (Match (Site 0))
                  (* This case can not happend *)
                  (* | Context _ => None *)
                  | None => c (None)
              )
            )
          )
        )
      )
    )
  )
  (* We have two controls but of different kinds *)
  (* So we add the current control as context on the possible match *)
  else (
    matchGroundPGPGCPS pgag redex (
      fn v => case v of
        Match pgres => c (Context (Control (ctlag, pgres)))
      | Context pgres => c (Context (Control (ctlag, pgres)))
      | None => c None
    )
  )

```

```

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

```

Listing 8: Case Par

```

(* In a parallel composition in the redex we can match *)
(* the redex in either branch *)
| Par (pg1ag, pg2ag) => (
  (* Match entire redex in left branch *)
  matchGroundPGPGCPS pg1ag redex (
    fn v1 => case v1 of
      Match pgres => c (Context (Par (pgres, pg2ag)))
    | Context pgres => c (Context (Par (pgres, pg2ag)))
    | None => (
      (* Match entire redex in right branch *)
      matchGroundPGPGCPS pg2ag redex (
        fn v2 => case v2 of
          Match pgres => c (Context (Par (pg1ag, pgres)))
        | Context pgres => c (Context (Par (pg1ag, pgres)))
        | None => (
          (* Match one part of the redex in one branch *)
          (* and the other in the other branch *)
          case redex of
            Par (pg1re, pg2re) => (
              matchGroundPGPGCPS pg1ag pg1re (
                fn v3 => case v3 of
                  Match _ => (
                    matchGroundPGPGCPS pg2ag pg2re (
                      fn v4 => case v4 of
                        Match _ => c (Match (Site 0))
                      | _ => c None
                    )
                  )
                )
              )
            | _ => (
              (* Permute matching in branches *)
              matchGroundPGPGCPS pg1ag pg2re (
                fn v5 => case v5 of
                  Match _ => (
                    matchGroundPGPGCPS pg2ag pg1re (
                      fn v6 => case v6 of
                        Match _ => c (Match (Site 0))
                      | _ => c None
                    )
                  )
                )
              )
            | _ => c None
          )
        )
      )
    )
  )
)

```


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 (ctlag, pgag) => (
    case redex
    of Control (ctlred, pgrid) => (
      if ctlag = 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, ctlag, pgag, pgrid))
      (* We have two controls but of different kinds *)
      (* So we add the current control as context on the possible match *)
      else match2 pgag redex (C3 (c, ctlag))
    )
    (* The redex is not a control on top-level, so we add *)
    (* the agent control as context *)
    | _ => match2 pgag redex (C4 (c, ctlag))
  )
  (* New case *)
  (* An atomic control can only be matched by another atomic *)
  (* control of same kind *)
  | Atomic strag => (case redex
    of Atomic stre => (
      if strag = stre then apply c (Match (Site 0))
      else apply c None
    )
    | _ => apply c None)
  (* New case *)
  (* In a parallel composition in the redex we can match *)
  (* the redex in either branch *)
  | Par (pg1ag, pg2ag) =>
    (* Match entire redex in left branch *)
    match2 pg1ag redex (C5 (c, pg1ag, pg2ag, redex))

```

```

(* 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, ctlag, pgag, pgrid)) v =
  (
    case v of
      Match pgres => apply c (Context (Control (ctlag, pgres)))
    | Context pgres => apply c (Context (Control (ctlag, pgres)))
    | None =>
      (* Try to match when this control in the agent *)
      (* should match this control in the redex *)
      match2 pgag pgrid (C2 (c))
  )
| apply (C2 (c)) v =
  (
    case v of
      Match _ => apply c (Match (Site 0))
      (* This case can not happend *)
      (* / Context _ => None *)
    | None => apply c (None)
  )
| apply (C3 (c, ctlag)) v =
  (
    case v of
      Match pgres => apply c (Context (Control (ctlag, pgres)))
    | Context pgres => apply c (Context (Control (ctlag, pgres)))
    | None => apply c None
  )
| apply (C4 (c, ctlag)) v =
  (
    case v of
      Match pgres => apply c (Context (Control (ctlag, pgres)))
    | Context pgres => apply c (Context (Control (ctlag, pgres)))
    | None => apply c None
  )
| apply (C5 (c, pg1ag, pg2ag, redex)) v =
  (
    case v of
      Match pgres => apply c (Context (Par (pgres, pg2ag)))
    | Context pgres => apply c (Context (Par (pgres, pg2ag)))
    | None =>
      (* Match entire redex in right branch *)
      match2 pg2ag redex (C6 (c, pg1ag, pg2ag, redex))
  )
| apply (C6 (c, pg1ag, pg2ag, redex)) v =

```

```

(
  case v of
    Match pgres => apply c (Context (Par (pglag, pgres)))
  | Context pgres => apply c (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) =>
        match2 pglag pg1re (C7 (c, pglag, pg2ag, pg1re, pg2re))
      | _ => apply c None
    )
  )
| apply (C7 (c, pglag, pg2ag, pg1re, pg2re)) v =
  ( case v of
    Match _ => match2 pg2ag pg2re (C8 (c))
    | _ =>
      (* Permute matching in branches *)
      match2 pglag pg2re (C9 (c, pg2ag, pg1re))
  )
| apply (C8 (c)) v =
  ( case v of
    Match _ => apply c (Match (Site 0))
    | _ => apply c None )
| apply (C9 (c, pg2ag, pg1re)) v =
  ( case v of
    Match _ => match2 pg2ag pg1re (C10 c)
    | _ => apply c None )
| apply (C10 (c)) v =
  ( case v of
    Match _ => apply c (Match (Site 0))
    | _ => apply c None )

```

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

Listing 15: Driver Method

```
fun main2 agent redex = match2 agent redex C0
```

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

Listing 16: Test Cases

```

val test1 = main2 example redex = Context( Control( "v0", Control ( "v1", Site 0)));
val test2a = main2 example2a redex2 = Match (Site 0);
val test2b = main2 example2b redex2 = Context( Control( "v0", Site 0) );
val test3 = main2 example3 redex3 = Context( Control( "v2", Control ( "v1", Site 0)));
val test4 = main2 example4 redex4 = None;
val test1 = true : bool
- val test2a = true : bool
- GC #0.0.0.1.21.743: (0 ms)
val test2b = true : bool
- val test3 = true : bool
- val test4 = true : bool

```

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 arguments 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 toplevel, 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.

$\langle C0, v \rangle_{\text{app}}$	$\Rightarrow v$
$\langle C1(c, \text{ctlag}, \text{pgag}, \text{pgred}), \text{Match } \text{pgres} \rangle_{\text{app}}$	$\Rightarrow \langle c, (\text{Context}(\text{Control}(\text{ctlag}, \text{pgres}))) \rangle_{\text{app}}$
$\langle C1(c, \text{ctlag}, \text{pgag}, \text{pgred}), \text{Context } \text{pgres} \rangle_{\text{app}}$	$\Rightarrow \langle c, (\text{Context}(\text{Control}(\text{ctlag}, \text{pgres}))) \rangle_{\text{app}}$
$\langle C1(c, \text{ctlag}, \text{pgag}, \text{pgred}), \text{None} \rangle_{\text{app}}$	$\Rightarrow \langle \text{pgag}, \text{pgred}, C2(c) \rangle_{\text{match2}}$
$\langle C2(c), \text{Match } _ \rangle_{\text{app}}$	$\Rightarrow \langle c, \text{Match}(\text{Site } 0) \rangle_{\text{app}}$
$\langle C2(c), \text{None} \rangle_{\text{app}}$	$\Rightarrow \langle c, \text{None} \rangle_{\text{app}}$
$\langle C3(c, \text{ctlag}), \text{Match } \text{pgres} \rangle_{\text{app}}$	$\Rightarrow \langle c, (\text{Context}(\text{Control}(\text{ctlag}, \text{pgres}))) \rangle_{\text{app}}$
$\langle C3(c, \text{ctlag}), \text{Context } \text{pgres} \rangle_{\text{app}}$	$\Rightarrow \langle c, (\text{Context}(\text{Control}(\text{ctlag}, \text{pgres}))) \rangle_{\text{app}}$
$\langle C3(c, \text{ctlag}), \text{None} \rangle_{\text{app}}$	$\Rightarrow \langle c, \text{None} \rangle_{\text{app}}$
$\langle C4(c, \text{ctlag}), \text{Match } \text{pgres} \rangle_{\text{app}}$	$\Rightarrow \langle c, (\text{Context}(\text{Control}(\text{ctlag}, \text{pgres}))) \rangle_{\text{app}}$
$\langle C4(c, \text{ctlag}), \text{Context } \text{pgres} \rangle_{\text{app}}$	$\Rightarrow \langle c, (\text{Context}(\text{Control}(\text{ctlag}, \text{pgres}))) \rangle_{\text{app}}$
$\langle C4(c, \text{ctlag}), \text{None} \rangle_{\text{app}}$	$\Rightarrow \langle c, \text{None} \rangle_{\text{app}}$
$\langle C5(c, \text{pg1ag}, \text{pg2ag}, \text{redex}), \text{Match } \text{pgreg} \rangle_{\text{app}}$	$\Rightarrow \langle c, (\text{Context}(\text{Par}(\text{pgres}, \text{pg2ag}))) \rangle_{\text{app}}$
$\langle C5(c, \text{pg1ag}, \text{pg2ag}, \text{redex}), \text{Context } \text{pgreg} \rangle_{\text{app}}$	$\Rightarrow \langle c, (\text{Context}(\text{Par}(\text{pgres}, \text{pg2ag}))) \rangle_{\text{app}}$
$\langle C5(c, \text{pg1ag}, \text{pg2ag}, \text{redex}), \text{None} \rangle_{\text{app}}$	$\Rightarrow \langle \text{pg2ag}, \text{redex}, (C6(c, \text{pg1ag}, \text{pg2ag}, \text{redex})) \rangle_{\text{match2}}$
$\langle C6(c, \text{pg1ag}, \text{pg2ag}, \text{redex}), \text{Match } \text{pgreg} \rangle_{\text{app}}$	$\Rightarrow \langle c, (\text{Context}(\text{Par}(\text{pg1ag}, \text{pgres}))) \rangle_{\text{app}}$
$\langle C6(c, \text{pg1ag}, \text{pg2ag}, \text{redex}), \text{Context } \text{pgreg} \rangle_{\text{app}}$	$\Rightarrow \langle c, (\text{Context}(\text{Par}(\text{pg1ag}, \text{pgres}))) \rangle_{\text{app}}$
$\langle C6(c, \text{pg1ag}, \text{pg2ag}, \text{Par}(\text{pg1re}, \text{pg2re})), \text{None} \rangle_{\text{app}}$	$\Rightarrow \langle \text{pg1ag}, \text{pg1re}, (C7(c, \text{pg1ag}, \text{pg2ag}, \text{pg1re}, \text{pg2re})) \rangle_{\text{match2}}$
$\langle C6(c, \text{pg1ag}, \text{pg2ag}, _), \text{None} \rangle_{\text{app}}$	$\Rightarrow \langle c, \text{None} \rangle_{\text{app}}$
$\langle C7(c, \text{pg1ag}, \text{pg2ag}, \text{pg1re}, \text{pg2re}), \text{Match } _ \rangle_{\text{app}}$	$\Rightarrow \langle \text{pg2ag}, \text{pg2re}, (C8(c)) \rangle_{\text{match2}}$
$\langle C7(c, \text{pg1ag}, \text{pg2ag}, \text{pg1re}, \text{pg2re}), _ \rangle_{\text{app}}$	$\Rightarrow \langle \text{pg1ag}, \text{pg2re}, (C9(c, \text{pg2ag}, \text{pg1re})) \rangle_{\text{match2}}$
$\langle C8(c), \text{Match } _ \rangle_{\text{app}}$	$\Rightarrow \langle c, (\text{Match}(\text{Site } 0)) \rangle_{\text{app}}$
$\langle C8(c), _ \rangle_{\text{app}}$	$\Rightarrow \langle c, \text{None} \rangle_{\text{app}}$
$\langle C9(c, \text{pg2ag}, \text{pg1re}), \text{Match } _ \rangle_{\text{app}}$	$\Rightarrow \langle \text{pg2ag}, \text{pg1re}, (C10c) \rangle_{\text{match2}}$
$\langle C9(c, \text{pg2ag}, \text{pg1re}), _ \rangle_{\text{app}}$	$\Rightarrow \langle c, \text{None} \rangle_{\text{app}}$
$\langle C10(c), \text{Match } _ \rangle_{\text{app}}$	$\Rightarrow \langle c, (\text{Match}(\text{Site } 0)) \rangle_{\text{app}}$
$\langle C10(c), _ \rangle_{\text{app}}$	$\Rightarrow \langle c, \text{None} \rangle_{\text{app}}$

Transitions from a `match2` configuration

$\langle (\text{Control}(\text{ctlag}, \text{pgag}), (\text{Control}(\text{ctlag}, \text{pgred})), c) \rangle_{\text{match2}}$	$\Rightarrow \langle \text{pgag}, \text{redex}, (C1(c, \text{ctlag}, \text{pgag}, \text{pgred})) \rangle_{\text{match2}}$
$\langle (\text{Control}(\text{ctlag}, \text{pgag}), (\text{Control}(\text{ctlag}, \text{pgred})), c) \rangle_{\text{match2}}$	$\Rightarrow \langle \text{pgag}, \text{redex}, (C3(c, \text{ctlag})) \rangle_{\text{match2}}$
$\langle (\text{Control}(\text{ctlag}, \text{pgag}), _ , c) \rangle_{\text{match2}}$	$\Rightarrow \langle \text{pgag}, \text{redex}, (C4(c, \text{ctlag})) \rangle_{\text{match2}}$
$\langle (\text{Atomic}(\text{strag}), (\text{Atomic}(\text{strag})), c) \rangle_{\text{match2}}$	$\Rightarrow \langle c, (\text{Match}(\text{Site } 0)) \rangle_{\text{app}}$
$\langle (\text{Atomic}(\text{strag}), (\text{Atomic}(\text{stre})), c) \rangle_{\text{match2}}$	$\Rightarrow \langle c, \text{None} \rangle_{\text{app}}$
$\langle (\text{Atomic}(\text{strag}), _ , c) \rangle_{\text{match2}}$	$\Rightarrow \langle c, \text{None} \rangle_{\text{app}}$
$\langle (\text{Par}(\text{pg1ag}, \text{pg2ag}), \text{redex}, c) \rangle_{\text{match2}}$	$\Rightarrow \langle \text{pg1ag}, \text{redex}, (C5(c, \text{pg1ag}, \text{pg2ag}, \text{redex})) \rangle_{\text{match2}}$

The driver method.

$$pg1pg2 \Rightarrow \langle pg1, pg2, C0 \rangle_{\text{match2}}$$

References

- [1] Ole Høgh Jensen. *Mobile Processes in Bigraphs*. PhD thesis, Department of Computer Science, Aalborg University, 2006. Forthcoming.
- [2] Ole Høgh Jensen and Robin Milner. Bigraphs and mobile processes (revised). Technical Report UCAM-CL-TR-580, University of Cambridge, Computer Laboratory, 2004.
- [3] John C. Reynolds. Definitional interpreters for higher-order programming languages. *Higher-Order and Symbolic Computation*, 11(4):363–397, 1998.