UNIVERSITY OF SUSSEX

COMPUTER SCIENCE

A chemical abstract machine for graph reduction

Alan Jeffrey

Report 3/92 August 1992

Computer Science
School of Cognitive and Computing Sciences
University of Sussex
Brighton BN1 9QH

ISSN 1350–3170

A chemical abstract machine for graph reduction

ALAN JEFFREY

ABSTRACT. Graph reduction is an implementation technique for the lazy λ-calculus. It has been used to implement many non-strict functional languages, such as lazy ML, Gofer and Miranda. Parallel graph reduction allows for concurrent evaluation. In this paper, we present parallel graph reduction as a Chemical Abstract Machine, and show that the resulting testing semantics is adequate wrt testing equivalence for the lazy λ-calculus. We also present a π-calculus implementation of the graph reduction machine, and show that the resulting testing semantics is also adequate.

1 Introduction

The lazy reduction strategy for the λ-calculus investigated by ABRAMSKY (1989) has only two reduction rules:

$$\frac{}{(\lambda x.E)F \rightarrow E[F/x]} \quad \frac{E \rightarrow E'}{EF \rightarrow E'F}$$

This can be compared with the full evaluation strategy of BARENDREGT (1984):

$$\frac{}{(\lambda x.E)F \twoheadrightarrow E[F/x]} \quad \frac{E \twoheadrightarrow E'}{EF \twoheadrightarrow E'F} \quad \frac{F \twoheadrightarrow F'}{EF \twoheadrightarrow EF'} \quad \frac{E \twoheadrightarrow E'}{\lambda x.E \twoheadrightarrow \lambda x.E'}$$

If the full evaluation strategy can terminate, then the lazy evaluation strategy will. For example, if we define:

$$\begin{aligned} \mathsf{K} &= \lambda xy.x \\ \mathsf{I} &= \lambda x.x \\ \mathsf{Y} &= \lambda x.((\lambda y.x(yy))(\lambda y.x(yy))) \end{aligned}$$

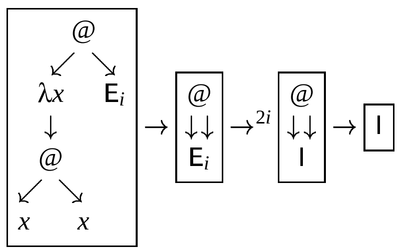
then $\mathsf{YI} \rightarrow^{\infty}$ but $\mathsf{KI}(\mathsf{YI}) \not\rightarrow^{\infty}$, whereas $\mathsf{KI}(\mathsf{YI}) \twoheadrightarrow^{\infty}$. However, the lazy evaluation strategy is very inefficient, since it may duplicate arguments when applying a function. For example, if we define:

$$\begin{aligned} \mathsf{E}_0 &= \mathsf{I} \\ \mathsf{E}_{i+1} &= (\lambda x.xx)\mathsf{E}_i \end{aligned}$$

Author's address: COGS, University of Sussex, Brighton BN1 9QH, UK.
EMail: alanje@cogs.sussex.ac.uk
This work was supported by SERC project GR/H 16537.

Then $\mathsf{E}_i \twoheadrightarrow^{2i} \mathsf{I}$ but $\mathsf{E}_i \rightarrow^{2^{i+1}-2} \mathsf{I}$, that is the lazy strategy can be exponentially worse than the full strategy. Thus, the early functional languages, such as LISP (MCCARTHY *et al.*, 1962) used a strict reduction scheme rather than the lazy reduction scheme.

Graph reduction was introduced by WADSWORTH (1971) as a means of efficiently implementing the lazy reduction strategy. Rather than reducing syntax trees, we reduce syntax *graphs* which allows a more efficient representation of sharing. For example, we can represent the reduction of E_{i+1} as:



Graph reduction has been used to implement non-strict functional languages such as JOHNSSON's lazy ML (1984), JONES's Gofer (1992) and TURNER's Miranda (1985). It is discussed in PEYTON JONES's textbook (1987).

However, there has been little work in the formal semantics of graph reduction. BARENDREGT *et al.* (1987) have shown that graph reduction is sound and complete with respect to term reduction. LESTER (1989) has shown that the *G*-machine of AUGUSTSSON (1984) and JOHNSSON (1984) is adequate wrt a denotational model of the lazy λ-calculus. In this paper, we provide an alternative presentation of graph reduction, as a *Chemical Abstract Machine* (CHAM), in the style of BERRY and BOUDOL (1990).

The CHAM was introduced as a way of presenting the operational semantics of parallel languages in a clean fashion. It has been used to give a semantics for MILNER's CCS (1989) and MILNER, PARROW and WALKER's π-calculus (1989).

Here, we shall give a semantics for parallel graph reduction with blocking, as described by PEYTON JONES (1987). We will show that this is an adequate semantics for the lazy λ-calculus, and that it can be implemented in a variant of the π-calculus.

2 The lazy lambda-calculus

The λ-calculus, introduced by CHURCH (1941), has the following syntax:

$$E ::= x \mid EE \mid \lambda x.E$$

where x ranges over an infinite set of variables. This can be given a number of operational semantics, but we shall only look at two of these. We shall call these the *lazy* semantics:

$$\frac{}{(\lambda x.E)F \to E[F/x]} \quad \frac{E \to E'}{EF \to E'F}$$

and the *full* semantics:

$$\frac{}{(\lambda x.E)F \twoheadrightarrow E[F/x]} \quad \frac{E \twoheadrightarrow E'}{EF \twoheadrightarrow E'F} \quad \frac{F \twoheadrightarrow F'}{EF \twoheadrightarrow EF'} \quad \frac{E \twoheadrightarrow E'}{\lambda x.E \twoheadrightarrow \lambda x.E'}$$

Here, $E[F/x]$ is E, with every free occurrence of x replaced by F, up to the usual renaming of bound variables.

PROPOSITION 1.

1. *If* $E \to E'$ *and* $E \to E''$ *then* $E' = E''$.
2. *If* $E \to^{\infty}$ *and* $E \twoheadrightarrow E'$ *then* $E' \to^{\infty}$.
3. *If* $E\ (\to\twoheadrightarrow^{*})^{\infty}$ *then* $E \to^{\infty}$.

PROOF. Part 1 is a simple induction. Part 2 follows from Theorem 13.2.2 of BARANDREGT (1984), since $E \to E'$ is the leftmost full reduction of E. Part 3 follows from Theorem 13.2.6 of BARANDREGT (1984). □

We can define a variant of MORRIS's testing pre-order (BARENDREGT, 1984, Exercise 16.5.5):

$$E \sqsubseteq F \quad \text{iff} \quad \forall C\,.\,C[F] \to^{\infty} \Rightarrow C[E] \to^{\infty}$$

We can also define a variant of the λ-calculus with recursive declarations and strictness annotations:

$$\begin{aligned} M &::= x \mid xy \mid \lambda x.M \mid \mathsf{rec}\, x := D \,\mathsf{in}\, M \\ D &::= ?M \mid !M \end{aligned}$$

Here:

- $\mathsf{rec}\, x := ?M \,\mathsf{in}\, N$ declares x recursively to be M in the context N. For example, a fixed point of f is $\mathsf{rec}\, x := ?f \,\mathsf{in}\, \mathsf{rec}\, y := ?xy \,\mathsf{in}\, y$.
- $\mathsf{rec}\, x := !M \,\mathsf{in}\, N$ is the same, except that x is strict in N, and so evaluation of M can be sparked off as a parallel computation.

We shall let bound variables be α-converted. The free variables of M are $\mathrm{fv}\,M$:

$$\begin{aligned} \mathrm{fv}\,x &= \{x\} \\ \mathrm{fv}(xy) &= \{x,y\} \\ \mathrm{fv}(\lambda x.M) &= \mathrm{fv}\,M \setminus \{x\} \\ \mathrm{fv}(\mathsf{rec}\, x := D \,\mathsf{in}\, M) &= (\mathrm{fv}\,D \cup \mathrm{fv}\,M) \setminus \{x\} \\ \mathrm{fv}(!M) &= \mathrm{fv}\,M \\ \mathrm{fv}(?M) &= \mathrm{fv}\,M \end{aligned}$$

There is a translation $\overline{\cdot}$ from the λ-calculus to the λ-calculus with rec:

$$\begin{aligned}\overline{\mathrm{x}} &= x\\ \overline{\mathrm{EF}} &= \mathsf{rec}\, x := !\,\overline{\mathrm{E}}\,\mathsf{in}\,\mathsf{rec}\, y := ?\,\overline{\mathrm{F}}\,\mathsf{in}\, xy\\ \overline{\lambda \mathrm{x}.\mathrm{E}} &= \lambda x.\,\overline{\mathrm{E}}\end{aligned}$$

Note that in the translation of EF, we know that E will be used, and so it can be evaluated strictly. On the other hand, we do not know if F will be used or not, so it cannot be annotated.

3 The chemical abstract machine

The Chemical Abstract Machine (CHAM) of BERRY and BOUDOL (1990) is a way of presenting the operational semantics of parallel systems. We shall use it to give a semantics for parallel graph reduction of the λ-calculus with rec.

A CHAM gives reductions between *solutions*, which are multisets (or *bags*) of *molecules*. The definition of molecules is specific to each CHAM, but a solution can always be regarded as a molecule. In such a molecule, $\{\!|m_1,\ldots,m_n|\!\}$, the multiset brackets $\{\!|\cdots|\!\}$ are called a *membrane*. There are three types of reduction:

- *Heating rules*, of the form $S \rightharpoonup S'$.
- *Cooling rules*, of the form $S \rightharpoondown S'$.
- *Reaction rules*, of the form $S \mapsto S'$.

Heating and cooling rules are always given in pairs $S \rightleftharpoons S'$, whereas reaction rules are irreversible. We shall write $\rightleftharpoons^*$ for the transitive, reflexive, symmetric closure of $\rightleftharpoons$, write $\rightarrow$ for $\rightleftharpoons^* \mapsto \rightleftharpoons^*$, and let $\Rightarrow$ range over $\rightharpoonup$, $\rightharpoondown$ and $\mapsto$. All CHAMs have the following structural rules, where $m[\cdot]$ is a molecule containing precisely one hole:

$$\frac{S \Rightarrow S'}{S \uplus S'' \Rightarrow S' \uplus S''} \quad \frac{S \Rightarrow S'}{\{\!|m[S]|\!\} \Rightarrow \{\!|m[S']|\!\}}$$

In addition, the CHAMs we shall consider in this paper allow the outermost membrane of any solution to be ignored. This allows us to write $m_1,\ldots,m_n \Rightarrow m'_1,\ldots,m'_{n'}$ for $\{\!|m_1,\ldots,m_n|\!\} \Rightarrow \{\!|m'_1,\ldots,m'_{n'}|\!\}$:

$$\{\!|S|\!\} \rightleftharpoons S$$

The molecules and reduction rules are specific to each CHAM. In the case of the graph reduction CHAM, molecules are defined:

$$m ::= x := D \mid S \mid \nu x.S$$

The free variables of m are $\mathrm{fv}\, m$:

$$\mathrm{fv}(x := D) = \{x\} \cup \mathrm{fv}\, D$$

$$\begin{aligned}
\mathrm{fv}\,\{\!|m_1,\ldots,m_n|\!\} &= \mathrm{fv}\,m_1 \cup \cdots \cup \mathrm{fv}\,m_n \\
\mathrm{fv}(\nu x.m) &= \mathrm{fv}\,m \setminus \{x\}
\end{aligned}$$

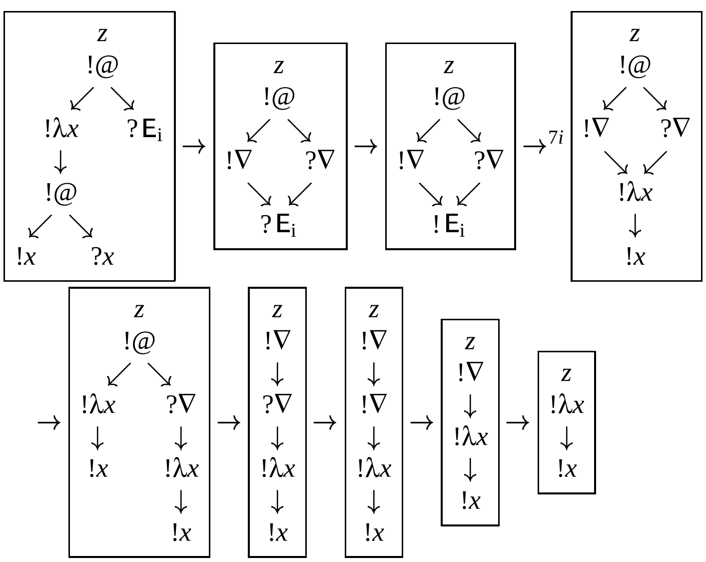
The defined variables of m are $\mathrm{dv}\,m$:

$$\begin{aligned}
\mathrm{dv}(x := D) &= \{x\} \\
\mathrm{dv}\,\{\!|m_1,\ldots,m_n|\!\} &= \mathrm{dv}\,m_1 \cup \cdots \cup \mathrm{dv}\,m_n \\
\mathrm{dv}(\nu x.m) &= \mathrm{dv}\,m \setminus \{x\}
\end{aligned}$$

We shall only consider solutions which do not define any variables twice, so in any solution $\{\!|m_1,\ldots,m_n|\!\}$, the defined variables of each m_i are distinct. For example, we do not allow solutions such as $\nu x.\{\!|x := !\lambda w.M, x := !\lambda w.N, y := !xw|\!\}$ which could reduce nondeterministically to $\{\!|y := !M|\!\}$ or to $\{\!|y := !N|\!\}$. If $\tilde{x} = x_1,\ldots,x_n$ then we can write $\nu x.m$ for $\nu x.\{\!|m|\!\}$ and $\nu\tilde{x}.m$ for $\nu x_1 \ldots \nu x_n.m$. Define:

- a molecule is a *positive ion* with *valency* x iff it is $x := ?M$ or $x := !\lambda y.M$.
- a molecule is a *negative ion* with *valency* y iff it is $x := !y$ or $x := !yz$.
- a molecule is *ionic* iff it is a positive or negative ion.
- a solution is *plasmic* iff it is $\{\!|\nu\tilde{y}.\{\!|m_1,\ldots,m_n|\!\}|\!\}$ or $\{\!|m_1,\ldots,m_n|\!\}$ where each m_i is ionic. A plasma is positive (negative) iff it contains only positive (negative) ions

Plasmas can be regarded as graphs, for example the graph reduction:



is represented by the CHAM reduction:

$$
\begin{aligned}
&\nu xy.\{\!|z := !xy, y := ?\,\mathsf{E}_{\mathrm{i}}, x := !\,\lambda \mathrm{w.ww}|\!\}\\
&\quad \to \nu uvy.\{\!|z := !uv, y := ?\,\mathsf{E}_{\mathrm{i}}, v := ?y, u := !y|\!\}\\
&\quad \to \nu uvy.\{\!|z := !uv, y := !\,\mathsf{E}_{\mathrm{i}}, v := ?y, u := !y|\!\}\\
&\quad \to^{7i} \nu uvy.\{\!|z := !uv, y := !\,\mathsf{I}, v := ?y, u := !y|\!\}\\
&\quad \to \nu uvy.\{\!|z := !uv, y := !\,\mathsf{I}, v := ?y, u := !\,\mathsf{I}|\!\}\\
&\quad \to \nu vy.\{\!|z := !v, y := !\,\mathsf{I}, v := ?y|\!\}\\
&\quad \to \nu vy.\{\!|z := !v, y := !\,\mathsf{I}, v := !y|\!\}\\
&\quad \to \nu v.\{\!|z := !v, v := !\,\mathsf{I}|\!\}\\
&\quad \to \{\!|z := !\,\mathsf{I}|\!\}
\end{aligned}
$$

In these diagrams:

- Tagged nodes $x := !M$ are labelled with a !.
- Untagged nodes $x := ?M$ are labelled with a ?.
- Application nodes $x := yz$ are labelled with a @.
- Indirection nodes $x := y$ are labelled with a y, if y is free, and with ∇ otherwise.
- Function nodes $x := \lambda y.M$ are labelled with a λy, and have the graph for $\{\!|z := !M|\!\}$ drawn beneath them, for some fresh variable z.

The *supercombinators* of HUGHES (1984) can be regarded as a form of solution. PEYTON JONES (1987) presents supercombinators as a collection of definitions, plus a variable to be evaluated. These are drawn:

$$
\frac{\text{Definitions}}{\text{Variable}}
$$

These can be regarded as solutions. For example, the supercombinator program:

$$
\frac{\begin{array}{l}\$X_0\,\tilde{x}_0 = E_0\\ \$X_1\,\tilde{x}_1 = E_1\\ \quad\vdots\\ \$X_n\,\tilde{x}_n = E_n\end{array}}{\$X_0}
$$

is equivalent to the solution:

$$
\nu X_0 \ldots X_n \{\!|X_0 := !\,\lambda\tilde{\mathrm{x}}_0.\mathrm{E}_0, X_1 := ?\,\lambda\tilde{\mathrm{x}}_1.\mathrm{E}_1, \ldots, X_n := ?\,\lambda\tilde{\mathrm{x}}_{\mathrm{n}}.\mathrm{E}_{\mathrm{n}}|\!\}
$$

The most important heating rule allows recursive declarations to become part of a solution, whilst hiding the bound variable. This is only valid when it would not cause the free variable x to become bound by y, which we can achieve by α-converting y first.

$$x := !\,\mathsf{rec}\, y := D\, \mathsf{in}\, M \rightleftharpoons \nu y.\{\!|x := !M, y := D|\!\} \qquad (x \neq y)$$

The scope of a hidden variable can migrate, as long as this does not result in variable capture:

$$m, \nu x.m' \rightleftharpoons \nu x.\{\!|m, m'|\!\} \qquad (x \notin \mathrm{fv}\, m)$$

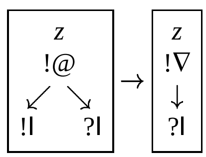
Hidden variables may be α-converted, exchanged and evaporated:

$$\begin{aligned} \nu x.m &\rightleftharpoons \nu y.(m[y/x]) \qquad (y \notin \mathrm{fv}\, m) \\ \nu xy.m &\rightleftharpoons \nu yx.m \\ \nu x.\{\!|\,|\!\} &\rightleftharpoons \{\!|\,|\!\} \end{aligned}$$

Finally, we can perform garbage collection on positive plasmas, since a hidden positive plasma can never make any reductions:

$$\nu\tilde{x}.\{\!|\tilde{x} := \tilde{D}|\!\} \rightleftharpoons \{\!|\,|\!\} \qquad (\{\!|\tilde{x} := \tilde{D}|\!\} \text{ is a positive plama})$$

We shall sometimes write $\rightleftharpoons_\gamma$ for this thermal action, and $\rightleftharpoons_{\neq\gamma}$ for any other thermal action. For example, the graph reduction:



can be derived:

$$\begin{aligned} &\nu yx.\{\!|x := !\mathsf{I}, y := ?\mathsf{I}, z := !xy|\!\} \\ &\quad \mapsto \nu yx.\{\!|x := !\mathsf{I}, y := ?\mathsf{I}, z := !y|\!\} \\ &\quad \rightharpoondown \nu yx.\{\!|x := !\mathsf{I}, \{\!|y := ?\mathsf{I}, z := !y|\!\}\,|\!\} \\ &\quad \rightharpoondown \nu y.\{\!|\nu x.\{\!|x := !\mathsf{I}|\!\}\,, \{\!|y := ?\mathsf{I}, z := !y|\!\}\,|\!\} \\ &\quad \rightharpoonup_\gamma \nu y.\{\!|\,\{\!|y := ?\mathsf{I}, z := !y|\!\}\,|\!\} \\ &\quad \rightharpoonup \nu y.\{\!|y := ?\mathsf{I}, z := !y|\!\} \end{aligned}$$

A reaction can occur whenever one positive and one negative ion with the same valency exist in a solution. Since there are two kinds of positive ion and two kinds of negative ion, there are four reaction rules. The first two allow untagged molecules to become tagged:

$$\begin{aligned} x := !y, y := ?M &\mapsto x := !y, y := !M \\ x := !yz, y := ?M &\mapsto x := !yz, y := !M \end{aligned}$$

These can be drawn:

$$\boxed{\begin{array}{c} !\nabla \\ \downarrow \\ ?M \end{array}} \mapsto \boxed{\begin{array}{c} !\nabla \\ \downarrow \\ !M \end{array}}$$

$$\boxed{\begin{array}{cc} & !@ \\ \swarrow & \\ ?M & \end{array}} \mapsto \boxed{\begin{array}{cc} & !@ \\ \swarrow & \\ !M & \end{array}}$$

This models the first phase of graph reduction—we search along the spine of a graph, tagging nodes for evaluation. Note that strict reactions do not occur:

$$x := !yz, z := ?M \not\mapsto x := !yz, z := !M$$

If a tagged indirection node points to a function, we can just copy the function. The SKIM (STOYE *et al.*, 1984) and *G*-machine (JOHNNSON, 1984) use this as a method of eliminating indirection nodes. It was shown by LESTER (1989) to be adequate:

$$x := !y, y := !\lambda w.M \mapsto x := !\lambda w.M, y := !\lambda w.M$$

This can be drawn:

$$\boxed{\begin{array}{c} !\nabla \\ \downarrow \\ !\lambda w.M \end{array}} \mapsto \boxed{\begin{array}{c} !\lambda w.M \\ \\ !\lambda w.M \end{array}}$$

If an application node points to a function, it can be β-reduced:

$$x := !yz, y := !\lambda w.M \mapsto x := !M[z/w], y := !\lambda w.M$$

This can be drawn:

$$\boxed{\begin{array}{ccccc} & & !@ & & \\ & \swarrow & & \searrow & \\ & !\lambda w & & & N \\ & \downarrow & & & \\ & M & & & \\ \swarrow & & \searrow & & \\ w & \cdots & w & & \end{array}} \mapsto \boxed{\begin{array}{cccccc} & & M & & & \\ & \swarrow & & \searrow & & \\ & !\lambda w & & \nabla & \cdots & \nabla \\ & \downarrow & & & \searrow\swarrow & \\ & M & & & N & \\ \swarrow & & \searrow & & & \\ w & \cdots & w & & & \end{array}}$$

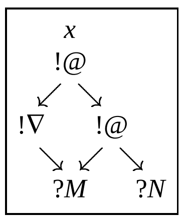
We shall sometimes write $\mapsto_\beta$ for this reaction, and $\mapsto_{\neq\beta}$ for any other reaction. This CHAM is summarized in Table 1.

This CHAM implements the algorithm for parallel graph reduction described by PEYTON JONES (1987). A process is assigned to evaluating a node, which is tagged. It then searches along the spine, tagging each node as it passes. If

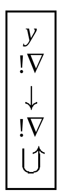
$$
\begin{aligned}
\{\!|S|\!\} &\rightleftharpoons S \\
x := !\,\mathsf{rec}\, y := D\,\mathsf{in}\, M &\rightleftharpoons \nu y.\{\!|x := !M, y := D|\!\} \qquad (x \neq y) \\
m, \nu x.m' &\rightleftharpoons \nu x.\{\!|m, m'|\!\} \qquad (x \notin \mathrm{fv}\, m) \\
\nu x.m &\rightleftharpoons \nu y.(m[y/x]) \qquad (y \notin \mathrm{fv}\, m) \\
\nu xy.m &\rightleftharpoons \nu yx.m \\
\nu x.\{\!|\,|\!\} &\rightleftharpoons \{\!|\,|\!\} \\
\nu \tilde{x}.\{\!|\tilde{x} := \tilde{D}|\!\} &\rightleftharpoons \{\!|\,|\!\} \qquad (\{\!|\tilde{x} := \tilde{D}|\!\} \text{ is a positive plasma}) \\
x := !y, y := ?M &\mapsto x := !y, y := !M \\
x := !yz, y := ?M &\mapsto x := !yz, y := !M \\
x := !y, y := !\lambda w.M &\mapsto x := !\lambda w.M, y := !\lambda w.M \\
x := !yz, y := !\lambda w.M &\mapsto x := !M[z/w], y := !\lambda w.M
\end{aligned}
$$

TABLE 1. Summary of the graph reduction CHAM

it reaches a function node which can be β-reduced, it does so. If it reaches a function node which cannot be β-reduced, this is returned as the result. If it reaches a previously tagged application or indirection node, it is blocked until the tagged node is evaluated. For example, in the graph:

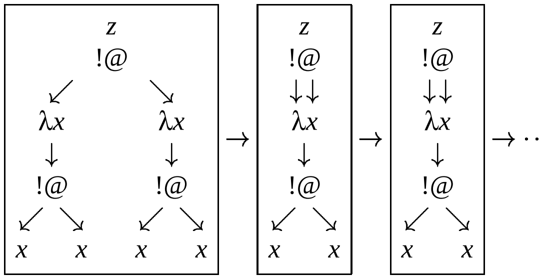


only one process will evaluate M. This is mirrored in the CHAM by the fact that M will only be reduced once. However, this algorithm produces some surprising results with cyclic graphs. The solution $\{\!|y := !\,\mathsf{rec}\, x := !x\,\mathsf{in}\, x|\!\}$ heats to become the plasma $\{\!|\nu x.\{\!|y := !x, x := !x|\!\}\,|\!\}$ and the graph:



This has no reductions, because it is negative. This is mirrored in the parallel graph reduction algorithm, since the process evaluating y will discover that the indirection node at x has already been tagged. Thus, it is possible for evaluations to deadlock, when a sequential algorithm would diverge.

Our translation of the λ-calculus will not produce cyclic graphs, although it can still produce divergent terms. For example, the translation of $(\lambda x.xx)(\lambda x.xx)$ has the reductions:



Since E has no tight loops, we will be able to show that the CHAM semantics for the λ-calculus is adequate. To do this, we define the testing preorder on molecules:

$$m \sqsubseteq m' \quad \text{iff} \quad \forall C \,.\, C[m'] \to^{\infty} \Rightarrow C[m] \to^{\infty}$$

and show in the next section that the CHAM semantics is *adequate*, that is if $(x := ?\mathrm{E}) \sqsubseteq (x := ?\mathrm{F})$ then $E \sqsubseteq F$.

PROPOSITION 2.

1. *For any S there is a plasmic S' such that $S \rightharpoonup_{\neq\gamma} S'$.*
2. *If S is plasmic and $S \to S'$ then $S \mapsto\rightleftharpoons^* S'$.*
3. $S \to S'$ *iff* $S \rightharpoonup^*_{\neq\gamma}\mapsto\rightleftharpoons^* S'$
4. $S\ (\rightharpoonup^*_{\neq\gamma}\mapsto)^{\infty}$ *iff* $S \to^{\infty}$
5. $S \to^{\infty}_{\neq\beta}$ *iff* $S \to^{\infty}_{\neq\beta\neq\gamma}$
6. $S \not\to^{\infty}_{\neq\beta}$
7. *If $S \to S'$ and $S \to S''$ then $S' \rightleftharpoons^* S''$ or $S' \to S'''$ and $S'' \to S'''$.*
8. $S \not\to^{\infty}$ *iff* $S \to^* \not\to$

PROOF.

1. Prove by induction on S.
2. Prove by case analysis that if S is plasmic and $S \rightleftharpoons\mapsto S'$ then $S \mapsto\rightleftharpoons S'$. Then show by induction on $\rightleftharpoons^*$ that if $S \rightleftharpoons^*\mapsto S'$ that $S \mapsto\rightleftharpoons^* S'$.
3. If $S \to S'$ then by parts 1 and 2, $S \rightharpoonup^*_{\neq\gamma} S'' \mapsto\rightleftharpoons^* S'$ for some plasmic S''. Thus $S \to_{\neq\gamma}\rightleftharpoons^* S'$.
4. If $S \to^{\infty}$ then define S_i by induction:
   - $S_0 = S$.
   - Assuming $S_i \to^{\infty}$, $S_i \to S' \to^{\infty}$, so by part 3, $S_i \rightharpoonup^*_{\neq\gamma}\mapsto_{\neq\gamma} S_{i+1} \rightleftharpoons^* S'$, so $S_{i+1} \to^{\infty}$.

Thus, $S = S_0 \rightharpoonup^*_{\neq\gamma}\mapsto S_1 \rightharpoonup^*_{\neq\gamma}\mapsto \cdots$ and so $S \to^\infty$ implies $S\ (\rightharpoonup^*_{\neq\gamma}\mapsto)^\infty$. The reverse implication is trivial.

5. Is proved similarly.

6. Define $\#m$ by:

$$
\begin{aligned}
\#\{\!|m_1,\ldots,m_n|\!\} &= \#m_1 + \cdots + \#m_n \\
\#(\nu x.m) &= \#m \\
\#(x := D) &= \#D \\
\#(!M) &= \#M \\
\#(?M) &= \#M + 1 \\
\#(\lambda x.M) &= 0 \\
\#y &= 1 \\
\#yz &= 1 \\
\#(\mathsf{rec}\, x := D \,\mathsf{in}\, M) &= \#D + \#M
\end{aligned}
$$

Then:

$$
\begin{aligned}
S \rightleftharpoons_{\neq\gamma} S' &\Rightarrow \#S = \#S' \\
S \mapsto_{\neq\beta} S' &\Rightarrow \#S > \#S'
\end{aligned}
$$

Thus, $S \not\to^\infty_{\neq\beta\neq\gamma}$, so by part 5, $S \not\to^\infty_{\neq\beta}$.

7. Show by case analysis that if S is plasmic and $S \mapsto S'$ and $S \mapsto S''$ then $S' = S''$ or $S' \mapsto S'''$ and $S'' \mapsto S'''$.

   Then for any S, if $S \to S'$ and $S \to S''$ then by parts 1 and 2 there is a plasmic S_1 such that $S \rightleftharpoons^* S_1 \mapsto S'_1 \rightleftharpoons^* S'$ and $S_1 \mapsto S''_1 \rightleftharpoons^* S''$. Then either $S'_1 = S''_1$, so $S' \rightleftharpoons^* S''$, or $S'_1 \mapsto S'''$ and $S''_1 \mapsto S'''$, so $S' \to S'''$ and $S'' \to S'''$.

8. If $S \to^\infty$ and $S \to^* \not\to$ then we must have:

$$
\begin{aligned}
S = S^0_0 \to S^0_1 \to \cdots \to S^0_n \to S^0_{n+1} \\
S = S^0_0 \to S^1_1 \to \cdots \to S^n_n \not\to
\end{aligned}
$$

   Then for $0 < i < j \leq n+1$, define S^i_j such that $S^{i-1}_{j-1} \to S^i_j$ and $S^i_{j-1} \to S^i_j$:

   - If $S^{i-1}_{j-1} \rightleftharpoons^* S^i_{j-1}$ then let $S^i_j = S^{i-1}_j$.
   - Otherwise, $S^{i-1}_{j-2} \to S^{i-1}_{j-1}$ and $S^{i-1}_{j-2} \to S^i_{j-1}$ so by part 7, we can find S^i_j such that $S^{i-1}_{j-1} \to S^i_j$ and $S^i_{j-1} \to S^i_j$.

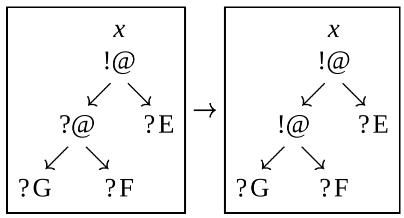
   Then $S^n_n \to S^n_{n+1}$ which contradicts our hypothesis. Thus, if $S \to^* \not\to$ then $S \not\to^\infty$. □

Note that Proposition 2.7 shows that this CHAM is *Church-Rosser*.

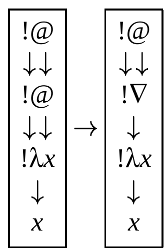
4 The CHAM semantics is adequate

MILNER, PARROW and WALKER (1989) have shown a very close relationship between the lazy reduction strategy and their translation of the λ-calculus into the π-calculus, that $E \to F$ iff $[\![E[\![\to [\![F[\![$.

This is not be true of the CHAM semantics for graph reduction, where there is no one-to-one correspondence between CHAM reductions and λ-calculus reductions. For example, the CHAM reduction:

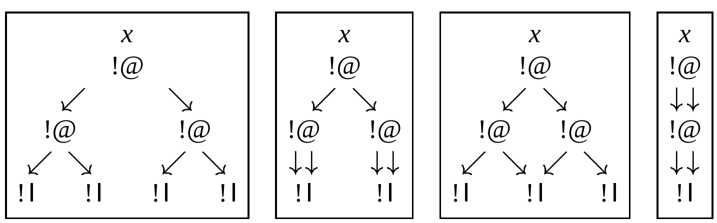


corresponds to the null reduction $GFE \to^0 GFE$, and the CHAM reduction:



corresponds to $\mathsf{II}(\mathsf{II}) \rightarrow\!\!\!\twoheadrightarrow \mathsf{II}$. In general, each CHAM reduction $S \to_{\neq\beta} S'$ corresponds to a λ-calculus reduction $E \to^0 E$, and each CHAM reduction $S \to_\beta S'$ corresponds to a λ-calculus reduction $E \rightarrow\!\!\!\twoheadrightarrow^* E'$.

In order to formalize this, we need to know when a CHAM solution can implement a λ-term. For example, some possible implementations of $\mathsf{II}(\mathsf{II})$ are:



We will write $S \vdash M \triangleleft E$ iff M implements E in the solution S. For example, $S \vdash x \triangleleft \mathsf{II}(\mathsf{II})$ for any of the above solutions. $S \vdash M \triangleleft E$ is defined in Table 2.

PROPOSITION 3.

$$(x)\ \frac{}{S \vdash x \triangleleft x}\ [x \notin \mathrm{dv}\, S]$$

$$(xy)\ \frac{S \vdash x \triangleleft E \quad S \vdash y \triangleleft F}{S \vdash xy \triangleleft EF}$$

$$(\lambda x)\ \frac{S \vdash M \triangleleft E}{S \vdash \lambda x.M \triangleleft \lambda x.E}\ [x \notin \mathrm{fv}\, S]$$

$$(\nu x)\ \frac{S \vdash M \triangleleft E}{\nu x.S \vdash M \triangleleft E}\ [x \notin \mathrm{fv}\, M, E]$$

$$(\rightharpoonup)\ \frac{S \vdash M \triangleleft E}{S' \vdash M \triangleleft E}\ [S' \rightharpoonup S]$$

$$(?)\ \frac{S, x := !M \vdash N \triangleleft E}{S, x := ?M \vdash N \triangleleft E}$$

$$(:= E)\ \frac{S, x := !M \vdash x \triangleleft E}{S \vdash M \triangleleft E}\ [x \notin \mathrm{fv}\, E, M, S]$$

$$(:= I)\ \frac{S \vdash M \triangleleft E}{S, x := !M \vdash x \triangleleft E}\ [x \notin \mathrm{fv}\, E, M, S]$$

TABLE 2. The definition of $S \vdash M \triangleleft E$.

1. *If* $S \vdash M \triangleleft E$ *and* $S \vdash M \triangleleft F$ *then* $E = F$*.*
2. *If* $S, x := !yz \vdash x \triangleleft E$ *then* $S \vdash y \triangleleft F$*,* $S \vdash z \triangleleft G$ *and* $E = FG$*.*
3. *If* $S, x := !\lambda y.M \vdash x \triangleleft E$ *then* $S \vdash M \triangleleft F$ *and* $E = \lambda y.F$*.*
4. *If* $\nu x.S \vdash M \triangleleft E$ *and* $x \notin \mathrm{fv}\, M, E$ *then* $S \vdash M \triangleleft E$
5. *If* $S, x := !M \vdash x \triangleleft E$ *and* $S, x := !M \vdash N \triangleleft F$ *then* $S \vdash N \triangleleft G$ *and* $F = G[E/x]$*.*
6. *If* $S, x := !y, y := !M \vdash N \triangleleft E$ *then* $S, x := !M, y := !M \vdash N \triangleleft E$*.*
7. *If* $x \notin \mathrm{fv}\, E$ *then* $x := !\mathrm{E} \vdash x \triangleleft E$*.*

PROOF.

1. First, prove by induction on the proof of $S \vdash x \triangleleft E$ that if $x \notin \mathrm{dv}\, S$ and $S \vdash x \triangleleft E$ then $E = x$. Then prove by induction on the proof of $S \vdash M \triangleleft E$ that if $S \vdash M \triangleleft E$ and $S \vdash M \triangleleft F$ then $E = F$. The only tricky case is if $S \vdash M \triangleleft E$ follows from (x), in which case $M = E = x$, and since $S \vdash x \triangleleft F$ the above gives that $F = x$.

2. Prove by induction on the proof that if $\nu\tilde{w}.(S, x := !yz) \vdash x \triangleleft E$ or $S \vdash yz \triangleleft E$ then $S \vdash y \triangleleft F$ and $S \vdash z \triangleleft G$ and $E = FG$.

3. Prove by induction on the proof that if $\nu\tilde{w}.(S, x := !\lambda y.M) \vdash x \triangleleft E$ or $S \vdash \lambda y.M \triangleleft E$ then for any z, $S[z/y] \vdash M \triangleleft F$ and $E = \lambda y.F$. The only tricky case is if $S \vdash \lambda y.M \triangleleft E$ follows from (νy), in which case $S = \nu y.S'$ and $S' \vdash \lambda y.M \triangleleft E$. By induction, $S'[u/y] \vdash M \triangleleft F$ and $E = \lambda y.F$ for some fresh u, so by (νu), $\nu u.(S'[u/y]) \vdash M \triangleleft F$ so by $(\rightharpoonup)$, $\nu y.S' \vdash M \triangleleft F$, so $S \vdash M \triangleleft F$, so since $y \notin \mathrm{fv}\, S$,

$S[z/y] \vdash M \triangleleft F$.

4. Prove by induction on the proof of $\nu\tilde{x}x.S \vdash M \triangleleft E$ that if $\nu\tilde{x}x.S \vdash M \triangleleft E$ and $x \notin \mathrm{fv}\, M, E$ that $\nu\tilde{x}.S \vdash M \triangleleft E$.

5. Prove by induction on the proof of $\nu\tilde{w}.(S, x := !M) \vdash N \triangleleft F$ that if $\nu\tilde{w}.(S, x := !M) \vdash x \triangleleft E$ and $\nu\tilde{w}.(S, x := !M) \vdash N \triangleleft F$ then $S \vdash N \triangleleft G$ and $F = G[E/x]$.

6. First, prove by induction on the proof of $S \vdash M \triangleleft E$ that if $y \notin \mathrm{fv}\, E, M, S$ and $S \vdash M \triangleleft E$ then $S, y := !N \vdash M \triangleleft E$. Then prove by induction on the proof of $S \vdash M \triangleleft E$ that if $\nu\tilde{w}(S, x := !y, y := !M) \vdash N \triangleleft E$ then $S, x := !M, y := !M \vdash N \triangleleft E$ and if $\nu\tilde{w}(S, y := !M) \vdash y \triangleleft E$ then $S, y := !M \vdash M \triangleleft E$.

7. Proved by induction on E. ◻

However, the definition of $S \vdash M \triangleleft E$ ignores whether declarations are tagged or untagged. In order to reason about the reductions of a solution, we shall need to take tagging into account.

The sequence $x_0 := D_0, \ldots, x_n := D_n$ is an x-spine in S iff:

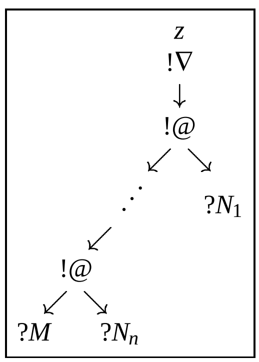
$$S \rightleftharpoons^* \nu\tilde{y}.\{\!|\tilde{x} := \tilde{D}, S'|\!\}$$

where $x = x_0$ and each $x_i := D_i$ is a negative ion with valency x_{i+1}. The variable x is lazy in a plasma S iff the negative ions of S form an x-spine. The variable x is always lazy in S iff for any $S \rightarrow^* S'$, x is lazy in S'.

For example, the z-spine of the plasma:

$$\begin{aligned} \nu\tilde{y}\tilde{z}.\{\!|z := !z_0, z_0 := !z_1 y_1, \ldots, z_{n-1} := !z_n y_n,\\ z_n := ?M, y_0 := ?N_1, \ldots, y_n := ?N_n|\!\} \end{aligned}$$

which can be drawn:



is $z := !z_0, z_0 := !z_1 y_1, \ldots, z_{n-1} := !z_n y_n$. Since these are the only negative ions, this solution is lazy.

PROPOSITION 4. *x is always lazy in* $\{\!|x := !\,\mathrm{E}|\!\}$.

PROOF. Define x is invariantly lazy (or i-lazy) in S by:

- x is i-lazy in $\{\!|\,|\!\}$
- If x is i-lazy in S and $x \neq y$ then x is i-lazy in $\nu y.S$
- If y is i-lazy in S then x is i-lazy in $\{\!|x := !y, S|\!\}$.
- If y is i-lazy in S then x is i-lazy in $\{\!|x := !yz, S|\!\}$.
- If x is i-lazy in S and y is i-lazy in $\{\!|y := !M|\!\}$ then x is i-lazy in $\{\!|y := ?M, S|\!\}$.
- If x is i-lazy in S and for all z, y is i-lazy in $\{\!|y := !M[z/w]|\!\}$ then x is i-lazy in $\{\!|y := !\lambda w.M, S|\!\}$.
- If x is i-lazy in S and $S \rightleftharpoons S'$ then x is i-lazy in S'.

Then we can prove:

- If x is i-lazy in S then x is lazy in S.
- If x is i-lazy in S and $S \rightarrow S'$ then x is i-lazy in S'.
- x is i-lazy in $\{\!|x := !\,\mathrm{E}|\!\}$.

Thus, if x is i-lazy in S then x is always lazy in S, and so x is always lazy in $\{\!|x := !\,\mathrm{E}|\!\}$. □

Then we can show the important properties, that link CHAM reductions to λ-calculus reductions.

PROPOSITION 5. *If $S \vdash x \triangleleft E$ and x is always lazy in S then:*

1. *If $S \rightarrow_{\neq\beta} S'$ then $S' \vdash x \triangleleft E$.*
2. *If $S \rightarrow_{\beta} S'$ then $E \rightarrow\twoheadrightarrow^* E'$ and $S' \vdash x \triangleleft E'$.*
3. *If $E \rightarrow E'$ then $S \rightarrow^*_{\neq\beta}\rightarrow_{\beta} S'$, $E' \twoheadrightarrow^* E''$ and $S' \vdash x \triangleleft E''$.*

PROOF.

1. If:
$$S \rightleftharpoons^* \nu\tilde{y}\{\!|u := !v, v := ?M, \ldots|\!\}$$
$$S' \rightleftharpoons^* \nu\tilde{y}\{\!|u := !v, v := !M, \ldots|\!\}$$
then $S' \vdash x \triangleleft E$. If:
$$S \rightleftharpoons^* \nu\tilde{y}\{\!|u := !vw, v := ?M, \ldots|\!\}$$
$$S' \rightleftharpoons^* \nu\tilde{y}\{\!|u := !vw, v := !M, \ldots|\!\}$$
then $S' \vdash x \triangleleft E$. If:
$$S \rightleftharpoons^* \nu\tilde{y}\{\!|u := !v, v := !\lambda y.M, \ldots|\!\}$$
$$S' \rightleftharpoons^* \nu\tilde{y}\{\!|u := !\lambda y.M, v := !\lambda y.M, \ldots|\!\}$$
then $S' \vdash x \triangleleft E$. These are the only three rules for $S \rightarrow_{\neq\beta} S'$, so $S' \vdash x \triangleleft E$
2. Since x is lazy in S, and $S \rightarrow_{\beta} S'$, we know that:
$$S \rightleftharpoons^* \nu\tilde{z}.\{\!|x_{n+1} := !\lambda w.M, S''|\!\}$$

$$
\begin{aligned}
S' &\rightleftharpoons^* \nu\tilde{z}.\{\!|x_{n+1} := !M[y_n/w], S''|\!\} \\
S'' &\rightleftharpoons^* \{\!|x_0 := !M_0, \ldots, x_n := !M_n, \ldots|\!\} \\
M_i &= x_{i+1} \text{ or } x_{i+1}y_{i+1} \\
x_0 &= x
\end{aligned}
$$

We shall prove the case when each $M_i = x_{i+1}y_{i+1}$, since indirection nodes make little difference. By Propositions 3.4, 3.2 and 3.3 we have:

$$
\begin{aligned}
E &= (\lambda w.F)F_n \cdots F_0 \\
x_{n+1} := !\lambda w.M, S'' &\vdash x_i \lhd (\lambda w.F)F_n \cdots F_i \\
x_{n+1} := !\lambda w.M, S'' &\vdash M \lhd F \\
x_{n+1} := !\lambda w.M, S'' &\vdash y_i \lhd F_i \\
S'' &\vdash x_{n+1} \lhd \lambda w.F \\
S'' &\vdash y_n \lhd F_n
\end{aligned}
$$

and by Proposition 3.5:

$$
\begin{aligned}
S'' &\vdash y_i \lhd G_i \\
S'' &\vdash x_i \lhd (\lambda w.F)F_n G_{n-1} \cdots G_i \\
F_i &= G_i[(\lambda w.F)F_n/x_n]
\end{aligned}
$$

So:

$$
\begin{aligned}
S' &\vdash M \lhd F \\
S' &\vdash M[y_n/w] \lhd F[F_n/w] \\
S' &\vdash x_n \lhd F[F_n/w] \\
S' &\vdash y_n \lhd F_n \\
S' &\vdash y_i \lhd H_i \\
S' &\vdash x_i \lhd F[F_n/w]H_{n-1} \cdots H_i \\
S' &\vdash x \lhd E' \\
H_i &= G_i[F[F_n/w]/x_n] \\
E' &= F[F_n/w]H_{n-1} \cdots H_0
\end{aligned}
$$

Since $(\lambda w.F)F_n \to F[F_n/w]$:

$$F_i = G_i[(\lambda w.F)F_n/x_n] \twoheadrightarrow^* G_i[F[F_n/w]/x_n] = H_i$$

so:

$$
\begin{aligned}
E = (\lambda w.F)F_n \cdots F_0 &\to F[F_n/w]F_{n-1} \cdots F_0 \\
&\twoheadrightarrow^* F[F_n/w]H_{n-1} \cdots H_0 = E'
\end{aligned}
$$

Thus, $S' \vdash x \lhd E'$ and $E \to\twoheadrightarrow^* E'$.

3. Define S_i by induction:
   - $S_0 = S$.
   - Let $S_i = \nu\tilde{z}.\{\!|x_0 := D_0, \ldots, x_m := D_m|\!\}$ and let $x_0, \ldots, x_n$ be the x-spine of S_i. If $D_{n+1} = ?M$ then define:

   $$S_{i+1} = \nu\tilde{z}.\{\!|x_0 := D_0, \ldots, x_n := D_n,\\ x_{n+1} := !M, x_{n+2} := D_{n+2}, \ldots, x_m := D_m|\!\}$$

   so $S_i \to_{\neq\beta} S_{i+1}$. If $D_n = !x_{n+1}$ and $D_{n+1} = !\lambda w.M$ then define:

   $$S_{i+1} = \nu\tilde{z}.\{\!|x_0 := D_0, \ldots, x_{n-1} := D_{n-1},\\ x_n := D_{n+1}, x_{n+1} := D_{n+1}, \ldots, x_m := D_m|\!\}$$

   so $S_i \to_{\neq\beta} S_{i+1}$. If $D_n = !x_{n+1}y$ and $D_{n+1} = !\lambda w.M$ then define:

   $$S' = \nu\tilde{z}.\{\!|x_0 := D_0, \ldots, x_{n-1} := D_{n-1},\\ x_n := !M[y/w], x_{n+1} := D_{n+1}, \ldots, x_m := D_m|\!\}$$

   so $S_i \to_\beta S'$.

   Thus, either $S \to^{\infty}_{\neq\beta}$ (which is impossible, by Proposition 2.6) or $S \to^{*}_{\neq\beta}\to_\beta S'$. By parts 1 and 2, $E \to E''' \twoheadrightarrow^* E''$ and $S' \vdash x \triangleleft E''$. Since $E \to E'$ and $E \to E'''$, by Proposition 1.1 $E' = E'''$, and so $E \to E' \twoheadrightarrow^* E''$. $\square$

From the above propositions, we can show that a λ-term diverges iff its CHAM implementation diverges.

PROPOSITION 6. *If* $x \notin \mathrm{fv}E$ *then* $E \to^\infty$ *iff* $\{\!|x := !\mathrm{E}|\!\} \to^\infty$

PROOF.

$\Rightarrow$ Define S_i and E_i by induction:

- $S_0 = \{\!|x := \mathrm{E}|\!\}$ and $E_0 = E$.
- By induction, $S_i \vdash x \triangleleft E_i$ and $E_i \to^\infty$. Then by Proposition 5.3, $S_i \to^+ S_{i+1}$, $E_i \to\twoheadrightarrow^* E_{i+1}$ and $S_{i+1} \vdash x \triangleleft E_{i+1}$. By Proposition 1.2, since $E_i \to^\infty$, we know $E_{i+1} \to^\infty$.

Thus $\{\!|x := !\mathrm{E}|\!\} = S_0 \to^+ S_1 \to^+ S_2 \to^+ \cdots$.

$\Leftarrow$ Since $\{\!|x := !\mathrm{E}|\!\} \to^\infty$, by Proposition 2.6:

$$\{\!|x := !\mathrm{E}|\!\} = S_0 \to^*_{\neq\beta}\to_\beta S_1 \to^*_{\neq\beta}\to_\beta \cdots$$

so by Propositions 5.1 and 5.2:

$$E = E_0 \to\twoheadrightarrow^* E_1 \to\twoheadrightarrow^* \cdots$$

so $E\ (\to\twoheadrightarrow^*)^\infty$, so by Proposition 1.3, $E \to^\infty$. $\square$

Thus, we can show that the CHAM semantics is adequate.

THEOREM 7 (ADEQUACY). *If* $(x := ?\mathrm{E}) \sqsubseteq (x := ?\mathrm{F})$ *then* $E \sqsubseteq F$.

PROOF. For any context C:

$$
\begin{aligned}
C[F] \to^{\infty} &\Rightarrow (\lambda w.C[w])F \to^{\infty} \\
&\Rightarrow \{\!|z := !\,(\lambda \mathrm{w.C[w]})\mathrm{F}|\!\} \to^{\infty} \\
&\Rightarrow \nu xy.\{\!|z := !yx, y := !\,\lambda \mathrm{w.C[w]}, x := ?\,\mathrm{F}|\!\} \to^{\infty} \\
&\Rightarrow \nu xy.\{\!|z := !yx, y := !\,\lambda \mathrm{w.C[w]}, x := ?\,\mathrm{E}|\!\} \to^{\infty} \\
&\Rightarrow \{\!|z := !\,(\lambda \mathrm{w.C[w]})\mathrm{E}|\!\} \to^{\infty} \\
&\Rightarrow (\lambda w.C[w])E \to^{\infty} \\
&\Rightarrow C[E] \to^{\infty}
\end{aligned}
$$

Thus, $E \sqsubseteq F$. □

However, it is not fully abstract.

THEOREM 8. $E \sqsubseteq F$ *does not imply* $(x := ?\mathrm{E}) \sqsubseteq (x := ?\mathrm{F})$.

PROOF. The CHAM semantics is not fully abstract because we can define ABRAMSKY's (1989) C combinator in the λ-calculus with rec:

$$\mathsf{C} = \lambda y.\,\mathsf{rec}\, x := !y \,\mathsf{in}\, \mathsf{I}$$

C cannot be defined in the λ-calculus, and provides extra testing power to the λ-calculus with rec. Define:

$$
\begin{aligned}
E &= \lambda x.x(\lambda y.x\top\bot y)\top \\
F &= \lambda x.x(x\top\bot)\top \\
\top &= \lambda x.(\mathsf{YK}) \\
\bot &= \mathsf{YI} \\
C[\cdot] &= \nu xy.\{\!|\cdot, y := !\mathsf{C}, z := !xy|\!\}
\end{aligned}
$$

ONG (1988, Theorem 4.5.1.1) has shown that E and F are *applicative bisimilar* (ABRAMSKY, 1989) and so $E \sqsubseteq F$. However:

$$
\begin{aligned}
&C[x := ?\mathrm{E}] \to^{*} \{\!|z := !\top|\!\} \not\to \\
&C[x := ?\mathrm{F}] \to^{*} \nu xy.\{\!|x := !\bot, y := !x, z := !\top|\!\} \to^{\infty}
\end{aligned}
$$

Thus, by Proposition 2.8, $C[x := ?\mathrm{E}] \not\to^{\infty}$, so $(x := ?\mathrm{E}) \not\sqsubseteq (x := ?F)$. □

It is an open problem as to whether the CHAM semantics is fully abstract wrt the λ-calculus with C, and as to whether the canonical semantics for the lazy λ-calculus $D \simeq (D \to D)_{\bot}$ is adequate wrt the CHAM semantics.

$$
\begin{array}{rcll}
\{\!|S|\!\} & \rightleftharpoons & S & \\
P \mid Q & \rightleftharpoons & P, Q & \\
\nu x.P & \rightleftharpoons & \nu x.\{\!|P|\!\} & \\
m, \nu x.m' & \rightleftharpoons & \nu x.\{\!|m, m'|\!\} & (x \notin \mathrm{fv}\, m) \\
\nu x.m & \rightleftharpoons & \nu y.(m[y/x]) & (y \notin \mathrm{fv}\, m) \\
\nu xy.m & \rightleftharpoons & \nu yx.m & \\
\nu xx.m & \rightleftharpoons & \nu x.m & \\
[x = x]P & \rightleftharpoons & P & \\
[x \neq y]P & \rightleftharpoons & P & (x \neq y) \\
A(\tilde{x}) & \rightleftharpoons & P[\tilde{x}/\tilde{y}] & (A(\tilde{y}) \stackrel{\text{def}}{=} P) \\
\overline{x}[yz], x(vw).P & \mapsto & P[y/v, z/w] &
\end{array}
$$

TABLE 3. CHAM for the π-calculus

5 The asynchronous pi-calculus

The π-calculus, introduced by MILNER, PARROW and WALKER (1989) is a process algebra in which scope is considered important. MILNER has shown that it can be used to model pointer-structures (1991) and the lazy λ-calculus (1992), which has been further investigated by SANGIORNI (1991).

Since the π-calculus was designed with pointer structures and the λ-calculus in mind, it seems natural to use it to encode a parallel graph reduction algorithm. We shall consider a variant of BOUDOL's asynchronous π-calculus (1992). This has the syntax:

$$P ::= \overline{x}[yz] \mid x(yz).P \mid P \mid P \mid \nu x.P \mid [x = y]P \mid [x \neq y]P \mid A(\tilde{x})$$

Here:

- $\overline{x}[yz]$ is the process which outputs the pair (y,z) along channel x.
- $x(yz).P$ is the process which inputs a pair (y',z') along channel x, then behaves like $P[x'/x, y'/y]$.
- $P \mid Q$ places P and Q in parallel.
- $\nu x.P$ creates a new channel x for use in P.
- $[x = y]P$ acts like P whenever $x = y$, and deadlocks otherwise.
- $[x \neq y]P$ acts like P whenever $x \neq y$, and deadlocks otherwise.
- $A(\tilde{x})$ is a recursive definition, in the style of MILNER (1989). We shall assume an environment of definitions $A(\tilde{x}) \stackrel{\text{def}}{=} P$, where $\mathrm{fv}\, P \subseteq \tilde{x}$.

The CHAM for this variant of the asynchronous π-calculus is given in Table 3, and is very similar to BOUDOL's CHAM for the asynchronous π-calculus (1992). The only new rules are:

- $\{\!|S|\!\} \rightleftharpoons S$, which is missing from BOUDOL's paper. This rule is required to prove the result that for any solution S there is a process P such that $S \rightleftharpoons^* \{\!|P|\!\}$. For example, we cannot show $\{\!|\{\!|\overline{x}[yz]|\!\}|\!\} \rightleftharpoons^* \{\!|\overline{x}[yz]|\!\}$ without this rule.

- $[x = x]P \rightleftharpoons P$ and $[x \neq y]P \rightleftharpoons P$ whenever $x \neq y$, which gives semantics for the conditional operators missing from BOUDOL's paper.
- $A(\tilde{x}) \rightleftharpoons P[\tilde{x}/\tilde{y}]$ whenever $A(\tilde{y}) \stackrel{\text{def}}{=} P$, which gives semantics for recursive definitions which were not used in BOUDOL's paper.

We can define much of the same vocabulary for this CHAM as we did for the graph reduction CHAM. In Proposition 10 we shall see that this abuse of notation is justified.

- A molecule is a positive ion with valency x iff it is $x(yz).P$.
- A molecule is a negative ion with valency x iff it is $\overline{x}[yz]$.
- A molecule is ionic iff it is a positive or negative ion.
- A solution is plasmic iff it is $\{\!|\nu\tilde{x}.\{\!|P_1, \ldots, P_n|\!\}|\!\}$ or $\{\!|P_1, \ldots, P_n|\!\}$ and each P_i is ionic. A plasma is positive (negative) iff it contains only positive (negative) ions.

We can give a translation of each molecule of the graph reduction CHAM into the π-calculus. This uses a special variable $*$, which we shall use to represent a function which is being evaluated, but which has not (yet) been given an argument. The semantics for terms is:

$$
\begin{aligned}
[\![x[\![z &\stackrel{\text{def}}{=} \overline{x}[*z] \\
[\![xy[\![z &\stackrel{\text{def}}{=} \overline{x}[yz] \\
[\![\lambda x.M[\![z &\stackrel{\text{def}}{=} !z(xy).([x = *][\![\lambda x.M[\![y \mid [x \neq *][\![M[\![y) \\
[\![\mathsf{rec}\, x := D\, \mathsf{in}\, M[\![z &\stackrel{\text{def}}{=} \nu x([\![D[\![x \mid [\![M[\![z) \qquad (x \neq z)
\end{aligned}
$$

where MILNER's (1991) *replication* operator is defined:

$$!P \stackrel{\text{def}}{=} !P \mid P$$

Note that the definition of $[\![\lambda x.M[\![$ is recursive, which is why we are taking recursion to be primitive, rather than replication. It is not obvious whether one could define semantics using replication for which Proposition 11 would hold. Note also that $[\![\mathsf{rec}\, x := D\, \mathsf{in}\, M[\![z$ is defined only when $x \neq z$, but we can use α-conversion on x to assure this. The semantics for declarations is:

$$
\begin{aligned}
[\![!M[\![z &\stackrel{\text{def}}{=} [\![M[\![z \\
[\![?M[\![z &\stackrel{\text{def}}{=} z(xy).(\overline{z}[xy] \mid [\![M[\![z)
\end{aligned}
$$

The semantics for molecules is:

$$[\![x := D[\![\stackrel{\text{def}}{=} [\![D[\![x$$

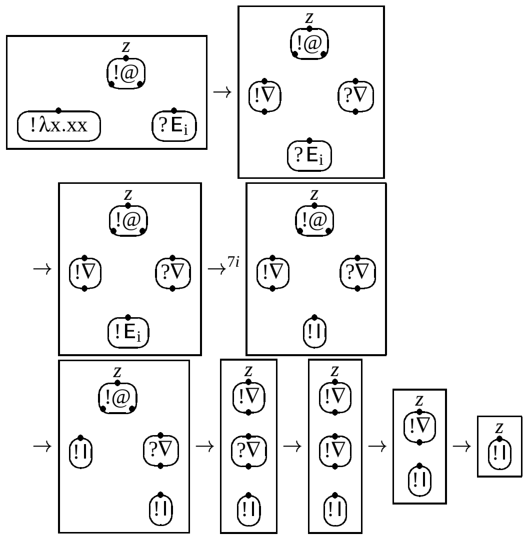
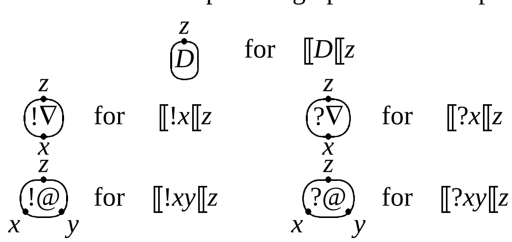


TABLE 4. A sample graph reduction in the π-calculus

$$\llbracket \nu x.m \rrbracket \stackrel{\text{def}}{=} \nu x.\llbracket m \rrbracket$$

$$\llbracket \{\!| m_1, \ldots, m_n |\!\} \rrbracket \stackrel{\text{def}}{=} \{\!| \llbracket m_1 \rrbracket, \ldots, \llbracket m_n \rrbracket |\!\}$$

This semantics can be drawn with process graphs. For example, if we draw:



Then the reduction of E_i given on page 5 can be drawn (with some extraneous processes removed to account for garbage collection) in Table 4. This is exactly the same reduction as given on page 5.

In general, we can show that each CHAM reduction is matched by exactly one π-calculus reduction, and thus that a CHAM solution diverges iff its π-calculus semantics diverges. This will allow us to show that the π-calculus semantics is adequate wrt the CHAM semantics for graph reduction (and so wrt the λ-calculus).

PROPOSITION 9.

1. *For any* S *there is a plasmic* S' *such that* $S \rightharpoonup S'$.

2. *If S is plasmic and $S \to S'$ then $S \mapsto \rightleftharpoons^* S'$.*
3. $S \to^\infty$ *iff* $S (\rightharpoonup^* \mapsto)^\infty$

PROOF. As for Proposition 2. $\square$

6 The pi-calculus semantics is adequate

We will show that the π-calculus semantics models the CHAM semantics for graph reduction without garbage collection. To begin with, we can justify an abuse of notation:

PROPOSITION 10.

1. *m is a (positive) (negative) ion iff $[\![m]\!]$ is a (positive) (negative) ion.*
2. *S is a (positive) (negative) plasma iff $[\![S]\!]$ is a (positive) (negative) plasma.*

PROOF. A simple case analysis. $\square$

Then we can show that each reduction of the graph reduction CHAM can be matched by a π-calculus reduction:

PROPOSITION 11.

1. *If $[\![S]\!] \rightharpoonup T$ then $T \rightharpoonup^* [\![S']\!]$ and $S \rightharpoonup_{\neq\gamma} S'$.*
2. *If $S \rightharpoonup_{\neq\gamma} S'$ then $[\![S]\!] \rightharpoonup\rightharpoonup^* [\![S']\!]$.*
3. *If $[\![S]\!] \mapsto T$ then $T \rightharpoonup^* [\![S']\!]$ and $S \mapsto S'$.*
4. *If $S \mapsto S'$ then $[\![S]\!] \mapsto\rightharpoonup^* [\![S']\!]$.*
5. $[\![S]\!] \to^\infty$ *iff* $S \to^\infty$

PROOF. Parts 1–4 are simple case analyses. From parts 1–4, $[\![S]\!] (\rightharpoonup^* \mapsto)^\infty$ iff $S\ (\rightharpoonup^*_{\neq\beta} \mapsto)^\infty$, so by Propositions 2.4 and 9.3, $[\![S]\!] \to^\infty$ iff $S \to^\infty$, which proves part 5. $\square$

This means that the π-calculus semantics is adequate.

THEOREM 12 (ADEQUACY). *If $[\![S]\!] \sqsubseteq [\![S']\!]$ then $S \sqsubseteq S'$.*

PROOF. For any C, if $C[S'] \to^\infty$ then $[\![C[S']]\!] \to^\infty$ so $[\![C]\!][[\![S']\!]] \to^\infty$ so $[\![C]\!][[\![S]\!]] \to^\infty$ so $[\![C[S]]\!] \to^\infty$ so $C[S] \to^\infty$. Thus, $S \sqsubseteq S'$. $\square$

However, it is not fully abstract.

THEOREM 13. *$m \sqsubseteq m'$ does not imply $[\![m]\!] \sqsubseteq [\![m']\!]$.*

PROOF. Define $\mathcal{R}$ to be a *simulation up to* $\rightleftharpoons^*$, similar to (MILNER, 1989, Definition 4.4), iff whenever $m \mathrel{\mathcal{R}} m'$ and $m' \to m'''$ then $m \to m''$ and $m'' \rightleftharpoons^* \mathcal{R} \rightleftharpoons^* m'''$. Let $\preceq$ be the largest simulation up to $\rightleftharpoons^*$, and let $\preceq^C$ be the largest pre-congruence contained in $\preceq$:

$$m \preceq^C m' \text{ iff } \forall C \,.\, C[m] \preceq C[m']$$

Then if $m \preceq m'$ and $m' \to^{\infty}$ then $m \to^{\infty}$, so if $m \preceq^{C} m'$ then $m \sqsubseteq m'$. Define:

$$
\begin{aligned}
m &= \{\!| x := !z, y := !z |\!\} \\
m' &= \{\!| x := !y, y := !z |\!\} \\
\mathcal{R} &= \{ (C[x := !z, y := !z] \\
&\qquad\quad [x := !z, y := !\lambda w_1.M_1, z := !\lambda w_1.M_1] \cdots \\
&\qquad\quad [x := !z, y := !\lambda w_n.M_n, z := !\lambda w_n.M_n], \\
&\quad\; C[x := !y, y := !z] \\
&\qquad\quad [x := !y, y := !\lambda w_1.M_1, z := !\lambda w_1.M_1] \cdots \\
&\qquad\quad [x := !y, y := !\lambda w_n.M_n, z := !\lambda w_n.M_n]) \\
&\quad\; | \; C \text{ is an } n+1\text{-place context} \}
\end{aligned}
$$

Then $\mathcal{R}$ is a simulation up to $\rightleftharpoons^*$, so for any context C, $C[m] \preceq C[m']$, so $m \preceq^{C} m'$, so $m \sqsubseteq m'$. We will now show that $[\![m [\![\not\sqsubseteq [\![m' [\![$. Define:

$$C[\cdot] = \{\!| \cdot, y(uv).\bot |\!\}$$

where $\bot$ is a divergent process such as:

$$\bot \stackrel{\text{def}}{=} \nu w.(\overline{w}[ww] \mid w(uv).\bot)$$

Then:

$$C[[\![m [\![] \rightleftharpoons^* \{\!| \overline{z}[*x], \overline{z}[*y], y(uv).\bot |\!\} \not\to$$

whereas:

$$C[[\![m' [\![] \rightleftharpoons^* \{\!| \overline{y}[*x], \overline{z}[*y], y(uv).\bot |\!\} \to \{\!| \overline{z}[*y], \bot |\!\} \to^{\infty}$$

so $[\![m [\![\not\sqsubseteq [\![m' [\![$. □

SANGIORNI (1991) has investigated λ-calculi for which MILNER's π-calculus translation is fully abstract. It is an open problem as to whether similar results can be shown for the CHAM for graph reduction.

Acknowledgements

Many thanks to Lennart Augustsson for getting the ball rolling, to Thomas Hallgren, Magnus Carlsson and Niklas Röjemo for conversations about the innards of LML, and to K. V. S. Prasad, Jenny Petersson, Matthew Hennessy, Edmund Robinson, Steve Schneider and Mark Jones for many discussions. This work was conceived at Chalmers University and written at Sussex University and Drive Cottage, Ampney Knowle, Cirencester.

References

ABRAMSKY, S. (1989). The lazy lambda calculus. In TURNER, D., editor, *Declarative Programming*. Addison-Wesley.

AUGUSTSSON, L. (1984). A compiler for lazy ML. In *Proc. ACM Symp. Lisp and Functional Programming*, pages 218–227.

BARENDREGT, H. (1984). *The Lambda Calculus*. North-Holland. Studies in logic 103.

BARENDREGT, H. P., VAN EEKELEN, M. C. J. D., GLAUERT, J. R. W., KENNAWAY, J. R., PLASMEIJER, M. J., and SLEEP, M. R. (1987). Term graph rewriting. In *Proc. PARLE 87*, volume 2, pages 141–158. Springer-Verlag. LNCS 259.

BERRY, G. and BOUDOL, G. (1990). The chemical abstract machine. In *Proc. 17th Ann. Symp. Principles of Programming Languages*.

BOUDOL, G. (1992). Asynchrony and the pi-calculus. INRIA Sophia-Antipolis.

CHURCH, A. (1941). *The Calculi of Lambda Conversion*. Princeton University Press.

HUGHES, R. J. M. (1984). *The Design and Implementation of Programming Languages*. D.Phil thesis, Oxford University.

JOHNNSON, T. (1984). Effecient compilation of lazy evaluation. In *Proc. Sigplan 84 Symp. Compiler Construction*, pages 58–69.

JONES, M. (1992). The Gofer technical manual. Part of the Gofer distribution.

LESTER, D. (1989). *Combinator Graph Reduction: A Congruence and its Applications*. D.Phil thesis, Oxford University.

MCCARTHY, J., ABRAHAMS, P. W., EDWARDS, D. J., HART, T. P., and LEVIN, M. I. (1962). *The Lisp 1.5 Programmers Kit*. MIT Press.

MILNER, R. (1989). *Communication and Concurrency*. Prentice-Hall.

MILNER, R. (1991). The polyadic π-calculus: a tutorial. In *Proc. International Summer School on Logic and Algebra of Specification*, Marktoberdorf.

MILNER, R. (1992). Functions as processes. *Math. Struct. in Comput. Science*, 2:119–141.

MILNER, R., PARROW, J., and WALKER, D. (1989). A calculus of mobile processes. Technical reports ECS-LFCS-89-86 and -87, LFCS, University of Edinburgh.

ONG, C.-H. L. (1988). *The Lazy Lambda Calculus: An Investigation into the Foundations of Functional Programming*. PhD thesis, Imperial College, London University.

PEYTON JONES, S. (1987). *The Implementation of Functional Programming Languages*. Prentice-Hall.

SANGIORGI, D. (1991). The lazy lambda calculus in a concurrency scenario. Technical Report ECS-LFCS-91-189, LFCS, Edinburgh University.

STOYE, W. R., CLARKE, T. J. W., and NORMAN, A. C. (1984). Some practical methods for rapid combinator reduction. In *Proc. ACM Symp. Lisp and Functional Programming*, pages 159–166.

TURNER, D. (1985). Miranda: A non-strict functional language with polymorphic types. In *Proc. IFIP Conf. Functional Programming Languages and Computer Architecture*. Springer-Verlag. LNCS 201.

WADSWORTH, C. P. (1971). *Semantics and Pragmatics of the Lambda Calculus*. D.Phil thesis, Oxford University.