A chemical abstract machine for graph reduction
Extended abstract

ALAN JEFFREY
COGS, UNIV. OF SUSSEX, BRIGHTON BN1 9QH, UK
ALANJE@COGS.SUSX.AC.UK

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:

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

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:

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

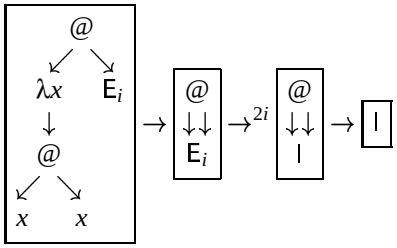
then $\mathsf{YI} \to^\infty$ but $\mathsf{KI}(\mathsf{YI}) \not\to^\infty$, whereas $\mathsf{KI}(\mathsf{YI}) \twoheadrightarrow^\infty$. However, the lazy evaluation strategy is very inefficient, since it may duplicate arguments when applying a function. For example, if we define:

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

Then $\mathsf{E}_i \twoheadrightarrow^{2i} \mathsf{I}$ but $\mathsf{E}_i \to^{2^{i+1}-2} \mathsf{I}$, 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.

This work was supported by SERC project GR/H 16537.

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. 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 $|\cdot|$ from the λ-calculus to the λ-calculus with rec:

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

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 a solution $\{\!|m_1, \ldots, m_n|\!\}$, the multiset brackets $\{\!|\cdots|\!\}$ are called a *membrane*. Let S range over solutions, and let $S \uplus S'$ be the multiset union of S and S'. Each CHAM has 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:

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

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 $\mathrm{fv}\, m$:

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

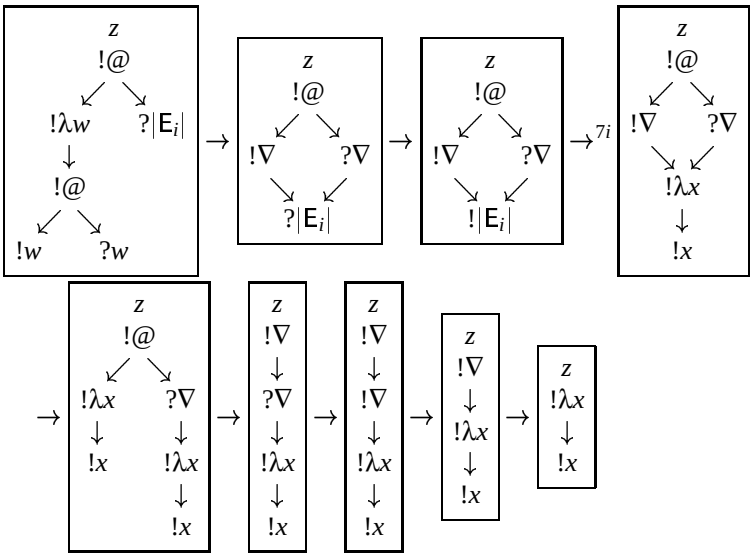
The defined variables of m are $\mathrm{dv}\, m$:

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

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:

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

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 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:

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

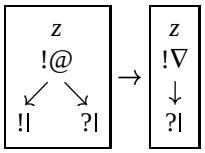
Hidden variables may be α-converted, exchanged and evaporated:

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

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:



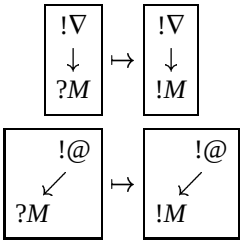
can be derived:

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

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:



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

TABLE 1. Summary of the graph reduction CHAM

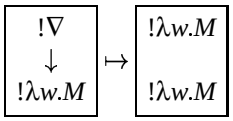
This models the first phase of graph reduction—we search along the spine of a graph, tagging nodes for evaluation. Note that strict reactions do not occur:

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

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:

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

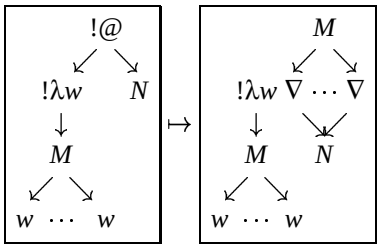
This can be drawn:



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

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

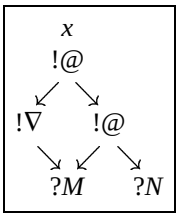
This can be drawn:



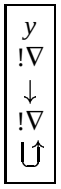
We shall sometimes write $\mapsto_\beta$ for this reaction, and $\mapsto_{\neq\beta}$ for any other reaction. This CHAM is summarized in Table 1.

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 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:

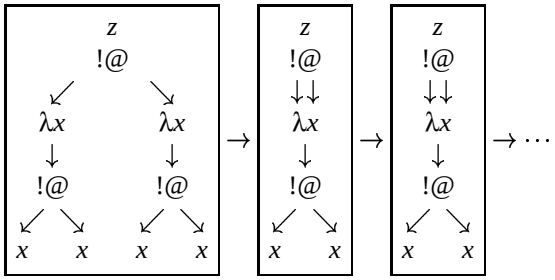


only one process will evaluate M. This is mirrored in the CHAM by the fact that M will only be reduced once. However, this algorithm produces some surprising results with cyclic graphs. The solution $\{|y := !\,\mathsf{rec}\,x := !x\,\mathsf{in}\,x|\}$ heats to become the plasma $\{|\nu x.\{|y := !x,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

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

and show that the CHAM semantics is *adequate*, that is if $(x := ?|E|) \sqsubseteq (x := ?|F|)$ then $E \sqsubseteq F$.

THEOREM 1 (ADEQUACY). *If* $(x := ?|E|) \sqsubseteq (x := ?|F|)$ *then* $E \sqsubseteq F$.

PROOF. Given in (JEFFREY, 1992). □

However, it is not fully abstract.

THEOREM 2. $E \sqsubseteq F$ *does not imply* $(x := ?|E|) \sqsubseteq (x := ?|F|)$.

PROOF. Given in (JEFFREY, 1992). □

It is an open problem as to whether the CHAM semantics is fully abstract wrt ABRAMSKY's (1989) λ-calculus with C, and as to whether the canonical semantics for the lazy λ-calculus $D \simeq (D \to D)_\perp$ is adequate wrt the CHAM semantics.

4 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 SANGIORGI (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 2, 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.

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

TABLE 2. CHAM for the π-calculus

We can define much of the same vocabulary for this CHAM as we did for the graph reduction CHAM.

- 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:

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

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 a semantics using replication for which there would be a one-to-one correspondence between CHAM reductions and π-calculus reductions. 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:

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

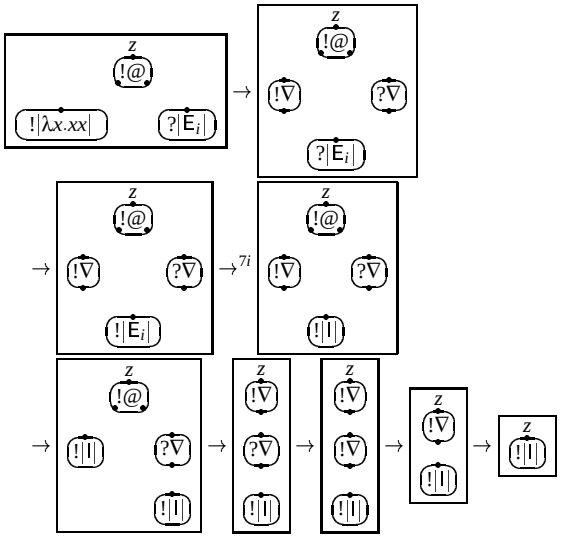
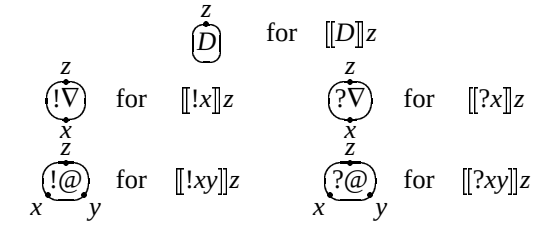


TABLE 3. A sample graph reduction in the π-calculus

The semantics for molecules is:

$$[\![x := D]\!] \stackrel{\text{def}}{=} [\![D]\!]x$$
$$[\![\nu x.m]\!] \stackrel{\text{def}}{=} \nu x.[\![m]\!]$$
$$[\![\{m_1, \ldots, m_n\}]\!] \stackrel{\text{def}}{=} \{[\![m_1]\!], \ldots, [\![m_n]\!]\}$$

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



Then the reduction of E_i given in section 1 can be drawn (with some extraneous processes removed to account for garbage collection) in Table 3. This is exactly the same reduction as given in Section 3.

In general, we can show that each CHAM reduction is matched by exactly one π-calculus reduction, and thus that the π-calculus semantics is adequate wrt the CHAM semantics for graph reduction (and so wrt the λ-calculus).

THEOREM 3 (ADEQUACY). *If* $[\![S]\!] \sqsubseteq [\![S']\!]$ *then* $S \sqsubseteq S'$.

PROOF. Given in (JEFFREY, 1992). □

However, it is not fully abstract.

THEOREM 4. *$m \sqsubseteq m'$ does not imply $[\![m]\!] \sqsubseteq [\![m']\!]$.*

PROOF. Given in (JEFFREY, 1992). □

SANGIORGI (1991) has investigated λ-calculi semantics 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.

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.

JEFFREY, A. (1992). A chemical abstract machine for graph reduction. Technical report 3/92, University of Sussex.

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.

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.