Directed HyperFlatLMNtal

sano

November 10, 2020

1 Abstract

An attempt to make HyperFlatLMNtal[2] to deal with directed graphs.

2 Background

LMNtal[1] is a language that can deal with graphs and the programs can be verified by Model Checker SLIM. However, the links in FlatLMNtal are only able to connect exactly two ports, which makes it hard to emulate pointer , n to 1 connection, related programs.

A list shared by 2 thread:

Two threads, th_A and th_B, are sharing one list, $[1, 2, 3]$. FlatLMNtal cannot deal with this situation.

$$th_A(X_1), th_B(X_1),$$
$$X_1 \mapsto c(1, X_2), X_2 \mapsto c(2, X_3), X_3 \mapsto c(3, X_4), X_4 \mapsto n$$

On the other hands, HyperFlatLMNtal, the extension of FlatLMNtal, has an excessive power, which can easily emulate pointers using Hyperlinks but an encoding of *dangling pointer* can easily occur.

HyperLMNtal representation of the above program:

The following program removes the first node of the list, which makes th_B to have a *dangling pointer* $!X_1$.

$$th_A(!X_1), th_B(!X_1),$$
$$!X_1 \bowtie c(1, !X_2), !X_2 \bowtie c(2, !X_3), !X_3 \bowtie c(3, !X_4), !X_4 \bowtie n,$$
$$(th_A(!X_1), !X_1 \bowtie c(1, !X_2) \vdash\ )$$

Same problem arises when we use membranes (cell) instead of Hyperlinks.

Capability typing[3] gives one solution to this problem but it is too much strict for this purpose.

Walking through a List:

The following program is il-typed in the capability typing system but it does not yields any null/dangling pointers and should be allowed.

$$R \mapsto walk(Prev), Prev \mapsto cons(Elem, Next)$$
$$\vdash R \mapsto walk(Next), Prev \mapsto cons(Elem, Next)$$

This is my attempt to avoid the former problematic situation yet providing some sense of flexibility as much as possible.

3 Syntax

3.1 Links and Atoms

- X denotes a link name. In the concrete syntax, link names are denoted by identifiers starting with capital letters.
- p denotes an atom name. In the concrete syntax, atom names are denoted by identifiers those are distinct from link names. The only reserved name is *alias*.

3.2 Processes

The syntax is given in Fig.1.

(process)P	$::= \mathbf{0}$	(null)
	$\mid X \mapsto p(X_1, \ldots, X_m)$	$(m \geq 0)$ (atom)
	$\mid (P, P)$	(molecule)
	$\mid \nu X.P$	(link creation)
	$\mid (P \vdash P)$	(rule)

Fig. 1. Syntax of Directed HyperFlatLMNtal

Given an link X, we define its *head* as an atom $X \mapsto p(X_1, \ldots, X_m)$. Also, the *tail* of the link X_i as the ith port of the atom. Here we shall also call X as the *incoming link* of the atom and X_i as the *outgoing link* of the atom.

An atom $X \mapsto alias(Y)$ is called a *aliasing*, which can be abbreviated as $X \mapsto Y$ and can be read as *aliasing X to Y*.

The set of the free link names in a process P is denoted as $fn(P)$ and is defined inductively as Fig.2.

$fn(\mathbf{0})$	$= \varnothing$
$fn(X \mapsto p(X_1, \ldots, X_m))$	$= \{X, X_1, \ldots, X_m\}$
$fn((P, Q))$	$= fn(P) \cup fn(Q)$
$fn(\nu X.P)$	$= fn(P) \backslash \{X\}$
$fn((P \vdash Q))$	$= \varnothing$

Fig. 2. Free link names of a process

We also define bound link names in a process P, $bn(P)$, as the relative complement of $fn(P)$ with respect to a set of the all link names appeared in P.

We assume that the set of free link names of the process which consists the whole program, the top level process, is an empty set. This can be easily achieved by just adding some extra link creations to the top level process if needed.

There are several conditions processes must satisfy.

No circular aliasing condition :

When applying the latter 2 conditions, aliasing should not form a cycle. That is, when traversing aliasing, there should be no same link name appears more than once.

Functional (right-unique) condition :

For all link $X \in fn(P)$, the *head* of X must not appear more than once in P.

Serial (left-total) Condition:

Given an process $\nu X.P$ where $X \in fn(P)$, the *head* of the X must exist in P .

3.3 Rules

Given a rule $(P \vdash Q)$, P is called the left-hand side and Q is called the right-hand side of the rule.

There are several conditions a rule $(P \vdash Q)$ must satisfy.

1. If the *head* of a link X occurs in P, the *head* of the link X must also occur in Q.
2. $fn(P) \supset fn(Q)$.

Notice HyperLMNtal needs a condition: the link creation must not appear in P, which is not required in this.

4 Operational Semantics

We first define structural congruence ($\equiv$) and then define the reduction relation ($\longrightarrow$) on processes.

4.1 Structural congruence

We define the relation $\equiv$ on processes as the minimal equivalence relation satisfying the rules shown in Fig.3.

Where $P[Y/X]$ is a link substitution that replaces all free occurrences of X with Y. If a free occurrence of X occurs in a location where Y would not be free, α-conversion (E1) may be required.

(E1)	$P \equiv P[Y/X]$
	where $X \in bn(P) \land Y \notin fn(P)$
(E2)	$(\mathbf{0}, P) \equiv P$
(E3)	$(P, Q) \equiv (Q, P)$
(E4)	$(P, (Q, R)) \equiv ((P, Q), R)$
(E5)	$P \equiv P' \Rightarrow (P, Q) \equiv (P', Q)$
(E7)	$\nu X.(X \mapsto Y, P) \equiv P[Y/X]$
(E8)	$\nu X.\mathbf{0} \equiv \mathbf{0}$
(E9)	$\nu X.\nu Y.P \equiv \nu Y.\nu X.P$
(E10)	$\nu X.(P, Q) \equiv (\nu X.P, Q)$
	where $X \notin fn(Q)$

Fig. 3. Structural congruence on processes

4.2 Reduction relation

We define the reduction relation $\longrightarrow$ on processes as the minimal relation satisfying the rules in Fig.4.

(R1)
$$\frac{P \longrightarrow P'}{(P, Q) \longrightarrow (P', Q)}$$

(R2)
$$\frac{P \longrightarrow P'}{\nu X.P \longrightarrow \nu X.P'}$$

(R3)
$$\frac{Q \equiv P \qquad P \longrightarrow P' \qquad P' \equiv Q'}{Q \longrightarrow Q'}$$

(R4)
$$(P, (P \vdash Q)) \longrightarrow (Q, (P \vdash Q))$$

Fig. 4. Reduction relation on processes

Example 1. Can the rule

$$(X \mapsto p(Y) \vdash X \mapsto q(Y))$$

rewrite an atom $X \mapsto p(X)$?

More precisely, the process

$$(X \mapsto p(X), (X \mapsto p(Y) \vdash X \mapsto q(Y)))$$

reduces to something?

The rule cannot be α-converted to the form

$$(X \mapsto p(X) \vdash \ldots)$$

However, the atom $X \mapsto p(Y)$ can be converted to $\nu Y.(Y \mapsto X, X \mapsto p(Y))$ by (E7).

Therefore, it can be rewritten as

$$\begin{aligned}
&(X \mapsto p(X), (X \mapsto p(Y) \vdash X \mapsto q(Y))) \\
&\equiv_{(\mathrm{E7})} \nu Y.(Y \mapsto X, (X \mapsto p(Y), (X \mapsto p(Y) \vdash X \mapsto q(Y)))) \\
&\longrightarrow \nu Y.(Y \mapsto X, (X \mapsto q(Y), (X \mapsto p(Y) \vdash X \mapsto q(Y)))) \\
&\equiv_{(\mathrm{E7})} (X \mapsto q(X), (X \mapsto p(Y) \vdash X \mapsto q(Y)))
\end{aligned}$$

As the above, we can match non-injective links using congruence rule on aliasing.

5 Needs for the program analysis

5.1 Abridged notation

Before moving on to more examples, we shall introduce some sugar syntax for convenience.

1. The parentheses of a molecule in a molecule can be omitted.

   For example,
   $$((P, Q), R)$$
   can be written as,
   $$(P, Q, R)$$
   .

2. The parentheses of the top level process, and the processes of left/right hand sides of the rules can be omitted.

   For example,
   $$(P, Q)$$
   can be written as,
   $$P, Q$$
   and
   $$((P, Q) \vdash R)$$
   can be written as,
   $$P, Q \vdash R$$
   .

3. The link creation of top level process can be omitted.

   For example,
   $$\nu A.\nu B.(A \mapsto a(A, B), B \mapsto b())$$
   can be written as,
   $$A \mapsto a(A, B), B \mapsto b()$$
   .

4. An atom
   $$\nu X.X \mapsto p(X_1, \ldots, X_n)$$
   where $\forall i(X_i \neq X)$

   can be written as,
   $$p(X_1, \ldots, X_n)$$
   .

5. the parentheses of an atom which has no outgoing link can be omitted.

6. $$\nu X_i.(X_0 \mapsto p(X_1, \ldots, X_i, \ldots, X_n), X_i \mapsto p(Y_1, \ldots, Y_n))$$
   where, $\forall j \neq i(X_j \neq X_i) \wedge \forall j(Y_j \neq X_i)$

   can be written as,
   $$X_0 \mapsto p(X_1, \ldots, X_{i-1}, p(Y_1, \ldots, Y_n), X_{i+1}, \ldots, X_n)$$
   .

5.2 An example that collapses

append cons

The following rule appends to lists.

$$\begin{aligned}
&R \mapsto \mathit{append}(\mathit{cons}(H, T), L) \\
&\vdash R \mapsto \mathit{cons}(H, \mathit{append}(T, L))
\end{aligned}$$

The above rule works fine but we need the "terminal" rule like the following, which yields an annoying problem.

append nil

The append of the nil and the list should just return the latter list.

$$R \mapsto \mathit{append}(\mathit{nil}, L) \vdash R \mapsto L$$

What happens if we apply it to the process

$$R \mapsto \mathit{append}(\mathit{nil}, R)$$

?

It will be like the following.

$$\begin{aligned}
&\nu R.(R \mapsto \mathit{append}(\mathit{nil}, R), (\ldots \vdash \ldots)) \\
&\equiv_{(\mathrm{E7})} \nu R.\nu L.(L \mapsto R, R \mapsto \mathit{append}(\mathit{nil}, L), (\ldots \vdash \ldots)) \\
&\longrightarrow \nu R.\nu L.(L \mapsto R, R \mapsto L, (\ldots \vdash \ldots)) \\
&\equiv_{(\mathrm{E7})} \nu R.(R \mapsto R, (\ldots \vdash \ldots)) \\
&\equiv_{(\mathrm{E7})} (\ldots \vdash \ldots)
\end{aligned}$$

This is actually... fine. The final result satisfies the conditions but how about this one?

What happens if we apply it to the process

$$R \mapsto append(nil, R), p(R)$$

?

Then will be like the following.

$$\begin{aligned}
&\nu R.(R \mapsto append(nil, R), p(R), (\ldots \vdash \ldots)) \\
&\equiv_{(E7)} \nu R.\nu L.(L \mapsto R, R \mapsto append(nil, L), p(R), (\ldots \vdash \ldots)) \\
&\longrightarrow \nu R.\nu L.(L \mapsto R, R \mapsto L, p(R), (\ldots \vdash \ldots)) \\
&\equiv_{(E7)} \nu R.(R \mapsto R, p(R), (\ldots \vdash \ldots)) \\
&\equiv_{(E7)} p(R), (\ldots \vdash \ldots)
\end{aligned}$$

Here the free link name R arises suddenly, which is surely undesirable.

5.3 The solutions

As shown in former example, the aliasing from a free link to a free link is somewhat dangerous. How should we avoid this ?

There should be at least 3 solutions.

- Prohibit the aliasing on the right-hand side of the rule.

  Then, for example, the above append-nil-rule can be rewritten with 2 rules

  $$\begin{aligned}
  &(R \mapsto append(nil, cons(H, T)) \vdash R \mapsto cons(H, T)), \\
  &(R \mapsto append(nil, nil) \vdash R \mapsto nil)
  \end{aligned}$$

  This is the simplest solution though it limits the expressiveness power.

- Check the rules (and the processes) can be well-typed with the capability-typing as following.

  Then, for example, the above append-nil-rule will be il-typed.

  $$\begin{aligned}
  &R \mapsto append(nil, R), p(R), && ① \\
  &(R \mapsto append(nil, L) \vdash R \mapsto L) && ②
  \end{aligned}$$

  By KCL, this should satisfy

  $$\begin{aligned}
  &-append/3 + append/2 + p/1 = 0 && \text{by } ① \\
  &append/3 = alias/1 && \text{by } ② \\
  &append/2 = alias/2 && \text{by } ②
  \end{aligned}$$

  By Conn, $alias/1 = alias/2$, therefore, $p/1 = 0$, which is il-typed.

- The easier solution is to just check whether the aliasing could match the not-injective links.

  For example, the above append-nil-rule will be prevented as following. Given an rule and the initial state

  $$\begin{aligned}
  &R \mapsto append(nil, R), p(R), && ① \\
  &(R \mapsto append(nil, L) \vdash R \mapsto L) && ②
  \end{aligned}$$

  , there we find a $R \mapsto L$ in right-hand side of the rule so we must check whether the R and L on the left-hand side, which is $append/3$ and $append/2$ respectively, of the rule form the loop or not.

  However, here, by ①, $append/3$ and $append/2$ is forming a loop. Therefore, it is not allowed.

6 Translation to the HyperLMNtal

An atom $X \mapsto p(X_1, \ldots, X_m)$ can be rewritten as $!X \bowtie p(X_1, \ldots, X_m)$.

Any rule $(P \vdash Q)$ can be rewritten as $(\nu X_1. \cdots .\nu X_n.P' \vdash \nu Y_1. \cdots .\nu Y_m.Q')$ $(n \geq 0, m \geq 0)$, a structurally congruent rule, where no link creation appears in P' and Q'.

Then, rewrite it as

```
P''
:- num(N_1), ···, num(N_n),
   new(Y_1), ··· , new(Y_n)
|  Q''
```

where the N_i are the numbers of occurrences of the link X_i in the P' and the `P''` and `Q''` is a inductively translated HyperLMNtal syntax forms of P' and Q'.

References

[1] Kazunori Ueda, LMNtal as a hierarchical logic programming language, Theoretical Computer Science, Volume 410, Issue 46, 2009, Pages 4784-4800, ISSN 0304-3975,

[2] Kazunori Ueda and Seiji Ogawa: HyperLMNtal: An Extension of a Hierarchical Graph Rewriting Model. Künstliche Intelligenz, Vol.26, No.1 (2012), pp.27-36. DOI:10.1007/s13218-011-0162-3.

[3] Ueda Kazunori. "Towards a Substrate Framework of Computation." In: Concurrent Objects and Beyond. Ed. by Gul Agha et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 341–366.

[4] Joachim Parrow : An Introduction to the π-Calculus, Chapter to appear in Handbook of Process Algebra, ed. Bergstra, Ponse and Smolka, Elsevier