Directed Hyper LMNtal

早稲田大学 基幹理工学部 情報理工学科 4年
佐野 仁

概要

動機

- （出来るだけ）LMNtal の利点を損なわずに
  有向ハイパーグラフを扱いたい

先行研究

- Capability Typing of HyperLMNtal

目的

- LMNtal の表現力の強化・応用の検討

貢献

- 操作的意味論をしっかりと定めた・POC の実装
- HyperLMNtal の操作的意味論の定義（副産物）

目次

目次

背景：グラフ書き換え系言語モデル LMNtal

LMNtal はグラフ書き換え系の言語

- **グラフを簡単に扱う** ことができる
  - 循環するグラフも可・ポインタ安全
- 可視化・モデル検査なども可能

LMNtal は言語モデル

- **ミニマム** で意味論がきちんと定義されている
  - HyperLMNtal は正直微妙だったので勝手に再定義したが
- （並行・並列）プログラミングの「基本・根本」を探るためのもの（だと勝手に思っている）
  - それは、**資源の共有** では？

背景：Increment in Promela

Promela (SPIN) の例題に関して

- これを LMNtal で検証したい（とする）

```
int x = 0;
proctype Proc() {
   int y = 0;  /* local variable */
   y = x;      /* LOAD           */
   y = y + 1;  /* INCREMENT      */
   x = y;      /* STORE          */
}
init{
   run Proc();
   run Proc();
   run Proc();
}
```

背景：Increment in LMNtal（プログラム）

LMNtal で書くと以下のようになる（はず）

```
proc(load, 0), proc(load, 0), proc(load, 0), x(0).

proc(load, $y), x($x)
 :- int($x), int($y)
 | proc(increment, $x), x($x).

proc(increment, $y)
 :- $y' := $y + 1
 | proc(store, $y').

proc(store, $y), x($x)
 :- int($x), int($y)
 | proc(halt, $y), x($y).
```

背景：Increment in LMNtal（問題点）

グローバル変数を「アトム名」で参照している

- 変数が複数あったときはどうする？
- 参照の受け渡し（indirection reference）はどうする？
- そもそもポインタのような複雑なもの（グラフ構造）を素直にエンコードできる？

```
proc(load, $y), x($x)
 :- int($x), int($y)
 | proc(increment, $x), x($x).
```

背景：Multi-threaded list traversing

前述の LMNtal の方式（アトム名での参照）は、
明らかに以下のような複雑な例に対しては無力

- どうする？

$$
\begin{aligned}
&thread(X_1),\ thread(X_1),\ thread(X_1),\\
&X_1 \longrightarrow cons(A_1, X_2),\\
&X_2 \longrightarrow cons(A_2, X_3),\\
&X_3 \longrightarrow cons(A_3, X_4),\\
&X_4 \longrightarrow cons(A_4, X_5),\\
&X_5 \longrightarrow cons(A_5, X_6),\\
&X_6 \longrightarrow nil.\\
&thread(Prev),\ Prev \longrightarrow cons(Elem, Next)\\
&\vdash thread(Next),\ Prev \longrightarrow cons(Elem, Next).
\end{aligned}
$$

これはただ辿るだけだけど、
ロックを掛けたり外したりしながら
資源を書き換えていくプログラムは
たくさんある c.f. 2020 前期 輪読本

背景：Increment in DHLMNtal

グローバル変数は（も）リンクで参照できる

- c.f. Name binding is easy with hypergraphs[7]

$$proc(load, X, 0), proc(load, X, 0), proc(load, X, 0), X \rightarrow 0.$$

$$proc(\mathbf{load}, X, \$y), X \rightarrow \mathbf{\$x}$$
$$\vdash int(\$x), int(\$y) \mid proc(increment, X, \mathbf{\$x}), X \rightarrow \mathbf{\$x}.$$

$$proc(\mathbf{increment}, X, \$y) \vdash \mathbf{\$y' := \$y + 1} \mid proc(store, X, \mathbf{\$y'}).$$

$$proc(\mathbf{store}, X, \mathbf{\$y}), X \rightarrow \$x$$
$$\vdash int(\$x), int(\$y) \mid proc(halt, X, \mathbf{\$y}), X \rightarrow \mathbf{\$y}.$$

背景：Shallow copy の必要性

LMNtal は ground がある（超強力）ので、
deep copy だけしていれば良い？

- Deep copy していると性能が悪くなる（ことがある）
  - hyperlink で置き換えたほうが性能が上がった例がある
    - 前期の FL インタプリタの環境（リスト）
    - しかも、cons アトム（大量に同じものがある）を探索するので Hyperlink にとって分が悪い例題（だと思う）
    - Hyperlink の traverse はかなり非効率なのに…
- shallow copy により参照が循環する（かもしれない）ことが重要な場合
  - 再帰関数の持つ環境など
- 親元のデータは共有したいとき

* これは純粋関数型にはほぼ不可能なので、これを LMNtal でやる（やれる）意味はとても大きい（と思う）

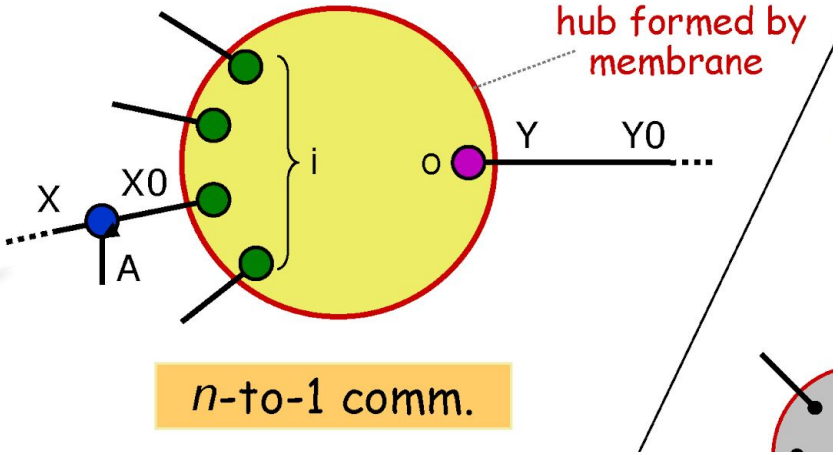
背景：N to 1 connection の必要性

LMNtal のリンクは 1 to 1 connection

- 「資源の共有」を表現するのが（Flat~だと）困難

膜かハイパーリンクを用いる必要がある

- 膜は記述量的にも処理系的にも重い
- ハイパーリンクだとかなり簡単に書けるが…



「膜だけで」ありとあらゆるものが
表せるというのは純粋に凄いとは思う

Dangling pointer：HyperLMNtal

ハイパーリンクは（膜も）「柔軟すぎて」
以下のようなプログラムも書けてしまう

- （もちろん dangling するかの検証もしたいので、意味がないわけではない
  →「null アトム」を用意すれば DHLMNtal でも書ける）

$$thread(!X),\ thread(!X),\ !X = resource.$$
$$thread(!X),\ !X = resource \vdash .$$

$$\sim\!> thread(!X).\ (\ldots \vdash).\ \%\ Dangling\ !!!$$

これでも良いなら c で書いて gdb なり使えば、となる（かも）

- 或いは関数型で参照リストの形で扱うとか
- LMNtal の良さは他にもあるので良いかもしれないけど

Dangling pointer：DHLMNtal

DHLMNtal では以下のプログラムは許容しない

$thread(X), thread(X), X \longrightarrow resource.$
$thread(X), X \longrightarrow resource \vdash .$ *% Compile Error !!!*

$\sim\!> thread(\boldsymbol{X}). (... \vdash).$ *% Never occurs !!!*

（もし null traversing するかが検証したいなら
以下のように書く）

$thread(X), thread(X), X \longrightarrow resource.$
$thread(X), X \longrightarrow resource \vdash X \longrightarrow \boldsymbol{null}. null \vdash.$

$\sim\!> thread(\boldsymbol{X}), \boldsymbol{X} \longrightarrow \boldsymbol{null}. (... \vdash).$

先行研究：Capability Typing

Hyperlink による dangling pointer （のエンコード）を
避ける手段として、capability typing が存在する

- しかし、この用途には制限が強すぎて、
  以下の例を弾いてしまう

$$thread(Prev), Prev \rightarrow cons(Elem, Next)$$
$$\vdash thread(Next), Prev \rightarrow cons(Elem, Next)$$

- 元々のプログラムが null free ならば、
  書き換え後も null free なはず
  - これは許容したい

動機：新たに導入したい利点・特徴

構造体とポインタを元にしたハイパーグラフ

- コンピュータ（のメモリ）を元に抽象化
  - 一般的なプログラム（に近いもの）を簡単に扱いたい
    - Shallow copy を含むプログラムを（も）書きたい
  - 無・双方向リンクのモデルと比較して実装が楽・効率的
    - c.f. L2C は単方向に向き付けできるものしか扱わない
    - 双方向リンクのモデリングも出来る
    - 低水準言語へのトランスパイルができる可能性が高い
- 局所・自由リンクとハイパーリンクの概念の統合
  - 任意の数のポートを持つ局所・自由リンクの導入

動機：損ないたくない LMNtal の利点

LMNtal の（良い）特徴は出来るだけ残したい

- Null・Dangling ポインタの排除
  - 構文条件（と型検査）により達成（したい）
- （細粒度の）並行性
  - （プログラムの実行において）「全体」に言及しない
  - 「num」（プログラム全体におけるハイパーリンクの個数を取得）は定義の上で使わない
- （単なるサブグラフへのマッチよりも）強力なパターンマッチング
  - LMNtal では（局所）リンクとハイパーリンクを分けてマッチできる
  - ＝ 共有している（かもしれない）資源へのマッチングと、そうでないものへのマッチングを区別できる

など

余談：Hyperlink の効率性

そもそも、Hyperlink は traverse していない

- `a(X), b(X) :- .`
  - findatom を一回して、
    そのリンク先を辿る


```
--memmatch:
    findatom      [1, 0, 'a'_1]
    deref         [2, 1, 0, 0]
    func            [2, 'b'_1]
```

- `a(!X), b(!X) :- .`
  - findatom を二回して、
    同一の hyperlink かを確認
  - バラバラに拾ってきて、
    毎回同じか確かめる


```
--memmatch:
    findatom      [1, 0, 'a'_1]
    findatom      [2, 0, 'b'_1]
--guard:
    derefatom     [3, 1, 0]
    ishlink         [3]
    isunary         [3]
    derefatom     [4, 2, 0]
    isunary         [4]
    samefunc        [3, 4]
```

余談：DHLMNtal（の理想）の traverse

DHLMNtal では、ほとんどのリンクは traverse できるはず

- $\nu X.(a(X), X \to b) \vdash .$
  - a を一回 findatom して、そのリンク先を辿る

Traverse 出来ないものもある（Hyperlink と同じ性能）

- 「同じものを指すか？」と言うもの

$a(X), b(X) \vdash .$

Capability typing の目的の一つは、
ポインタライクであるものを判別して
traverse できるようにすること（性能の向上）

- 先に簡単なもの（DHLMNtal）で実装・計測しながら予行演習する価値はある（かも）

余談：双方向リンク

双方向リンクも扱える

- リストの逆走査は前述のプログラムだと、
  現実装のハイパーリンクと同じくらい非効率

$$
\begin{aligned}
&\mathit{thread}(X_3),\ \mathit{thread}(X_3),\ \mathit{thread}(X_3),\\
&X_0 \longrightarrow \mathit{head},\\
&X_1 \longrightarrow \mathit{cons}(A_1, X_0, X_2),\\
&X_2 \longrightarrow \mathit{cons}(A_2, X_1, X_3),\\
&X_3 \longrightarrow \mathit{cons}(A_3, X_2, X_4),\\
&X_4 \longrightarrow \mathit{nil}.\\
&\mathit{thread}(\mathit{This}),\ \mathit{This} \longrightarrow \mathit{cons}(\mathit{Elem}, \mathit{Prev}, \mathit{Next})\\
&\vdash \mathit{thread}(\mathit{Prev}),\ \mathit{This} \longrightarrow \mathit{cons}(\mathit{Elem}, \mathit{Prev}, \mathit{Next}).
\end{aligned}
$$

目次

DHLMNtal : Syntax

$$
\begin{aligned}
P ::= {} & \mathbf{0} && \text{(null)} \\
& \mid X \rightarrow p(X1, ..., Xm) && \text{(atom)} \\
& \mid (P, P) && \text{(molecule)} \\
& \mid vX.P && \text{(link creation)} \\
& \mid (P \vdash P) && \text{(rule)}
\end{aligned}
$$

「create a $\boldsymbol{\nu}$ (= NEW) link X in P」
= リンク X は、
外のプロセスからは見えない

DHLMNtal：Free links　の定義

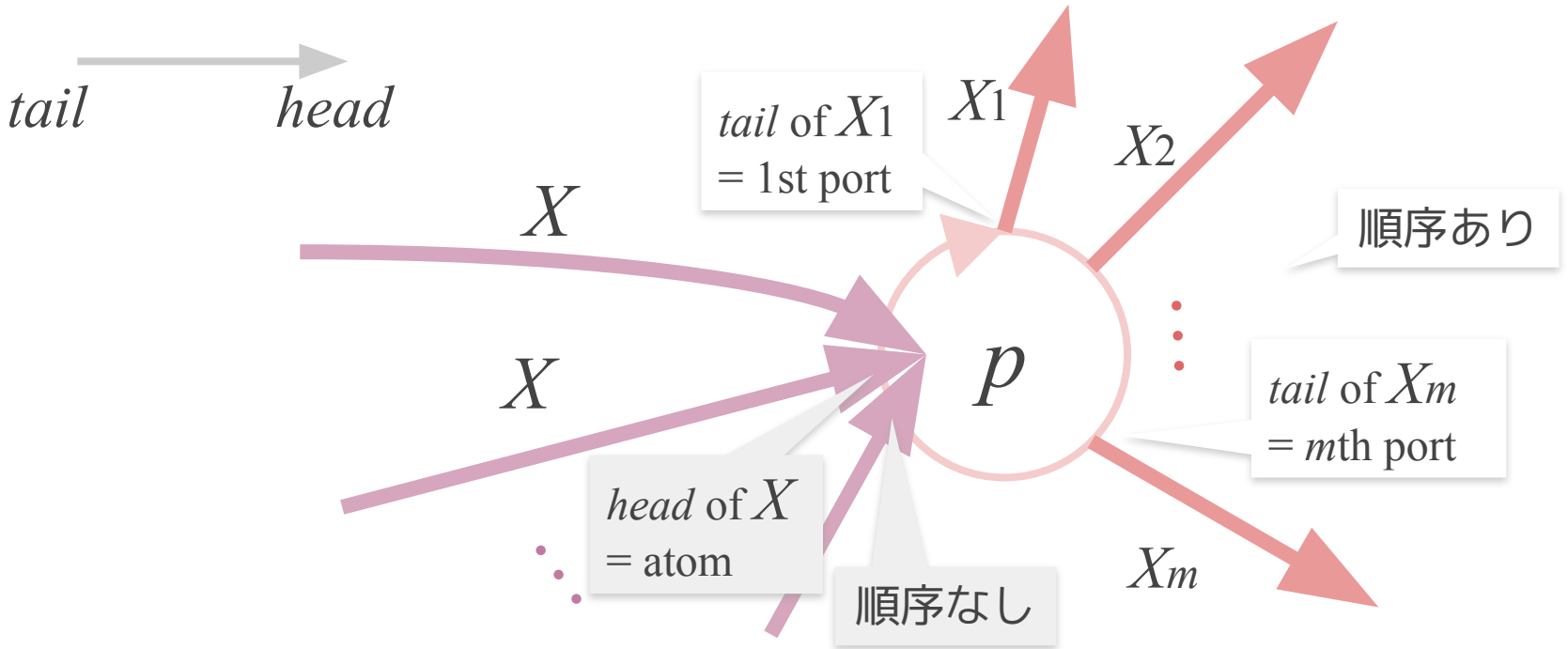
プロセス P の自由リンクの集合 $fl(P)$ を以下のように定める

$$
\begin{aligned}
fl(\mathbf{0}) &= \varnothing \\
fl(X \rightarrow p(X_1, ..., X_m)) &= \{X, X_1, ..., X_m\} \\
fl((P, Q)) &= fl(P) \cup fl(Q) \\
fl(\nu X.P) &= fl(P) \backslash \{X\} \\
fl((P \vdash Q)) &= \varnothing
\end{aligned}
$$

DHLMNtal：リンクの head・tail の定義

$X \to p(X_1, ..., X_m)$ があったとき、

- リンク X の *head* はアトム $X \to p(X_1, ..., X_m)$
  - グラフ理論では矢印の終点を *head* という
- リンク X_m の *tail* は上記アトムの i 番目のポート
  - グラフ理論では矢印の始点を *tail* という



DHLMNtal：Indirection の定義

特殊な1引数アトム $X \to Y$ を考える

- どうしても名前が欲しいのなら
  id とかということにする（何でも良い）
  - そうしたら id は予約語にしておく必要がある
  - その場合、↑は $X \to id(Y)$ の省略形（と定義する）

$Indirection\ (atom)$ と呼ぶことにする

- $X \to Y$ は、命令型言語風に書くと「X := Y」

DHLMNtal : Process Condition

プログラム全体のプロセス P において、$fl(P) = \varnothing$

- cf. LMNtal（コンパイラ）は
  「Singleton Link」と警告を出す

DHLMNtal : Link Conditions

循環する *Indirection* をなくしたときに、

Serial Condition

- $vX.P$ において $X \in fl(P)$ ならば
  X の *head* が P に出現する

Functional Condition

- P において $X \in fl(P)$ の *head* が P に出現し
  $Y \in fl(P)$ の *head* が P に出現するのならば
  $X \neq Y$

DHLMNtal : Rule Conditions

ルール $(P \vdash Q)$ において、

1. $fl(P) \supseteq fl(Q)$
2. P にルールが出現しない
3. P に $head$ が出現した任意の $X \in fl(P)$ に対し、
   同一識別子のリンク $X \in fl(Q)$ の $head$ が Q に出現する

目次

DHLMNtal：Structural Congruence

構造合同「≡」を
右の表を満たす
最小の同値関係として
定義

(E1) $(\mathbf{0}, P) \equiv P$

(E2) $(P, Q) \equiv (Q, P)$

(E3) $(P, (Q, R)) \equiv ((P, Q), R)$

(E4) $P \equiv P' \Rightarrow (P, Q) \equiv (P', Q)$

(E5) $\nu X.P \equiv \nu Y.P[Y/X]$
where $Y \notin fl(P)$

(E6) $\nu X.\nu Y.P \equiv \nu Y.\nu X.P$

(E7) $\nu X.(X \rightarrow Y, P) \equiv \nu X.P[Y/X]$

(E8) $\nu X.\mathbf{0} \equiv \mathbf{0}$

(E9) $\nu X.(P, Q) \equiv (\nu X.P, Q)$
where $X \notin fl(Q)$

DHLMNtal : Reduction Relation

遷移関係「~>」を以下の表を満たす
最小の同値関係として定義

$$\text{(R1)}\quad \frac{P \sim> P'}{(P, Q) \sim> (P', Q)}$$

$$\text{(R2)}\quad \frac{P \sim> P'}{\nu X.P \sim> \nu X.P'}$$

$$\text{(R3)}\quad \frac{Q \equiv P \quad P \sim> P' \quad P' \equiv Q'}{Q \sim> Q'}$$

$$\text{(E4)}\quad (P, (P \vdash Q)) \sim> (Q, (P \vdash Q))$$

DHLMNtal：非単射的なマッチング

(E7) を援用することにより、
非単射的なマッチングも出来る

$$
\begin{aligned}
& X \to p(X), (X \to p(Y) \vdash X \to q(Y)) \\
\equiv_{\text{(E7)}} & \; \nu Y.(Y \to X, X \to p(Y), (X \to p(Y) \vdash X \to q(Y))) \\
\rightsquigarrow & \; \nu Y.(Y \to X, X \to q(Y), (X \to p(Y) \vdash X \to q(Y))) \\
\equiv_{\text{(E7)}} & \; X \to q(Y), (X \to p(Y) \vdash X \to q(Y))
\end{aligned}
$$

目次

DHLMNtal：略記法の導入

更に例を紹介する前に、簡便さのため略記法を導入しておく

- $vX.(X \to p(X_1, ..., X_m))$ where $\forall i \in [1, \mathrm{m}]. X \neq X_i$ は $p(X_1, ..., X_m)$ と略記できる
- $vX_i.(X_0 \to p(X_1, ..., X_i, ..., X_m), X_i \to q(Y_1, ..., Y_n))$ where $\forall j \in [0, \mathrm{m}] \backslash \{i\}. \forall k \in [1, \mathrm{n}]. X_i \neq X_j \wedge X_i \neq Y_k$ は $X_0 \to p(X_1, ..., q(Y_1, ..., Y_n), ..., X_m)$ と略記できる
- プログラム全体の一番外側の *link creation* は適宜省略
- あとは LMNtal と同じ

HyperLMNtal：再定義版について

ここでの HyperLMNtal は自分が再定義したものを指す

- Redefining_HyperFlatLMNtal.pdf
  - （時間のあるときに）コメントください
- 素朴な定義だと、fusion がおかしくなる
  - sano/B4後期日誌
- 「num」はない→次スライド

（意味論上）「num」はあってはいけない

「num」は、
「**プログラム全体** のハイパーリンク（のポート）の個数」

- しかし、LMNtal （の意味論）は
  「ルールで書き換える部分以外は言及しない」
  ことが重要（と勝手に思っている）
  - （すごくすごくうまく実装すれば）書き換えの際に
    「stop the world」しなくて良い（細粒度の並行性）
  - 分散システムへの応用を考えていた（ように見える）
- 「LMNtal は並行（～並列）言語モデル」を謳うなら、
  「全体を止める・言及する」ものは
  意味論にあんまり含めたくない
  （もちろん現実・実装的には num もあった方が良いと思うけど…）

HyperLMNtal：構文、操作的意味論

$$
\begin{aligned}
P ::= \ & \mathbf{0} \\
& \mid p(!X_1, \ldots, !X_m) \\
& \mid (P, P) \\
& \mid vX.P \\
& \mid (P \vdash P)
\end{aligned}
$$

構文

ほとんど全部
DHLMNtal と同じ

$$
\begin{aligned}
fl(\mathbf{0}) &= \varnothing \\
fl(p(!X_1, \ldots, !X_m)) &= \{X_1, \ldots, X_m\} \\
fl((P, Q)) &= fl(P) \cup fl(Q) \\
fl(vX.P) &= fl(P) \backslash \{X\} \\
fl((P \vdash Q)) &= \varnothing
\end{aligned}
$$

自由リンク

遷移規則は DHLMNtal と全く同じ

(E1) $(\mathbf{0}, P) \equiv P$

(E2) $(P, Q) \equiv (Q, P)$

(E3) $(P, (Q, R)) \equiv ((P, Q), R)$

(E4) $P \equiv P' \Rightarrow (P, Q) \equiv (P', Q)$

(E5) $vX.P \equiv vY.P[!Y/!X]$
where $Y \notin fl(P)$

(E6) $vX.vY.P \equiv vY.vX.P$

(E7) $vX.(!X \bowtie !Y, P) \equiv vX.P[!Y/!X]$

(E8) $vX.\mathbf{0} \equiv \mathbf{0}$

(E9) $vX.(P, Q) \equiv (vX.P, Q)$
where $X \notin fl(Q)$

構造合同規則

HyperLMNtal：構文条件

ルール $(P \vdash Q)$ において、

1. $fl(P) \supseteq fl(Q)$
2. P にルールが出現しない
3. P に *link creation* が出現しない
   - **HyperLMNtal ではルール左辺に「new」を書けない**
     - DHLMNtal にはない制約（DHLMNtal では良い）
   - 「スコープを限定したマッチ」（＝ 局所リンク）が出来ない
     - 「num」を使って再現することになる
     - ↑（意味論的には）良くない

DHLMNtal と同じ

局所ハイパーリンク：ルール左辺での new

DHLMNtal

$$vX.(a(X), b(X), X \rightarrow c).$$
$$\boldsymbol{vX}.(a(\boldsymbol{X}), \boldsymbol{X} \rightarrow c) \vdash . \qquad \% \textit{ Does not match}$$

HyperLMNtal

自分の再定義版

$$vX.(a(!X), b(!X), c(!X)).$$
$$\boldsymbol{vX}.(a(\boldsymbol{!X}), c(\boldsymbol{!X})) \vdash . \qquad \% \textit{ Error : Cannot write like this}$$

現実装風に書いたもの

```
:- uniq, new($x) | a($x), b($x), c($x).
new($x), a($x), c($x) :- .          % Error
a($x), c($x) :- num($x) =:= 2 |.    % Using “num”
```

DHLMNtal で 無方向 Hyperlink

DHLMNtal のリンクは方向がつく

- が、無方向のリンクも簡単にエンコードできる

$$a(X), b(X), c(X), X \to link.$$

```
a(!X), b(!X), c(!X).
```

Fusion もできる（ちょっと面倒だけど）

```
a(!X), b(!Y) :- !X >< !Y.
```

$$a(X), b(Y), X \to link, Y \to link \vdash X \to Y, Y \to link.$$
$$a(X), b(X), X \to link \vdash X \to link.$$
$$link \vdash.$$

HyperLMNtal との比較：まとめ

HyperLMNtal にできて、DHLMNtal では許されないこと

- 無方向の（ハイパー）リンク
  - ただし容易にエンコードできる
- 各種構文条件に反すること
  - ただし、null アトムの例のように（安全に）回避できる

DHLMNtal にできて、HyperLMNtal でできないこと

- ルール左辺に new を書くこと
  - = 局所ハイパーリンク
  - ガードに num を書けば再現は出来る
    - が、num は（意味論的には）マズい（全体への言及）

目次

Dangling pointer：problem

ここまでに述べた構文条件だけでは
排除出来ない（したくない）ものも存在する

- 以下のルールは一見無害そうに見える
  （nil にリストを繋ぐ場合はリストをそのまま返す）

$$R \to append(nil, L) \vdash R \to L$$

- が、自己ループしているリンクに
  非単射的なマッチングを行うと、
  Serial condition が
  崩れる可能性がある

$$X \to append(nil, X), p(X),$$
$$(R \to append(nil, L) \vdash R \to L)$$
$$\sim> X \to X, p(X), (... \vdash ...)$$
$$\equiv p(X), (... \vdash ...)$$

Dangling pointer：solution

解決策としては（主に）2種類ある

- ルール右辺での自由リンクの *Indirection* を認めない

$$R \to append(nil, cons(H, T)) \vdash R \to \mathrm{cons}(H, T).$$
$$R \to append(nil, nil) \vdash R \to nil.$$

  - 先述のルールは二つに分けて書く必要がある
  - 表現力を損なう

- ルール右辺で *Indirection* している自由リンクの指すアトムに対して capability type checking を行う
  - 今考えているのはこっち

Capability type checking

先述のプログラムを capability type checking してみる

$$X \to append(nil, X), p(X), \quad \cdots ①$$
$$(R \to append(nil, L) \vdash R \to L) \quad \cdots ②$$

KCL

- ① の X に注目して 0 = - append/0 + append/2 + p/1
- ② の R に注目して - append/0 = - →/0
- ② の L に注目して append/0 = →/1

Conn（→/0 = →/1）より

- p/1 = 0（ダメ）

目次

POC の実装：現状

Proof Of Concept を実装した

- sano-jin/vertex: A proof of concept of DHLMNtal.
- Flat 版（ガードなし）
- Type check は未実装（実行時エラーを吐く）

Directed
Hyper
LMNtal

インタプリタ：具体構文

$\mathbf{0}$	`()`
$X \rightarrow p(X1, ..., Xm)$	`X -> p(X1, ..., Xm)`
(P, P)	`(P, P)`
$vX.P$	`\X.P`
$(P \vdash P)$	`(P :- P)`

インタプリタ：使い方

```
stack exec vertex-exe “sample.dhl”
```

で “sample.dhl” ファイルのプログラムを実行する

余談：

- 拡張子は別になんでも良い
- dhl は Deepnet HTTP Server Log の拡張子（らしい）
- あなたは.dhl拡張子に関する有用な情報を
  持っている場合は、私たちに書いてください！
  - よくわからないが、怪しげであった

インタプリタ：実行例（append）

実行

```
append(cons(a, cons(b, nil)), cons(c, nil)).

R -> append(cons(H, T), L) :- R -> cons(H, append(T, L)).
R -> append(nil, L) :- R -> L.
```

```
0:
L1 -> a. L2 -> b. L3 -> nil. L4 -> cons(L2, L3). L5 -> cons(L1, L4). L6 -> c. L7 -> nil. L8
-> cons(L6, L7). append(L5, L8).
R -> append(cons(H, T), L) :- R -> cons(H, append(T, L)). R -> append(nil, L) :- R -> L.

1: R -> append(cons(H, T), L) :- R -> cons(H, append(T, L)) ~>
L1 -> a. L2 -> b. L3 -> nil. L4 -> cons(L2, L3). L5 -> append(L4, L8). L6 -> c. L7 -> nil. L8
-> cons(L6, L7). cons(L1, L5).
R -> append(cons(H, T), L) :- R -> cons(H, append(T, L)). R -> append(nil, L) :- R -> L.

2: R -> append(cons(H, T), L) :- R -> cons(H, append(T, L)) ~>
L1 -> a. L2 -> b. L3 -> nil. L4 -> append(L3, L8). L5 -> cons(L2, L4). L6 -> c. L7 -> nil. L8
-> cons(L6, L7). cons(L1, L5).
R -> append(cons(H, T), L) :- R -> cons(H, append(T, L)). R -> append(nil, L) :- R -> L.

3: R -> append(nil, L) :- R -> L ~>
L1 -> a. L2 -> b. L5 -> cons(L2, L8). L6 -> c. L7 -> nil. L8 -> cons(L6, L7). cons(L1, L5).
R -> append(cons(H, T), L) :- R -> cons(H, append(T, L)). R -> append(nil, L) :- R -> L.
```

dumper をまだちゃんと実装していないのでかなり見難いが…

目次

まとめ：得られた知見

ポインタや構造体を表現する Hypergraph を
null・dangling free に扱うのは意外と難しい

- 制約は厳しすぎてもいけない（Capability typing）
- 形式的な証明については今後も考えていきたい

DHLMNtal の操作的意味論を定める過程で、
HyperLMNtal の操作的意味論も定義出来た

- DHLMNtal に関する、定義・実装の試みは、
  HyperLMNtal へそのまま応用できる可能性が
  非常に高い

参考文献

1. 上田和紀, 加藤紀夫: 言語モデル LMNtal, コンピュータ ソフトウェア, 21(2), 2004.
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. ALIMUJIANG Yasen: Hypergraph-Based Modeling of Formal Systems Involving Name Binding
4. Alimujiang Yasen and Kazunori Ueda, Name Binding is Easy with Hypergraphs. IEICE Transactions on Information and Systems, Vol.E101-D, No.4, 2018, pp.1126-1140, DOI: 10.1587/transinf.2017EDP7257.
5. 乾敦行，工藤晋太郎，原耕司，水野謙，加藤紀夫，上田和紀： 階層グラフ書換えモデルに基づく統合プログラミング言語LMNtal， コンピュータソフトウェア，Vol.25, No.1 (2008), pp.124-150.