



# 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

---

背景・動機

構文

操作的意味論

HyperLMNtal との比較

Capability type checking の必要性

実装

まとめ

# 目次

---

## 背景・動機

構文

操作的意味論

HyperLMNtal との比較

Capability type checking の必要性

実装

まとめ

# 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なになる（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？

$thread(X_1), thread(X_1), thread(X_1),$

$X_1 \rightarrow cons(A_1, X_2),$

$X_2 \rightarrow cons(A_2, X_3),$

$X_3 \rightarrow cons(A_3, X_4),$

$X_4 \rightarrow cons(A_4, X_5),$

$X_5 \rightarrow cons(A_5, X_6),$

$X_6 \rightarrow nil.$

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

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

これはただ迎るだけだけど、  
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<sup>[7]</sup>

$$\text{proc}(\text{load}, X, 0), \text{proc}(\text{load}, X, 0), \text{proc}(\text{load}, X, 0), X \rightarrow 0.$$
$$\text{proc}(\text{load}, X, \$y), X \rightarrow \$x$$
$$\vdash \text{int}(\$x), \text{int}(\$y) \mid \text{proc}(\text{increment}, X, \$x), X \rightarrow \$x.$$
$$\text{proc}(\text{increment}, X, \$y) \vdash \$y' := \$y + 1 \mid \text{proc}(\text{store}, X, \$y').$$
$$\text{proc}(\text{store}, X, \$y), X \rightarrow \$x$$
$$\vdash \text{int}(\$x), \text{int}(\$y) \mid \text{proc}(\text{halt}, X, \$y), X \rightarrow \$y.$$

# 背景：Shallow copy の必要性

---

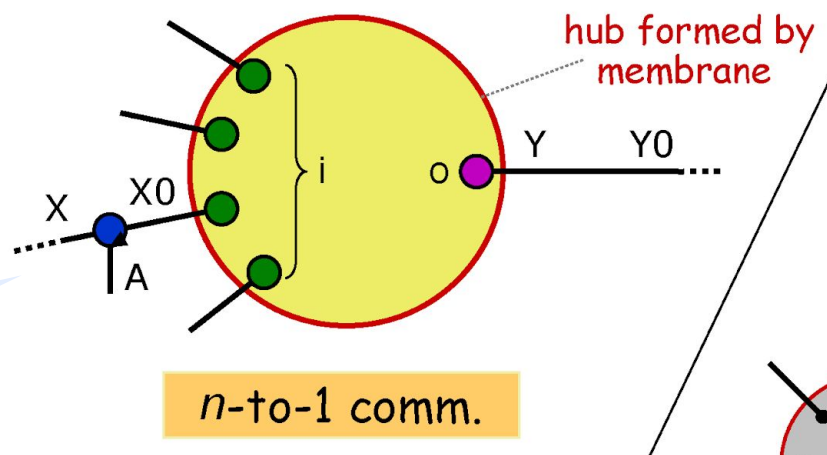
LMNtal は ground がある（超強力）ので、  
deep copy だけしていい？

- 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 \* 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）
  - 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

# 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 ⊢ .*

*~> thread(!X). (... ⊢). % 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

$thread(X), thread(X), X \rightarrow resource.$

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

$\sim > thread(\mathbf{X}). (... \vdash).$  *% 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)

$thread(X), thread(X), X \rightarrow resource.$

$thread(X), X \rightarrow resource \vdash X \rightarrow \text{null}. null \vdash.$

$\sim > thread(X), X \rightarrow \text{null}. (... \vdash).$

# 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

$$\begin{aligned} & thread(Prev), Prev \rightarrow cons(Elem, Next) \\ & \vdash thread(Next), Prev \rightarrow cons(Elem, Next) \end{aligned}$$

- 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:

<b>findatom</b>	[1, 0, 'a'_1]
<b>deref</b>	[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:

<b>findatom</b>	[1, 0, 'a'_1]
<b>findatom</b>	[2, 0, 'b'_1]

--guard:

derefatom	[3, 1, 0]
ishlink	[3]
isunary	[3]
derefatom	[4, 2, 0]
isunary	[4]
<b>samefunc</b>	[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

- $vX.(a(\mathbf{X}), \mathbf{X} \rightarrow b) \vdash .$ 
  - 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？」  
というもの

$a(\mathbf{X}), b(\mathbf{X}) \vdash .$

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

$thread(X3), thread(X3), thread(X3),$

$X_0 \rightarrow head,$

$X_1 \rightarrow cons(A_1, X_0, X_2),$

$X_2 \rightarrow cons(A_2, X_1, X_3),$

$X_3 \rightarrow cons(A_3, X_2, X_4),$

$X_4 \rightarrow nil.$

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

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

# 目次

---

背景・動機

構文

操作的意味論

HyperLMNtal との比較

Capability type checking の必要性

実装

まとめ

# DHLMNtal : Syntax

---

$P ::= \mathbf{0}$	(null)
$X \rightarrow p(X_1, \dots, X_m)$	(atom)
$(P, P)$	(molecule)
$\nu X.P$	(link creation)
$(P \vdash P)$	(rule)

「create a  $\nu$  (= 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

---

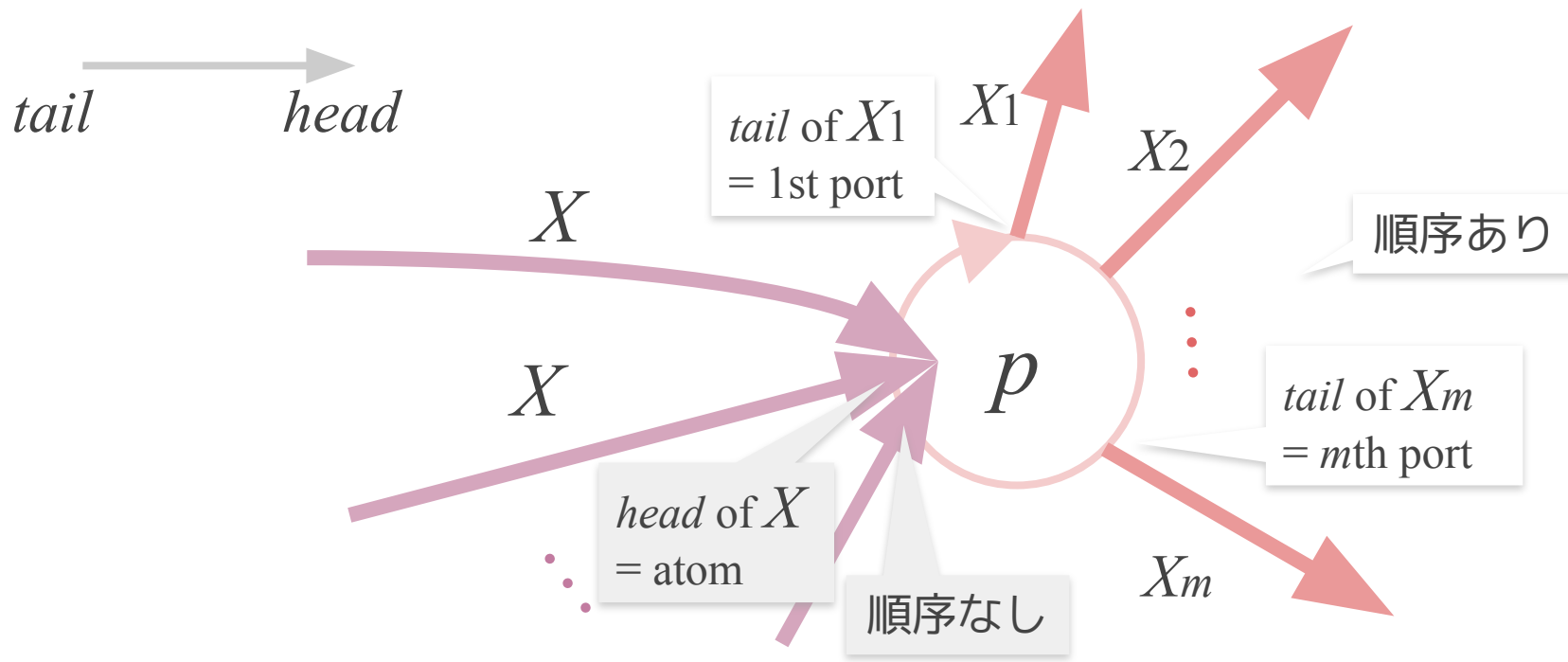
プロセス  $P$  の自由リンクの集合  $fl(P)$  を以下のように定める

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

# DHLMNtal : リンクの head・tail の定義

$X \rightarrow p(X_1, \dots, X_m)$  があったとき、

- リンク  $X$  の *head* はアトム  $X \rightarrow p(X_1, \dots, X_m)$ 
  - 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  $X \rightarrow Y$  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、 $\uparrow$  は  $X \rightarrow id(Y)$  の省略形 (と定義する)

*Indirection (atom)* と呼ぶことにする

- $X \rightarrow Y$  は、命令型言語風にかくと「 $X := Y$ 」

# DHLMNtal : Process Condition

---

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

- 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

---

背景・動機

構文

操作的意味論

HyperLMNtal との比較

Capability type checking の必要性

実装

まとめ

# DHLMNtal : Structural Congruence

構造合同「 $\equiv$ 」を  
右の表を満たす  
最小の同値関係として  
定義

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

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

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

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

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

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

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

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

$$(E9) \quad \nu X.(P, Q) \equiv (\nu X.P, Q) \\ \text{where } X \notin fl(Q)$$

# DHLMNtal : Reduction Relation

---

遷移関係「 $\sim>$ 」を以下の表を満たす  
最小の同値関係として定義

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

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

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

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

# 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

$$\begin{aligned} & X \rightarrow p(X), (X \rightarrow p(Y) \vdash X \rightarrow q(Y)) \\ & \equiv (E7) \nu Y. (Y \rightarrow X, X \rightarrow p(Y), (X \rightarrow p(Y) \vdash X \rightarrow q(Y))) \\ & \leadsto \nu Y. (Y \rightarrow X, X \rightarrow q(Y), (X \rightarrow p(Y) \vdash X \rightarrow q(Y))) \\ & \equiv (E7) X \rightarrow q(Y), (X \rightarrow p(Y) \vdash X \rightarrow q(Y)) \end{aligned}$$



# 目次

---

背景・動機

構文

操作的意味論

HyperLMNtal との比較

Capability type checking の必要性

実装

まとめ

# 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

- $\nu \mathbf{X}.(\mathbf{X} \rightarrow p(X_1, \dots, X_m))$  where  $\forall i \in [1, m]. X \neq X_i$  は  $p(X_1, \dots, X_m)$  と略記できる
- $\nu \mathbf{X}i.(X_0 \rightarrow p(X_1, \dots, \mathbf{X}i, \dots, X_m), \mathbf{X}i \rightarrow q(Y_1, \dots, Y_n))$   
where  $\forall j \in [0, m] \setminus \{i\}. \forall k \in [1, n]. X_i \neq X_j \wedge X_i \neq Y_k$  は  $X_0 \rightarrow p(X_1, \dots, q(Y_1, \dots, Y_n), \dots, X_m)$  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 もあった方が良くと思うけど…)

# HyperLMNtal : 構文、操作的意味論

$P ::= \mathbf{0}$

$| p(!X_1, \dots, !X_m)$

$| (P, P)$

$| \nu X.P$

$| (P \vdash P)$

ほとんど全部  
DHLMNtal と同じ

構文

$fl(\mathbf{0}) = \emptyset$

$fl(p(!X_1, \dots, !X_m)) = \{X_1, \dots, X_m\}$

$fl((P, Q)) = fl(P) \cup fl(Q)$

$fl(\nu X.P) = fl(P) \setminus \{X\}$

$fl((P \vdash Q)) = \emptyset$

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

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

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

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

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

構造合同規則

# HyperLMNtal : 構文条件

---

ルール  $(P \vdash Q)$  において、

1.  $fl(P) \supseteq fl(Q)$

DHLMNtal と同じ

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

# 局所ハイパーリンク：ルール左辺での new

## DHLMNtal

$\nu X.(a(X), b(X), X \rightarrow c).$

$\nu X.(a(\mathbf{X}), \mathbf{X} \rightarrow c) \vdash .$       *% Does not match*

## HyperLMNtal

自分の再定義版

$\nu X.(a(!X), b(!X), c(!X)).$

$\nu X.(a(\mathbf{!X}), c(\mathbf{!X})) \vdash .$       *% 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

$$a(\textcolor{red}{X}), b(\textcolor{red}{X}), c(\textcolor{red}{X}), \textcolor{red}{X} \rightarrow \textit{link}.$$
$$a(!X), b(!X), c(!X).$$

Fusion もできる (ちょっと面倒だけど)

$$a(!X), b(!Y) :- !X >< !Y.$$
$$a(\textcolor{red}{X}), b(\textcolor{red}{Y}), \textcolor{red}{X} \rightarrow \textit{link}, \textcolor{red}{Y} \rightarrow \textit{link} \vdash \textcolor{red}{X} \rightarrow \textcolor{red}{Y}, \textcolor{red}{Y} \rightarrow \textit{link}.$$
$$a(\textcolor{red}{X}), b(\textcolor{red}{X}), \textcolor{red}{X} \rightarrow \textit{link} \vdash \textcolor{red}{X} \rightarrow \textit{link}.$$
$$\textit{link} \vdash.$$



# 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 アトムの場合のように（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

---

背景・動機

構文

操作的意味論

HyperLMNtal との比較

Capability type checking の必要性

実装

まとめ

# Dangling pointer : problem

ここまで述べてき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)

$$R \rightarrow \text{append}(\text{nil}, L) \vdash R \rightarrow L$$

- 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

$$\begin{aligned} & X \rightarrow \text{append}(\text{nil}, X), p(X), \\ & (R \rightarrow \text{append}(\text{nil}, L) \vdash R \rightarrow L) \\ & \leadsto X \rightarrow X, p(X), (\dots \vdash \dots) \\ & \equiv p(\textcolor{red}{X}), (\dots \vdash \dots) \end{aligned}$$

# 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

$$R \rightarrow \text{append}(\text{nil}, \text{cons}(H, T)) \vdash R \rightarrow \text{cons}(H, T).$$
$$R \rightarrow \text{append}(\text{nil}, \text{nil}) \vdash R \rightarrow \text{nil}.$$

- 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

$$X \rightarrow \text{append}(\text{nil}, X), p(X), \quad \dots \textcircled{1}$$

$$(R \rightarrow \text{append}(\text{nil}, L) \vdash R \rightarrow L) \quad \dots \textcircled{2}$$

KCL

- ① の  $X$  に注目して  $0 = - \text{append}/0 + \text{append}/2 + p/1$
- ② の  $R$  に注目して  $- \text{append}/0 = - \rightarrow/0$
- ② の  $L$  に注目して  $\text{append}/0 = \rightarrow/1$

Conn ( $\rightarrow/0 = \rightarrow/1$ ) より

- $p/1 = 0$  (ダメ)

# 目次

---

背景・動機

構文

操作的意味論

HyperLMNtal との比較

Capability type checking の必要性

実装

まとめ

# 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

---

**0**

$X \rightarrow p(X_1, \dots, X_m)$

$(P, P)$

$\forall X.P$

$(P \vdash P)$

**()**

$X \rightarrow p(X_1, \dots, X_m)$

$(P, P)$

$\backslash X.P$

$(P :- P)$



# 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

---

背景・動機

構文

操作的意味論

HyperLMNtal との比較

Capability type checking の必要性

実装

まとめ

# 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.