2I1-4

階層グラフ書換えモデルを拡張したHyperLMNtal の実現

HyperLMNtal: an extension of a hierarchical graph rewriting language model

小川誠司*1 Seiji OGAWA　目黒学*1 Manabu MEGURO　上田和紀*2 Kazunori UEDA

*1早稲田大学大学院基幹理工学研究科 Graduate School of Fundamental Science and Engineering, Waseda University

*2早稲田大学理工学術院 Faculty of Science and Engineering, Waseda University

LMNtal (pronounced "elemental") is a concurrent language model based on hierarchical graph rewriting and unifies various computational models. We introduced *hyperlinks* into LMNtal to represent multipoint connectivity, and expanded it to a hierarchical hypergraph rewriting language model, *HyperLMNtal*. HyperLMNtal enabled concise description of computational models involving flexible and diverse forms of references between data, and enabled natural and efficient encoding of a constraint processing description language CHR. This paper describes the design and implementation of hyperlinks and introduces programming in HyperLMNtal by means of examples. We also show that our implementation has achieved favorable results in terms of both performance and computational complexity.

1. はじめに

社会における IT (情報技術) システムの役割が大きくなる一方で, システムを構成するプログラムの誤動作が重大な事故, 事件に繋がるケースも報告されている. LMMtal[1] はこれらのシステムの動作を簡潔に記述することができる高い表現力 [2] と, 記述したシステムの安全性や信頼性を検証可能な処理系 [3] を有するプログラミング言語である. より大規模で複雑なシステムへの適用を目指し, 言語の表現力の拡充や実行性能の向上, システム検証機能の改善が進められている.

LMNtal は階層グラフ構造の書換えに基づく並行計算モデルであり, 接続構造や階層構造を動的に変化させることによって複雑なデータ構造の操作を伴うプログラムを簡潔に記述することができる. しかし従来の LMNtal では, ハイパーグラフに代表されるデータ間の一対一以外の参照関係を, 実用的な実行性能の下で表現することはほぼ不可能であった. そこで我々は, ハイパーグラフを記述するための新たなデータ構造を設計し, LMNtal 処理系に実装を行うことで, LMNtal の言語モデルを階層ハイパーグラフ書換えモデル (HyperLMNtal) へと拡張した. そして並行制約プログラミング言語 CHR[4] など, これまで LMNtal では表現困難であったモデルを従来よりも自然に記述可能とし, かつ理想的な計算量での実行を実現した.

本稿では今回処理系に実装を行ったハイパーリンクの導入背景と目的, 設計と実装手法について解説し, ハイパーリンクを用いて記述した例題の記述性や実行性能について述べる.

2. 言語モデル LMNtal

LMNtal の基本データ構造である階層グラフは, アトムを基本構成要素として, それを膜とリンクの二つの手段で構造化したものである. 膜はアトムの多重集合を構成し, 多重集合は入れ子にできる. またリンクはアトム同士を一対一で接続する. どちらの構造も動的再構成が可能である.

2.1 基本構文と省略構文

LMNtal の構文は図 1 のように定義される. ただし本論文で使用しない構文要素は省いている. X_i は**リンク名**, p は**アトム名**であり, P は**プロセス**である. 具象構文ではリンクは大文字から始まる識別子, アトム名は小文字から始まる識別子で表現する. T はプロセスの書換え規則の表現に用いる**プロセステンプレート**であり, 局所文脈 (特定のセルの内部での文脈) を扱う機能をもつ.

0 は中身のないプロセス, $p(X_1, \ldots, X_m)$ は m 価のアトム, P, P はプロセスの並列合成である. `{`P`}` は**セル**と呼び, プロセスを**膜** `{ }` によってグループ化したものである. T `:-` T はプロセスの書換え規則で, 膜が形成する階層構造の同じ場所に置かれたプロセスを適用対象とする. 予約アトム名 `=` は**コネクタ**と呼ばれ, X `=` Y はリンク X の一端と Y の一端とを接続する機能をもつ. **ルール文脈**は膜の中のすべてのルールの多重集合とマッチし, **プロセス文脈**は膜の中のルール以外のプロセスのうち, 明示的に指定されていないもの全体とマッチする.

木構造のデータを他言語と同様に記述するための**項記法**が用意されており, アトム a の第 k 引数として, (リンク名のかわりに) 最終引数を省略したアトム b を書くと, a の第 k 引数と b の最終引数とがリンク接続されているものとみなす. たとえば `f(g(x))` は `f(A),g(B,A),x(B)` と等しい.

2.2 拡張構文

LMNtal における数値は, `8(X)` のような 1 価アトムで表現する. 引数はその数値を参照するプロセスにつながる.

アトムが整数型などの基本型に属するかどうかの検査や基本型に対する演算などを指定するために, **型付きプロセス文脈**および**ガード付きルール**という拡張構文を導入している.

通常のプロセス文脈のマッチする相手が膜 (階層構造) によって決まるのに対し, 型付きプロセス文脈のマッチする相手はグラフ構造 (接続構造) とグラフ中のアトム名によって決まる. たとえばガード付きルール

```
p(X),$n[X] :- int($n),$n>0 | p(Y),$n[Y],p(Z),$n[Z]
```

は, 1 価アトム `p` が正整数アトムにつながっている場合, その構造の複製を作る. ガード条件 `int($n)` は, `$n[X]` が整数アトムを表す型付きプロセス文脈であることを求め, `$n>0` はその整数値が正であることを求めている. 条件 `$n>0` は条件 `int($n)` を含意するので後者は省くことができ, さらに型付きプロセス文脈をリンクと同じ記法で書くことも認めているので, 上記のルールは `p(N) :- N>0 | p(N),p(N)` と書いてもよい. 現

連絡先: 小川誠司, 早稲田大学大学院基幹理工学研究科情報理工学専攻, 現在キヤノン (株), 〒 169-8555 新宿区大久保 3-4-1 63 号館 5 階 02 号, 03-5286-3340, seiji(at)ueda.info.waseda.ac.jp

$$\begin{array}{rrl} \text{(プロセス)} & P ::= & \mathbf{0} \mid p(X_1,\ldots,X_m) \mid P,P \mid \{P\} \mid T \texttt{ :- } T \\ \text{(プロセステンプレート)} & T ::= & \mathbf{0} \mid p(X_1,\ldots,X_m) \mid T,T \mid \{T\} \mid T \texttt{ :- } T \mid \texttt{@}p \mid \texttt{\$}p \end{array}$$

図 1: LMNtal の構文

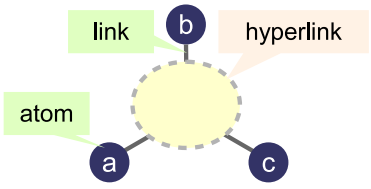

図 2: ハイパーリンクのイメージ図

処理系では `int` 以外に `unary` や `ground` 型などをガード条件として指定でき, それぞれ 1 価アトムおよび自由リンクを 1 本だけ持つ無階層グラフを表す.

ルール左辺の膜の後に “`/`” を付加すると, それ以上簡約不能なセルにしかマッチしなくなる. ルールの前に $name$`@@` と書くことでルールに $name$ という名前をつけることができる. また膜にも名前 (膜名) をつけることができる. 名前 m をもつ膜は $m\{\ldots\}$ と表記する. 膜名はルール名と異なり単なる注釈ではなく, ルール左辺の膜は同一の名前を持つ膜とだけマッチする. ルール左辺に “`\`” を記述すると, “`\`” よりも右側にあるルールの左辺のみを, ルール右辺へと書き換える.

LMNtal のルールは適用可能な限り繰り返し適用されるが, 実際のプログラムでは, あるルールを同等なグラフ構造に対して一度だけ適用させたい場合もある. `uniq` 制約をガードに持つルールは, 過去に当該ルールの左辺にマッチした構造を記憶し, 同型な構造に対して高々一度だけルール適用を許す条件を付加される. `uniq` 制約は型に関する制約ではないが, ガード制約の一つと位置づけられている.

2.3 LMNtal 処理系

LMNtal 処理系には Java によって開発されたコンパイラとランタイム, さらに C 言語によって開発されたランタイム (SLIM) が存在する. コンパイラは LMNtal ソースコードをテキスト形式の中間命令列に変換するので, SLIM は Java で書かれたコンパイラを利用することができる. また SLIM にはシステム検証機能が備わっている.

2.4 実行方式

LMNtal プログラムの実行はルールによる階層グラフ構造の書換えの繰り返しである. 階層グラフの初期構造はルールと共にプログラムによって与えられ, ルールの適用が不可能になった時点で実行を終了する. LMNtal の実行には適用するルールの選択とルールによる書換え箇所の選択の 2 つの非決定性が存在し, その選択は処理系に委ねられている. 以下では, ある時点の階層グラフに対するルール適用の可否の検査をテストと呼び, ルールが持つヘッド部に合致する部分グラフ構造をグラフ全体から探索する処理をマッチングと呼ぶ.

3. HyperLMNtal への拡張

本研究では新たなデータ構造である**ハイパーリンク**を LMNtal に導入し, 言語モデルを階層ハイパーグラフ構造書換えに基づく言語モデル (HyperLMNtal) へと拡張した. ハイパーグラフとは, 数学におけるグラフを一般化 (拡張) したものであり, エッジ (ハイパーエッジ) が任意個数のノードを連結することができる構造である.

$$\begin{array}{rl} \text{(生成)} & Head \texttt{ :- new}(X) \mid Atom(X) \\ \text{(型制約)} & Head \texttt{ :- hlink}(X) \mid Body \\ \text{(併合)} & Atom(H_1),\, Atom(H_2) \texttt{ :- } H_1 \texttt{ >< } H_2 \\ \text{(要素数取得)} & Atom(H_1) \texttt{ :- } H_2 = \texttt{num}\ (H_1) \mid Body \\ \text{(探索)} & Atom(\texttt{\$x}),\, Atom(\texttt{\$x}) \texttt{ :- } Body \end{array}$$

図 3: ハイパーリンク関連の記法

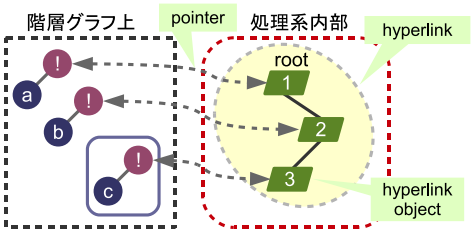

図 4: ハイパーリンクの概要図

3.1 背景と目的

LMNtal や CHR はルールによるデータ集合の書換えを計算処理としており, 非効率的なマッチングが発生すれば実行性能が悪化する. これらの言語にとって, 非効率的なマッチングの防止は実行性能向上に有効であると言える.

LMNtal が扱う階層グラフ構造上では, アトム間の参照関係はリンクによる一対一の接続のみであり, ハイパーグラフに代表される一対一以外の参照関係を, 現実的な実行性能 (特に時間計算量) の下で表現することはほぼ不可能である.

そこで本研究では, LMNtal の言語モデルを階層ハイパーグラフ書換えモデルへと拡張することを目指す. 具体的には, LMNtal のリンクを拡張した概念であるハイパーリンクを導入することで, CHR の論理変数などの実現困難であった計算モデルを従来よりも自然に記述可能にし, かつその接続関係をマッチングに活かすことで, 理想的な計算量での実行を可能にすることを目指した.

3.2 関連研究

CHR (Constraint Handling Rules) は多重制約集合の書換えに基づく言語モデルであり, 実アプリケーションへの利用実績もある汎用的な並行制約プログラング言語である. CHR の構文は膜の無い LMNtal と似通っており, LMNtal の有用性の証明のため, CHR プログラムは LMNtal を用いてエンコード可能 [5] であることを示してきた.

CHR は論理変数によってデータ間の複雑な参照構造を簡潔に表現可能である. またその参照関係を利用したマッチング最適化機能 [6] が CHR 処理系に実装されており, LMNtal よりも優れた計算量で実行可能な例題が存在する.

3.3 ハイパーリンクの設計

LMNtal 上でハイパーグラフを表現するため, 新たにハイパーリンクと呼ばれるデータ構造を考案した. ハイパーリンクは, 概念としてはハイパーグラフのハイパーエッジに照らし合わせてリンクを拡張するのが自然である. しかし, LMNtal の現行の構文との整合性を取り易くするために, 図 2 のように通常のリンク同士をつなぐハブ構造の役割を持つ構造を導入することで, ハイパーリンクを構成する.

```
refl @ leq(X,X) <=> true.
anti @ leq(X,Y),leq(Y,X) <=> X=Y.
idem @ leq(X,Y),leq(Y,Z) ==> leq(X,Z).
tran @ leq(X,Y)\leq(X,Y) <=> true.
```

図 5: 不等式制約ソルバの記述例 (chr)

```
refl@@ leq(X1,X2),{+X1,+X2,v(V),$p} :- int(V) | {$p}.
anti@@ leq(X1,Y1),leq(Y2,X2),
  {+X1,+X2,v(VX),$p},{+Y1,+Y2,v(VY),$q} :-
    int(VX),int(VY) | {$p,$q,v(VX)}.
idem@@ leq(X1,Y1),leq(X2,Y2),
  {+X1,+X2,v(VX),$p},{+Y1,+Y2,v(VY),$q} :-
    int(VX),int(VY) |
    leq(X1,Y1),{+X1,v(VX),$p},{+Y1,v(VY),$q}.
tran0@@ leq(X1,Y1),leq(Y2,Z2),
  {+X1,v(X),$p},{+Y1,+Y2,v(Y), q},{+Z2,v(Z),$r} :-
    uniq(X,Z),int(Y) |
    leq(X1,Y1),leq(Y2,Z2),leq(Z3,X3),{+X1,+X3,v(X),$p},
    {+Y1,+Y2,v(Y),$q},{+Z2,+Z3,v(Z),$r}.
tran1@@ leq(X1,Y1),leq(Y2,Z2),
  {+X1,+Z2,v(X),$p},{+Y1,+Y2,v(Y),$q} :-
    uniq(X),int(Y) |
    leq(X1,Y1),leq(Y2,Z2),leq(Z3,X3),
    {+X1,+X3,+Z2,+Z3,v(X),$p},{+Y1,+Y2,v(Y),$q}.
```

図 6: 不等式制約ソルバの記述例 (lmn-mem)

ハイパーリンクは階層グラフを構成するデータ構造の一つであるため, 通常のリンクと同様にルールよる操作が可能でなければならない. そこで新たに図 3 に示すようなハイパーリンク関連のガード制約, 演算子などを追加した.

3.4 ハイパーリンクの実装手法

新たな記法を追加するためにコンパイラの改良を行った. また実用的な処理系として開発, 公開されている SLIM への追加実装という形でハイパーリンクの実装を行った.

図 4 はハイパーリンクの概要図である. 2 引数の “!” をファンクタに持つアトム (**エクスクラメーションアトム**) に対し, ハイパーリンクを表す構造体 (**ハイパーリンクオブジェクト**) を対応付ける. “!” アトムの第 1 引数はハイパーリンクに直接接続されているアトムへのリンクであり, 第 2 引数にはハイパーリンクオブジェクトへのポインタを埋めこんでいる. 処理系内部でこのオブジェクトの集合をハイパーリンクとして扱う.

異なるハイパーリンク同士の接続は, 異なるオブジェクト集合の併合に相当する. そこでオブジェクトの管理には, 所属集合の解決と集合同士の併合を高速に処理可能な既存の Union-Find algorithm[7] を応用している. この手法がサポートしていない要素の削除などの独自処理も組み込んでいる.

4. 記述例とその評価

HyperLMNtal の評価実験として CHR のベンチマーク問題から 6 題を選択し, これを従来の LMNtal (lmn-int, lmn-mem) と HyperLMNtal (hyperlmn) の計 3 種類の LMNtal プログラムへとエンコードし, それぞれの記述性と実行性能を比較した. 本章ではその内の 4 題について解説している.

lmn-int は等しい整数アトムを同じハイパーグラフに属する要素とみなす手法であり, 比較的簡潔に擬似的ハイパーグラフ構造を記述出来るが, アトム間に実際の接続関係が無いためマッチングが非効率的である. 対して lmn-mem は膜を用いて図 2 のようなリンクのハブ構造を表現する手法である. 膜を介した実際の接続関係をマッチングに活かすことで効率的なマッチングが可能であるが, 構造が複雑化するために記述が困難であり, さらには冗長なルール適用が必要になる場合もある.

```
refl@@ leq($x,$x) :- .
anti@@ leq($x,$y),leq($y,$x) :- $x><$y.
idem@@ leq(X,$y),leq($y,Z)\ :- uniq(X,Z) | leq(X,Z).
tran@@ leq($x,$y)\leq($x,$y) :- .
```

図 7: 不等式制約ソルバの記述例 (hyperlmn)

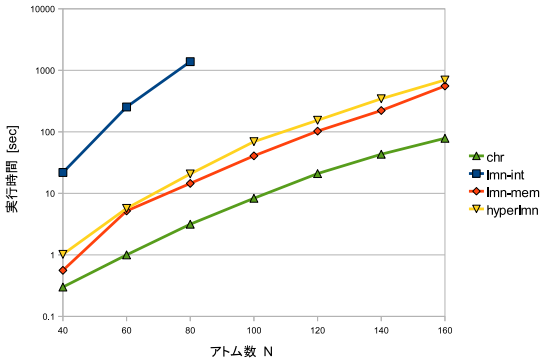

図 8: 不等式制約ソルバの実験結果

```
cycle@@ e($x,$y),e($y,$z),e($z,$x) :- cycle($x,$y,$z).
```

図 9: 閉路探索の記述例 (hyperlmn)

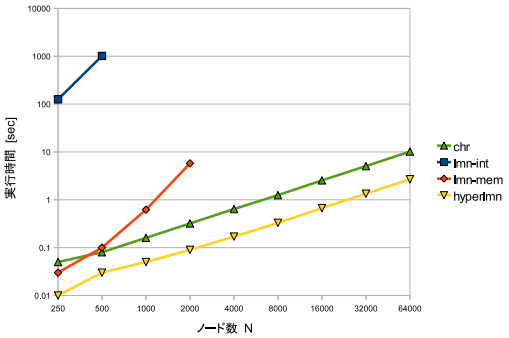

図 10: 閉路探索の実験結果

実行環境のハードウェアは Intel(R) Core(TM)2 Quad 2.60GHz, RAM 4GB, OS は ubuntu 9.10 である.

4.1 不等式制約ソルバ

図 5 は CHR の例題である不等式制約ソルバの記述例であり, leq(X,Y) は $X \leq Y$ を意味する. 例えば leq(A,B), leq(B,C) に対してはルール tran が適用され, 制約集合に leq(A,C) が一度だけ追加される.

図 6, 7 はこの例題をそれぞれ lmn-mem, hyperlmn に基づいて記述した例である. lmn-mem では膜を用いて論理変数を表現しているために, 他の手法よりプログラムが複雑化している. これに対し hyperlmn では CHR の論理変数をほぼ一対一にエンコード出来ていることが分かる.

このプログラムに対し, N 個の変数が 1 本の循環構造的な不等式関係をなしている制約群を与えた場合の実行時間を測定した. 例えば $N = 3$ の場合は leq(X,Y),leq(Y,Z),leq(Z,X) を与える. 結果を図 8 に示す. lmn-int は非効率的なマッチングの発生によって時間計算量が悪化しているのに対し, 他の 3 記法は同程度の性能を達成している.

4.2 閉路探索

2 点 X, Y 間のエッジを e(X,Y) で表し, その集合として与えられたグラフから 3 頂点の閉路を探索し消去する例題である. hyperlmn の記述例を図 9 に示す. 3 頂点の閉路は

```
make@@ make(A) :- root(A,O).
union@@ union(A,B) :- find(A,X), find(B,Y), link(X,Y).
findN@@ '~>'($a,B), find($a,link(find(X))) :-
    hlink(X) | find(B,link(find(X))), '~>'($a,X).
findR@@ root($b,R)\find($b,X) :- X=$b.
linkEq@@ link($a,$a) :- .
linkL@@ link($a,$b),root($a,NA),root($b,NB) :-
    NA>=NB, NB1=NB+1 |
    '~>'($b,$a), NA1=max(NA,NB1), root($a,NA1).
linkR@@ link($b,$a), root($a,NA), root($b,NB) :-
    NA>=NB, NB1=NB+1 |
    '~>'($b,$a), NA1 = max(NA,NB1), root($a,NA1).
max1@@ H=max(A,B) :- A=<B | H=B.
max2@@ H=max(A,B) :- A>B  | H=A.
```

図 11: Union-Find algorithm の記述例 (hyperlmn)

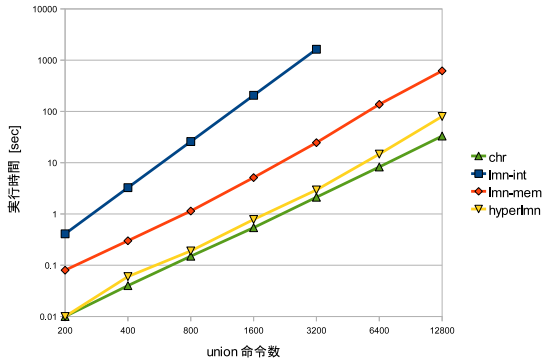

図 12: Union-Find algorithm の実験結果

e($x,$y),e($y,$z),e($z,$x) と表現できるため, 閉路の探索は 1 本のルールで記述することが出来る. このプログラムに頂点数 N の中で N 個の閉路を形成している初期エッジ群を与えた場合の実験結果を図 10 に示す. 構造が複雑化するために, lmn-int だけでなく lmn-mem も時間計算量が悪化していることが分かる. 対して hyperlmn は chr と同程度の計算量を実現しており, 単純な実行速度も chr を上回っている.

4.3 Union-Find algorithm

図 11 は HyperLMNtal を用いた Union-Find algorithm の記述例である. 要素を生成する N 個の make 命令と, それらを一つの集合に併合する $N-1$ 個の union 命令を初期命令として与える. N の値を変化させて実行した結果を図 12 に示す. lmn-int は非効率的なマッチングを発生させてしまう記法であるため, 時間計算量が悪化していることが分かる. それ以外の 3 種類の記法は計算量の面ではほぼ同等の性能であるが, 単純な実行時間では hyperlmn が lmn-mem に優っている.

4.4 RAM simulator

RAM simulator は, アドレスを持つメモリ領域に格納されているデータを, 用意された命令に従って操作する例題である. 今回選択した例題の中では最もルール数が多い例題であり, 構造が複雑化してしまうため, lmn-mem での記述は断念した. hyperlmn での記述例の抜粋を図 13 に示す. 例えばルール add は, 操作 add を持つ命令 prog のラベル $l とカウンタ pc が持つラベルが等しい場合に, アドレス $b のメモリ mem に格納されている値 Y をアドレス $a に格納されている値 X へ加算し, pc のラベルを N へと変更している. メモリの数を 30 とし, カウンタが持つラベルの数 N を変化させて実行時間 の測定を行った. 実験結果を図 14 に示す. 今回の測定ではマッチング時の非決定性に影響するメモリの数が 30 と固定であるため, lmn-int であっても時間計算量の悪化は見られない. しかし一定の非決定性は存在するために, 実行速度は hyperlmn が最も速い.

```
add@@ prog($l,N,add($b),$a),mem($b,Y) \ mem($a,X),
  pc($l) :- Z=X+Y,hlink(N) | mem($a,Z),pc(N).
sub@@ prog($l,N,sub($b),$a),mem($b,Y) \
  mem($a,X),pc($l) :- Z=X-Y,hlink(N) | mem($a,Z),pc(N).
```

図 13: RAM simulator の記述例 (hyperlmn, 抜粋)

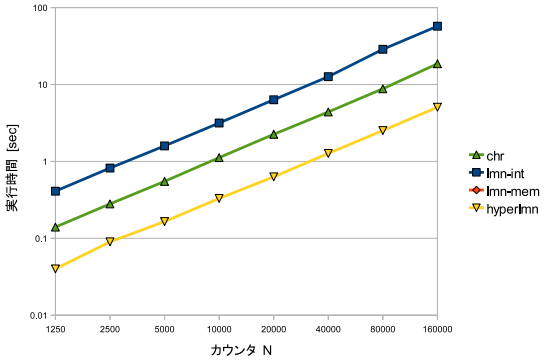

図 14: RAM シミュレータの実験結果

5. まとめと今後の課題

これまで LMNtal 上では表現困難であったデータ間の多様な参照関係をより自然に記述可能にするために, ハイパーグラフを表す新たなデータ構造であるハイパーリンクを処理系に実装した. そして CHR のベンチマーク問題を中心に, 実際にハイパーリンクを利用したプログラムを記述し, 従来よりも理想的な計算量でプログラムの実行が可能になったこと, 例題によっては CHR よりも高速に動作する例題があることなどを確認した. 今後の課題としては, より多くの例題のエンコード, SLIM が持つモデル検査機能への対応, ハイパーリンクへのデータの束縛の検討などが挙げられる.

謝辞 早稲田大学上田研究室を卒業した岡部亮氏, 佐々木隆之氏, 堀泰祐氏, 石川力氏, 後町将人氏には, 処理系の開発, ハイパーリンクの導入に協力していただいた. また同言語班の綾野貴之氏, 岩澤宏希氏, 中川遼平氏, 清水涼子氏らとの活発な議論も, 上記の開発を行う上で非常に有益なものであった. ここに感謝の意を表する. 本研究の一部は, 科学研究費補助金 (特定 18049015) の補助を得て行った.

参考文献

[1] Ueda, K.: LMNtal as a Hierarchical Logic Programming Lnaguage, Theoretical Computer Science, Vol. 410, No. 46, pp. 4784–4800, 2009.

[2] 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.

[3] 綾野貴之, 堀泰祐, 岩澤宏希, 小川誠司, 上田和紀: 統合開発環境による LMNtal モデル検査, コンピュータソフトウェア, Vol. 27, No. 4 (2010), pp. 197–214.

[4] Frühwirth, T.: Theory and Practice of Constraint Handling Rules, Journal of Logic Programming, Vol. 37, pp. 95–138, 1998.

[5] 小川誠司, 綾野貴之, 上田和紀: LMNtal を用いた状態空間探索, 第 23 回人工知能学会全国大会, 2H2–3, 2009.

[6] Frühwirth, T.: Constraint Handling Rules, Cambridge University Press, Cambrige, UK, 2009.

[7] Galil, Z. and Italiano, G. F.: Data Structures and Algorithms for Disjoint Set Union Problems, Computing Surveys, Vol. 23, N0. 3, pp. 319–344, 1991.