グラフ書換え系における静的グラフ型検査

吉元 佑介　上田 和紀

グラフ書換え系とは、グラフ構造を、部分グラフの書換え規則によって書き換えることによって計算を行う系である。グラフを書き換えの対象とすることで、複雑なポインタ構造をもったデータ構造の操作を自然に表現できる。だが、データ構造の操作を書換え規則として表したとき、その規則による書き換えが、データ構造の持つ不変条件を崩さないことを示すのは容易ではない。そこで、本研究では、グラフ書換え系に静的型体系を導入し、書換え規則がデータ構造を保存することを検証する手法を提案する。この静的型によって、複数のルートを持つグラフ、循環や共有を表現するグラフ、数値制約を満たすグラフが表現できることを確認した。また、多くの型について、変換ルールがデータ構造を保存するか検査できることを確認した。

Graph rewriting systems are computation models that proceed by applying subgraph rewriting rules to graphs. Since they rewrite graph structures directly, they enable us to express operations on data structures with complicated pointer linkings naturally. However, when we express operations on data structures by graph rewriting rules, it is hard to verify that the rules preserve invariants of the data structures. In this research, we introduced a static type system for graph rewriting systems and proposed a method to verify preservation of invariants for rewriting rules. We confirmed that the type system can express graph structures with multiple roots, loops, data sharings and numerical constraints. Furthermore, we confirmed that the proposed method can verify the preservation of invariants for rules for a large class of types.

1　はじめに

グラフ書換え系とは、グラフ構造を、部分グラフの書換え規則によって書き換える計算モデルである。グラフの書換え規則は、「左辺のグラフを右辺のグラフに書き換える」という形式で記述する。現在のグラフが、書換え規則の左辺を部分グラフを部分グラフとして含む場合、書換え規則が適用され、マッチした部分グラフが書き換え規則右辺のグラフに置換される。その応用例としては、オブジェクト指向言語の検証ツールである GROOVE[5] や、グラフ書換えの実行系およびモデル検査ツールである SLIM[9][7] があげられる。

グラフ書換え系は、グラフを直接操作する計算モデルであるため、入り組んだポインタ構造の操作を自然に記述できるという点で優れている。ここで、グラフ構造の持つ不変条件が、書換え規則による書換えを行っても保存されることが検証できると便利である。そこで、グラフ書換え系に静的型体系を導入し、不変条件の保存をグラフの型の保存という形で表し、それを検証する手法を提案する。

Static Graph Type Checking for Graph Rewriting Systems
Yusuke Yoshimoto, 早稲田大学基幹理工学研究科情報理工・情報通信専攻, Waseda University, Graduate School of Fundamental Science and Engineering, Department of Computer Science and Communications Engineering.

2　言語モデル LMNtal

本論文では、グラフ書換え系のひとつである LMNtal [10] に型体系を導入する。そのため、まずは LMNtal の構文と意味論を定義する。なお、ここで定義するのは、LMNtal のサブセットである Flat LMNtal に、ルールの生成を許さないという制限を加えた体系である。本論文では断りのない限り、LMNtal はここで定義する体系を指す。

2.1　構文

LMNtal の構文は、グラフとルール（書換え規則）から構成される。まず、LMNtal グラフの構文を以下

に示す。

$$
\begin{array}{rcll}
G & ::= & 0 & \text{(空)} \\
 & | & p(L_1, ..., L_n) & \text{(アトム)} \\
 & | & G, G & \text{(分子)}
\end{array}
$$

直観的には、空は、何もないことを表すグラフである。アトム $p(L_1, ..., L_n)$ は、p という名前のノードに、$L_1, ..., L_n$ という n 個の無向エッジ（LMNtal ではリンクと呼ぶ）が接続されたものを表す。なお、アトムの種類は、その名前 p と、接続すべきリンク数 n の組で表せる。この組をファンクタと呼び、p/n と表記する。分子は、二つのグラフを合わせたグラフである。

同名のリンクが 2 回出現した場合、エッジの接続を表す。グラフに対し、1 回しか出現しないリンクを自由リンク、ちょうど 2 回出現するリンクを局所リンクという。

G に含まれる自由リンクを明示したいときは、$L_1, ..., L_n$ がすべての自由リンクであるとき、$G[L_1, ..., L_n]$ と表記する。

次に、LMNtal ルールの構文を以下に示す。

$$R ::= G \text{ :- } G$$

直観的には、LHS :- RHS によって、グラフ LHS が RHS に書換え可能であるという規則を表す。

2.2 意味論

構文上異なる LMNtal グラフが、意味的に同じグラフを表すことがある。このことを表す、LMNtal グラフの同値関係を、図 1 を満たす最小の同値関係として定義する。

特徴的な点は、局所リンクの名前は意味を持たないことと、二つのリンクを持つイコールアトムは、リンクを接続するという意味を持った特別なアトムであるということである。

LMNtal グラフ G とルール $R := LHS$:- RHS が与えられたとき、G が G' に書換え可能であるという関係を $G \rightarrow_R G'$ で表す。直観的には、G に LHS と合同な部分グラフが含まれるとき、その部分グラフを RHS に置換した構造に書き換え可能である。これは、図 2 を満たす最小の関係として定義される。 なお、二つのルール LHS :- RHS と LHS' :- RHS' が合同であるとは、これらをグラフ :- (LHS, RHS) および :- (LHS', RHS') として見たときに合同であるという意味である。

$$0, G \equiv G$$

$$G_1, G_2 \equiv G_2, G_1$$

$$G_1, (G_2, G_3) \equiv (G_1, G_2), G_3$$

$$G \equiv G[L/M]$$

ただし、L は G の局所リンクで、M は新たなリンク名

$$\frac{G_1 \equiv G_1'}{G_1, G_2 \equiv G_1', G_2}$$

$$=(L, L) \equiv 0$$

$$=(L_1, L_2) \equiv =(L_2, L_1)$$

$$=(L, L'), p(L_1, ..., L, ..., L_n) \equiv p(L_1, ..., L', ..., L_n)$$

ただし，L は $p(L_1, ..., L, ..., L_n)$ の自由リンク

図 1　LMNtal グラフの合同関係

$$\frac{G_1 \rightarrow_R G_1'}{G_1, G_2 \rightarrow_R G_1', G_2}$$

$$\frac{R \equiv R' \quad G \rightarrow_{R'} G'}{G \rightarrow_R G'}$$

$$LHS \rightarrow_R RHS$$

図 2　LMNtal グラフのルールによる遷移関係

（$R = LHS$:- RHS）

意味である。

2.3 略記法

グラフの表記を簡略化するための略記法を導入する。適当なリンク名 L_{1i} を用いて、

$$p(L_{11}, ..., L_{1i}, ..., L_{1n}), q(L_{21}, ..., L_{2m}, L_{1i})$$

と表されるグラフは、

$$p(L_{11}, ..., q(L_{21}, ..., L_{2m}), ..., L_{1n})$$

と表記してよい。

また、ルールの表記を簡略化するための略記法を導入する。自由リンク $L_1, ..., L_n$ をもつ任意の連結なグラフを、適当な名前 p を用いて、$\$p(L_1, ..., L_n)$ と表記

してよい。これは構文としてはアトムに属し、この形のアトムを含むルールは、$p(L_1, ..., L_n)$ が表す全てのグラフに対するルールを意味する。

3 LMNtal グラフ型

LMNtal グラフに、Shape Type [2] をベースにした型体系を導入する。型に属すグラフの集合を、生成文法によって定義する。

3.1 構文

LMNtal グラフ型の定義構文を以下に示す。

```
defshape t(L1, L2, ..., Ln) {
    Rules
} nonterminal {
    Functors
}
```

t は型名である。型は n 個の自由リンク（順序有り）をもち、リンク名 $L_1 \sim L_n$ で表す。 Rules は生成規則の集合である。 Functors は非終端記号の集合である。なお、nonterminal 以降は省略可能で、省略した場合、非終端記号は t/n のみとなる。

3.2 意味論

型 $t(L_1, L_2, ..., L_n)$ に属すグラフの集合は、直観的には、以下の条件をみたすものである。

1. 文法の開始記号 $t(L_1, L_2, ..., L_n)$ に、生成規則を 0 回以上適用して生成されるグラフである
2. グラフに含まれる全てのアトムのファンクタは、非終端記号である

グラフ $G[L_1, ..., L_n]$ が 1 を満たすことを、$G[L_1, ..., L_n] \lhd t(L_1, ..., L_n)$ と表記する。これは、図 3 を満たす最小の関係として定義される。

ただし、 p は $t(L_1, ..., L_n)$ 型の推論規則に含まれる任意のルールである。

なお、推論規則において、遷移関係の表記は冗長であるので、以降は

$$\frac{G'[L_1, ..., L_n] \lhd t(L_1, ..., L_n)}{G[L_1, ..., L_n] \lhd t(L_1, ..., L_n)} \ (p)$$

と表記する。

$G[L_1, ..., L_n] \lhd t(L_1, ..., L_n)$ を 、「 $G[L_1, ..., L_n]$ は $t(L_1, ..., L_n)$ によって生成される」と表現する。また、$G[L_1, ..., L_n] \lhd t(L_1, ..., L_n)$ であり、かつ、2 を満たすことを、$G[L_1, ..., L_n] : t(L_1, ..., L_n)$ と表記し、「 $G[L_1, ..., L_n]$ は $t(L_1, ..., L_n)$ 型を持つ」と表現する。

また、LMNtal ルールのうち、型 $t(L_1, ..., L_n)$ の非終端記号のみから構成されるものを、型 $t(L_1, ..., L_n)$ の変換ルールと呼ぶ。「変換ルールが型を保存する」という関係を、以下をもって定義する。

ルール $R = LHS$:- RHS が型 $t(L_1, ..., L_n)$ を保存するとは、

$$\forall G \lhd t(L_1, ..., L_n).\ G \rightarrow_R G' \Rightarrow G' \lhd t(L_1, ..., L_n)$$

を満たすことである。

4 グラフ型の例

複数のルートを持ち、循環を含むデータ構造として、スキップリスト [4] をグラフ型で表現する。レベル 2 のスキップリストは、以下のグラフ型で表せる。

```
defshape skiplist(List1, List2) {
  (Prod0) skiplist(L1, L2) :- nil(L1, L2).
  (Prod1) nil(L1, L2) :- node1(X1, L1),
      nil(X1, L2).
  (Prod2) nil(L1, L2) :- node2(X1, X2, L1, L2),
      nil(X1, X2).
}
```

これは、リンクリストの一部のノードに、途中のノードを飛ばすポインタを付与したものである。ソートされたデータをスキップリストを用いて実装すると、途中のノードを飛ばすポインタを利用した、効率的な探索が可能となる。なお、ここではスキップリストの構造だけに着目しているため、各ノードが持つデータは無視している。

skiplist(List1, List2) 型をもつグラフの例を図 4 に示す。

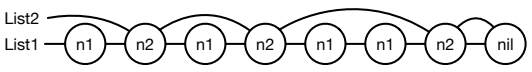

図 4　スキップリストの例

$$t(L_1, ..., L_n) \lhd t(L_1, ..., L_n)$$

$$\frac{G'[L_1, ..., L_n] \lhd t(L_1, ..., L_n) \quad G'[L_1, ..., L_n] \rightarrow_p G[L_1, ..., L_n]}{G[L_1, ..., L_n] \lhd t(L_1, ..., L_n)} \ (p)$$

図 3　グラフ型によるグラフの生成の定義

また、数値制約を含むデータ構造として、赤黒木 [1] をグラフ型で表現する。黒高さ 3 の赤黒木は、以下のグラフ型で表せる。

```
defshape rbtree(Tree) {
    (Prod0) rbtree(X) :- bnode(s(s(s(z))), X).
    (Prod1) bnode(z, X) :- l(X).
    (Prod2) bnode(s($m), X) :-
        b(node($m), node($m), X).
    (Prod3) node(z, X) :- l(X).
    (Prod4) node(s($m), X) :- b(node($m), node($m), X).
    (Prod5) node($m, X) :- r(bnode($m), bnode($m), X).
}
```

これは、以下の制約を満たすような二分木である。

1. 根と葉は黒に色づけられている
2. 赤に色づけられたノードの子は、黒に色づけられたノードである。
3. 任意のノードについて、葉に至るまでの全ての経路上に、同数の黒に色づけられたノードがある

3 の数値制約は、各ノード上に、そのノードから葉に至るまでの経路上の黒のノードの数を、0 と successor で表した自然数で持たせておくことで実現している。なお、スキップリストと同様、各ノードが持つデータは無視している。

rbtree(Tree) 型をもつグラフの例を、図 5 に示す。

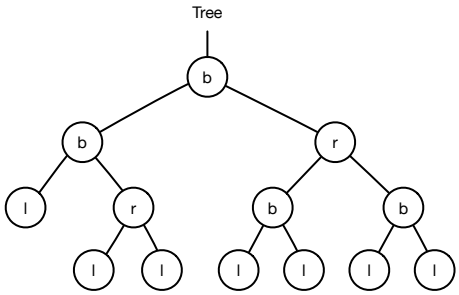

図 5　赤黒木の例

5　グラフ型検査

与えられたグラフ $G[L_1, ..., L_n]$ に関して、$G[L_1, ..., L_n] : t(L_1, ..., L_n)$ であるか検査する方法を示す。LMNtal グラフの定義から、グラフに含まれるアトムの数は有限なので、$G[L_1, ..., L_n]$ が非終端記号を含まないことは、全てのアトムを見ることで判定できる。$G[L_1, ..., L_n] \lhd t(L_1, ..., L_n)$ であるかどうかは、生成規則を逆向きにしたルールを用いて、$G[L_1, ..., L_n]$ が $t(L_1, ..., L_n)$ に遷移するかどうかを見ることで判定できる。例えば、 skiplist(List1, List2) 型を持つグラフ

```
node1(X11, List1),
node2(X12, X21, X11, List2),
nil(X12, X21)
```

の型検査は、図 6 に示す証明図を、下から上へ向かって構成することで行える。

6　ルール型検査

文脈自由な型、すなわち、全ての生成規則の左辺がひとつのアトムだけからなるような型については、与えられた変換ルールの型保存性が検査できる。検査は以下の 2 ステップで構成される。

1. 検査対象ルール左辺を含む t 型のグラフを生成する、証明図のパターンを全て列挙する
2. 全ての証明図パターンに対し、検査対象ルールの左辺グラフを右辺グラフに置換したグラフが t 型であることの証明が、パターンを元に構成できるか調べる

以降では、リンクの集合を $\boldsymbol{L}$ というように太字で表記する。

6.1　ステップ 1 : 証明図パターンの列挙

ステップ 1 を実現するには、変換ルール左辺に、生成規則を逆向きにしたルールを適用し、開始記号まで

$$\dfrac{\dfrac{\dfrac{\text{skiplist(List1, List2)} \lhd \text{skiplist(List1, List2)}}{\text{nil(List1, List2)} \lhd \text{skiplist(List1, List2)}}\,(\text{Prod}_0)}{\text{node1(X11, List1), nil(X11, List2)} \lhd \text{skiplist(List1, List2)}}\,(\text{Prod}_1)}{\text{node1(X11, List1), node2(X12, X21, X11, List2), nil(X12, X21)} \lhd \text{skiplist(List1, List2)}}\,(\text{Prod}_2)$$

図 6　スキップリストのグラフ型検査

遷移させればよい。ただし、変換ルール左辺以外のグラフが未知であるので、逆向きルール中の一部（ひとつ以上）のアトムと反応することを許す。

つまり、ルール左辺 $LHS[\boldsymbol{L_1}]$ に対し、証明図の最下部

$$G_n[\boldsymbol{L_n} \cup \boldsymbol{L}], LHS[\boldsymbol{L_n}] \lhd t(\boldsymbol{L})$$

からスタートし、

$$G_1[\boldsymbol{L_1} \cup \boldsymbol{L}], t(\boldsymbol{L_1}) \lhd t(\boldsymbol{L})$$

に到達するまで、以下を繰り返す。

1. 外部グラフ $G_i[\boldsymbol{L_i} \cup \boldsymbol{L}]$ のみを、生成規則によって 0 回以上還元する
2. 変換ルールの左辺グラフの一部である $LHS_i[\boldsymbol{L_i}]$ の部分グラフ $LHS'_i[\boldsymbol{L'_i}]$（空であってはならない）と、外部グラフの部分グラフ $G'_{i-1}[\boldsymbol{L'_{i-1}}]$（空でもよい）からなる分子を、生成規則によって還元する。ここで、 $LHS_i[\boldsymbol{L_i}]$ の残りを $LHS''_i[\boldsymbol{L''_i}]$ 、還元によってできるグラフ（これは一つのアトムのみから生る）を $G''_{i-1}[\boldsymbol{L'_i} \triangle \boldsymbol{L'_{i-1}}]$ とする。

これらの操作によって、部分的な証明図

$$\dfrac{\dfrac{G_{i-1}[\boldsymbol{L_i} \cup \boldsymbol{L} \setminus \boldsymbol{L'_{i-1}}], G''_{i-1}[\boldsymbol{L'_i} \triangle \boldsymbol{L'_{i-1}}], LHS''_i[\boldsymbol{L''_i}] \lhd t(\boldsymbol{L})}{G_{i-1}[\boldsymbol{L_i} \cup \boldsymbol{L} \setminus \boldsymbol{L'_{i-1}}], G'_{i-1}[\boldsymbol{L'_{i-1}}], LHS_i[\boldsymbol{L_i}] \lhd t(\boldsymbol{L})}\,\boxed{\text{Subproof}}}{G_i[\boldsymbol{L_i} \cup \boldsymbol{L}], LHS_i[\boldsymbol{L_i}] \lhd t(\boldsymbol{L})}$$

を得る。なお、2 の還元において、複数通りの還元が可能であったり、還元が不可能である場合がある。また、還元の結果得られた明示的なグラフが、すでに得られた明示的なグラフと同型である場合、その還元は考えない。

これを繰り返して証明図を構成した後、最後に

$$\dfrac{\dfrac{t(\boldsymbol{L'}) \lhd t(\boldsymbol{L})}{\boxed{\text{Subproof}}}}{G_1[\boldsymbol{L_1} \cup \boldsymbol{L}], t(\boldsymbol{L_1}) \lhd t(\boldsymbol{L})}$$

を加えて、ルール左辺を含むグラフの生成に関する証明図を得ることができる。

例えば、レベル 2 のスキップリストの変換ルール

```
node2(X1,Y1,X,Y), node2(X2,Y2,X1,Y1)
:- node1(X1,X), node2(X2,Y,X1,Y2)
```

の場合、ステップ 1 によって、図 7 に示す証明図のみが得られる。変換ルールの左辺グラフ node1(X1,X), node2(X2,Y,X1,Y2) を含むスキップリストは、全てこのパターンで生成することができる。

6.2　ステップ 2 : 証明図パターンの変形

ステップ 1 で得られた証明図の、一番下に現れる外部グラフ $G_n[\boldsymbol{L_n} \cup \boldsymbol{L}]$ は、そのパターンの証明図で生成される全ての外部グラフを代表している。従って、すべての証明図パターンの各々に対して、それを元に、

$$G_n[\boldsymbol{L_n} \cup \boldsymbol{L}], RHS[\boldsymbol{L_n}] \lhd t(\boldsymbol{L})$$

の証明図を構成できれば、任意の $t(\boldsymbol{L})$ 型のグラフについて、ルール LHS :- RHS は型を保存する事が分かる。

ここで、このことは型保存のための十分条件であるが、必要条件ではないことに注意する。たとえば、二つの証明図パターンが得られ、証明図最下部に現れる外部グラフが、それぞれ G, G' だったとする。もし、G として考えられるグラフの集合と、 G' として考えられるグラフの集合が一致するなら、二つのうち一つの証明図パターンに対して、書換え後のグラフが t 型である証明が得られれば十分である。必ずしも、両方に対して証明が得られる必要はない。

では、左辺グラフを含むグラフの証明図パターンを元に、書き換え後のグラフの証明図を構成する方法を示す。まず、$i \geq 1$ なる各々の i に対し、

$$\dfrac{G_i[\boldsymbol{L_{i+1}} \cup \boldsymbol{L} \setminus \boldsymbol{L'_i}], G'_i[\boldsymbol{L'_i}], LHS_{i+1}[\boldsymbol{L_{i+1}}] \lhd t(\boldsymbol{L})}{\boxed{\text{Subproof}_n}}$$
$$\dfrac{}{G_{i+1}[\boldsymbol{L_{i+1}} + \boldsymbol{L}], LHS_{i+1}[\boldsymbol{L_{i+1}}] \lhd t(\boldsymbol{L})}$$

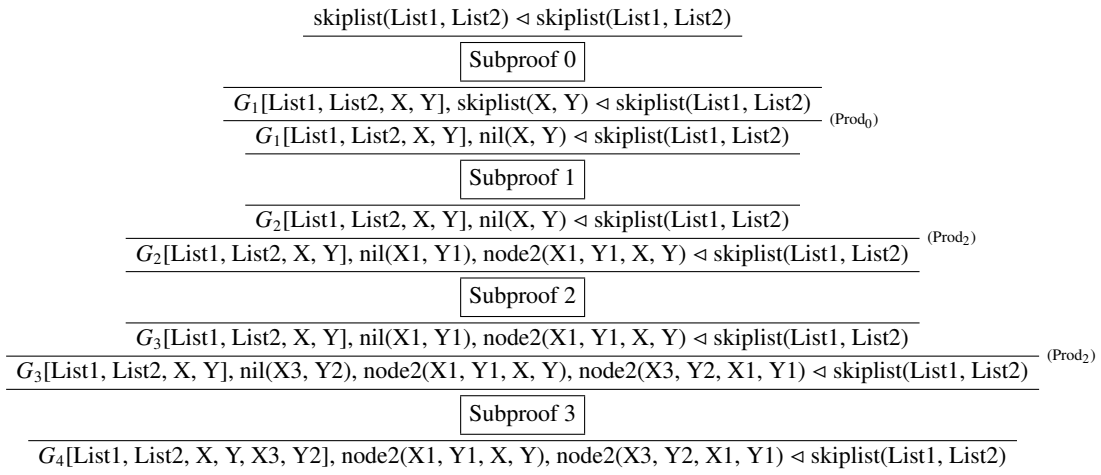

図 7　スキップリスト変換ルールの左辺グラフの生成

の形の部分的な証明図が得られているとする。

右辺グラフ $RHS_n[\boldsymbol{L_n}]$ と、グラフ $G'_1[\boldsymbol{L'_1}], ..., G'_{n-1}[\boldsymbol{L'_{n-1}}]$ からなる分子を考える。ただし、自由リンクは適切に接続されているものとする。 この分子をグラフ型検査にかける。ただし、型の持つ自由リンクの接続方法は問わない。

型検査に成功し、グラフがアトム t まで還元されると、グラフの自由リンクと型の自由リンクの対応関係がわかるので、それに従って、証明図中のグラフの自由リンク名を書き換える。

型検査に失敗した場合は、ルール型検査は失敗したとみなす。

スキップリストの例だと、右辺グラフ node1(X1, X), node2(X2, Y, X1, Y2) に付加すべきグラフは、 nil(X2, Y2) である。これらからなる分子を型検査にかけると、図 8 に示す証明図を得る。

ここでできた右辺グラフの証明図と、左辺グラフの証明図の Subproof を適切に組み合わせることで、$G_n[\boldsymbol{L_n} \cup \boldsymbol{L}], RHS[\boldsymbol{L_n}] \lhd t(\boldsymbol{L})$ の証明図を構成できる。

スキップリストの例だと、図 9 に示す証明図を得ることができる。

したがって、グラフ $G'_1[\boldsymbol{L'_1}], ..., G'_{n-1}[\boldsymbol{L'_{n-1}}]$ を、型の持つ自由リンクの接続方法を問わず型検査して、検査が成功すれば、変換ルールは型を保存することが分かる。

7　型検査の実装

グラフやルールに対する型検査は、既存の LMNtal 処理系を用いた実装が可能である。グラフの型検査を行うには、生成規則を逆向きにしたルールを、非決定的にグラフに適用すればよい。これはすでに実装ができている [8]。ルールに対する型検査を行うには、以下の操作が必要である。

1. 変換ルール左辺のグラフを、逆向きの生成規則との部分的なマッチにより書き換る。このとき、外部グラフを記録しておく。
2. 変換ルール右辺のグラフと記録した外部グラフを、逆向きの生成規則によって書き換える。

1 の部分的なマッチは、逆向きの生成規則から、可能な部分的マッチを表すルール群を生成して、それらを非決定的に適用することによって実現できている [8]。外部グラフの記録の際には、外部グラフと変換ルール左辺のグラフとの自由リンクを適切に対応させる必要がある。これは今後実装を行う予定である。

8　関連研究

循環構造を含むようなグラフに対する型付けは、Shape Type や Graph Type [3] といった先例がある。これらの研究は、グラフ書換え系でない言語の静的検証のためにグラフ型を用いており、本研究では、グラフ書換え系自体に型をつけているという違いがある。

$$\dfrac{\dfrac{\dfrac{\text{skiplist(List1, List2)} \lhd \text{skiplist(List1, List2)}}{\text{nil(List1, List2)} \lhd \text{skiplist(List1, List2)}}\,(\text{Prod}_0)}{\text{node1(X1, List1), nil(X1, List2)} \lhd \text{skiplist(List1, List2)}}\,(\text{Prod}_1)}{\text{node1(X1,List1), node2(X2, List2, X1, Y2), nil(X2, Y2)} \lhd \text{skiplist(List1, List2)}}\,(\text{Prod}_2)$$

図 8　スキップリストのルールの右辺グラフ型検査

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{skiplist(List1, List2)} \lhd \text{skiplist(List1, List2)}}{\boxed{\text{Subproof 0}}}}{G_1\text{[List1, List2, X, Y], skiplist(X, Y)} \lhd \text{skiplist(List1, List2)}}}{G_1\text{[List1, List2, X, Y], nil(X, Y)} \lhd \text{skiplist(List1, List2)}}\,(\text{Prod}_0)}{\boxed{\text{Subproof 1}}}}{G_2\text{[List1, List2, X, Y], nil(X, Y)} \lhd \text{skiplist(List1, List2)}}}{G_2\text{[List1, List2, X, Y], node1(X1, X), nil(X1, Y)} \lhd \text{skiplist(List1, List2)}}\,(\text{Prod}_1)}{\boxed{\text{Subproof 2}}}}{G_2\text{[List1, List2, X, Y], node1(X1, X) nil(X1, Y)} \lhd \text{skiplist(List1, List2)}}}{G_2\text{[List1, List2, X, Y], node1(X1, X), node2(X2, Y2, X, Y), nil(X2, Y2)} \lhd \text{skiplist(List1, List2)}}\,(\text{Prod}_2)}{\boxed{\text{Subproof 3}}}}{G_3\text{[List1, List2, X, Y, X2, Y2], node1(X1, X), node2(X2, Y2, X1, Y)} \lhd \text{skiplist(List1, List2)}}$$

図 9　スキップリスト変換ルールの右辺グラフの生成

また、数値制約を含むデータ構造への型付けは、Liquid Haskell [6] といった先例がある。これは関数型言語に対して数値制約を導入するものであり、一般的なグラフに対する制約を記述することはできない。

9　まとめと今後の課題

本論文では、グラフ書換え系のひとつである LMNtal に型体系を導入し、グラフの型検査と、変換ルールの型検査を行う方法を示した。

今後の課題として、検査可能なルールをより一般的なものにする、ということが挙げられる。本論文で示したルール型検査は、文脈自由な型に対してしか適用できない。しかし、グラフ型の例として示したように、文脈自由でない型を利用することで、制約の概念が表現できるようになる。このような型の変換ルールに対する型検査は重要である。

謝辞 LMNtal 処理系の開発に携わった早稲田大学上田研究室の先輩方、本発表に関する議論に快く応じてくれた現役生に感謝申し上げる。本研究の一部は、科学研究費補助金（基盤研究 (B) 26280024）の補助を得て行った。

参考文献

[1] Bayer, R.: Symmetric Binary B-Trees: Data Structure and Maintenance Algorithms, *Acta Informatica*, Vol. 1, No. 4, 1972, pp. 290–306.

[2] Fradet, P. and Métayer, D. L.: Shape Types, *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 1997, pp. 27–39.

[3] Klarlund, N. and Schwartzbach, M. I.: Graph Types, *Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 1993, pp. 196–205.

[4] Pugh, W.: Skip lists: a probabilistic alternative to balanced trees, *Communications of the ACM*, Vol. 3, No. 6(1990), pp. 668–676.

[5] Rensink, A.: The GROOVE simulator: A tool for state space generation, *Applications of Graph Transformations with Industrial Relevance (AGTIVE), LNCS 3062*, 2004, pp. 479–485.

[6] Vazou, N., Seidel, E. L., and Jhala, R.: LiquidHaskell: experience with refinement types in the real world, *Proceedings of the 2014 ACM SIGPLAN symposium on Haskell*, 2014, pp. 39–51.

[7] 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.

[8] 吉元佑介: グラフ書換え言語 LMNtal への Shape Type の導入と実装, 早稲田大学卒業論文, 2014.
[9] 石川力, 堀泰祐, 岡部亮, 村山敬, 上田和紀: 軽量な LMNtal 実行時処理系 SLIM の設計と実装, 情報処理学会全国大会講演論文集 第 *70* 回, 2008, pp. 153–154.
[10] 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, Vol. 21, No. 2(2004), pp. 126–142.