グラフ書換え言語におけるグラフ操作の軽量かつ静的な型検査

山本 直輝　上田 和紀

グラフ書換え言語 LMNtal は，グラフの書換えによって計算を表現するプログラミング言語としての側面と，一般の複雑なグラフ構造を扱えるモデリング言語としての側面とを兼ね備えた言語である．LMNtal プログラムは構文レベルでポインタ安全であることが保証されているが，静的に型エラーを報告する機能がないため，プログラマの意図しない振る舞いを起こす危険性がある．グラフ書換え言語では，関数型言語で扱える代数的データ構造よりも広範な構造を扱えるため，計算対象となるグラフの型の定式化は自明でない．特に，部分グラフから部分グラフへの書換えとして記述されるグラフの操作については，軽量な型安全性の検証手法が課題となっていた．そこで本論文では，グラフ書換え言語を対象とする静的かつ軽量な型検査手法として，型の差分情報に基づく手法を新たに提案する．

1　背景・目的

1.1　グラフ書換え言語 LMNtal

代数的データ型（リストや木構造）よりも複雑な接続構造をもつ**グラフ**を簡潔かつ安全に扱うために，第一級オブジェクトとしてのグラフを書き換えることによって計算を表現する，**グラフ書換え言語**というパラダイムが提案されている．中でも LMNtal[7] は，アトム（ノード）やルール（書換え規則）といった最小限の言語要素からなる言語モデルである．

LMNtal はプログラミング言語としての側面の他に，モデリング言語としての側面を持ち合わせている．これと対応して，LMNtal の実行時処理系である SLIM[11] には，実行時処理系とモデル検査器の 2 種類の実行モードがある．特にモデル検査器としての SLIM にはグラフの書換えによって遷移しうる全状態を出力する機能があるほか，LTL モデル検査等の機能も充実している．

LMNtal グラフは well-formed であるかぎり，不正なポインタを含まない．この意味で，LMNtal は言語仕様のレベルで不正なポインタを排除している（ポインタ安全である）といえる．一方で，LMNtal には静的型の概念がないため，書換えの結果がプログラマの意図しないものとなることがある．

例えば，スキップリスト[6] は線形リストに途中の幾つかのノードを通過するようなエッジを追加したデータ構造である（図 1 (a)）．これは LMNtal グラフとして図 1 (b) のように表現できる †1．

ここで，スキップリストの `n2` ノードが 2 つ並んだところがあったら，右側を `n1` に書き換える，という書換え規則を考え，次の 2 通りを書いたとする．

```
n2(X,A,B,C,D), n2(Y,E,F,B,A)
  :- n2(X,A,F,C,D), n1(Y,E,A).
```

```
n2(X,A,B,C,D), n2(Y,E,F,B,A)
  :- n2(X,A,E,C,D), n1(Y,F,A).
```

---

* Lightweight Static Type Checking of Graph Manipulation in Graph Rewriting Languages.
This is an unrefereed paper. Copyrights belong to the Author(s).

Naoki Yamamoto, Kazunori Ueda, 早稲田大学基幹理工学研究科情報理工・情報通信専攻, Dept. of Computer Science and Communications Engineering, Graduate School of Fundamental Science and Engineering, Waseda University.

†1　図中でノードの引数（ノードから伸びるエッジ）に付された矢印は第 1 引数の場所を指し，そこから矢印の向きに従って引数に順序がついていることを表す．

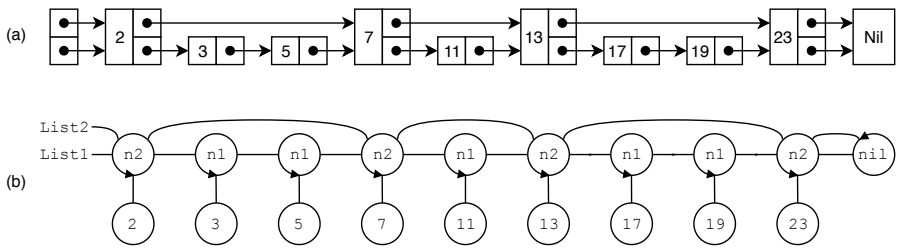

図 1　スキップリスト： (a) グラフによる表現 および (b) LMNtal グラフによる表現

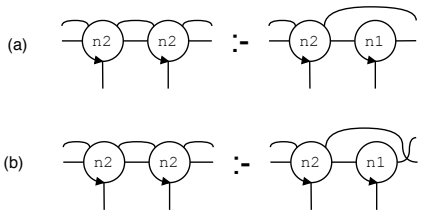

図 2　スキップリストの書換え規則：
(a) 正しいもの および (b) 誤ったもの

このように，特にテキスト表現においては，どちらが正しい書換え規則であるか，ひと目では判断し難い．なお，これらを図に直すと，前者は図 2 (a)，後者は図 2 (b) にそれぞれ示す通りの書換え規則になっている．後者は右側のエッジの繋がり方が変わってしまっているので，このルールを適用するとスキップリストの形状が破壊されてしまうことになる．

1.2　LMNtal を対象とする静的検査手法

前述の課題を解決するため，これまでにも LMNtal を対象とする幾つかの静的検査手法が提案されている．特に，Prolog 系言語におけるモード解析に由来する [8] の手法は，超辺によるノード間の接続について，入出力の方向性を解析し，所有権の分割を表現するため $[-1, 1]$ の実数値を割り当てて整合性を検査するものである．また，LMNtal ShapeType [10] は，Structured Gamma [3] およびそのサブセットの Shape types [2] を基にして，グラフの型を生成文法で定義し，ルールの型安全性を検査する手法である．

しかしながら，[8] の手法では入出力の方向性のみに着目しているため，そこを流れるデータの種類については検証の対象外である．また，LMNtal ShapeType ではルールによる書換えの前後においてグラフ全体の形状が破壊されないことまでを検査することができるが，マッチしうるグラフの全パターンについて網羅的に検証を行うため，生成規則およびルール左辺の形によっては膨大な数の場合分けが発生し，最悪の場合 1 つのルールの検証に数十分以上を要する場合があった．加えて，「書換え前のグラフが τ 型であったならば，書換え後のグラフも τ 型である」という性質（型保存性）を検査するにあたって，書換え前のグラフの型 τ を前提条件としてユーザの側でルールごとに明示しておかなくてはならないという煩雑さがあった．

1.3　本論文の貢献と構成

そこで本論文では，グラフ書換え言語を対象とする静的かつ軽量な型検査手法として，型の差分情報に基づく LMNtype を新たに提案する．LMNtype では，[8] の手法と同様に局所的な接続関係に着目して型検査を行うが，入出力の方向性に加えてそこを流れるデータの種類にも着目している．例えば，リストコンストラクタ (cons) の第 1 引数に int 型が入力され，第 2 引数にリスト型が入力されると，第 3 引数からリスト型が出力される，というように，各ノードについてそこから伸びるエッジごとに型制約を付す．なお，LMNtype における入出力の概念については上記の例をもとに 3 節の冒頭でさらに詳しく見ていく．

我々が従来研究してきた LMNtal ShapeType と比較すると，LMNtype の型検査の方がはるかに軽量であるが，LMNtype の型付けはあくまで局所的な接続関係に立脚するため，グラフ全体の形状がどのようであるかまでは素朴には検証することができない．例えば，LMNtype ではリストが通常のように線形な形状

プロセス

$P ::= \mathbf{0}$ (空)
| $p(\vec{X})$ (アトム)
| P, P (分子)
| $[\,RuleName@@\,]\ T$:- T (ルール)

プロセステンプレート

$T ::= \mathbf{0}$ (空)
| $p(\vec{X})$ (アトム)
| T, T (分子)

図 3　LMNtal の構文

をしている場合と，先頭と末尾とが接続されて環状となっている場合とを区別することができない（4.1 節参照）．これは，関数型など通常の言語では扱うことが難しい広範なグラフ構造を許容するため，敢えてそのように設計しているのだが，線形なリストのみを許すために拡張した型検査手法についても 5.4 節で議論する．

本論文の構成は以下の通りである．まず，2 節で型付けの対象言語であるグラフ書換え言語 LMNtal について簡単に紹介し，3 節で今回提案する型検査手法 LMNtype を導入する．4 節で型検査の具体的な例題を紹介し，5 節では LMNtype の性質や拡張の可能性等について論ずる．

2　LMNtal

本節では，文献[7] の内容をもとにグラフ書換え言語 LMNtal について簡単に説明する．なお，本来 LMNtal には膜による階層や，ルールの発火に関する制約条件を記述するためのガード及びプロセス文脈といった機能が備わっているが，本論文における型付けの対象言語としては，簡単のためこれらを除外したサブセット言語を扱う．

以下，構文要素 X の列 $X_1, \ldots, X_n$ $(n \geq 0)$ を $\vec{X}^n$ と略記し，n が重要でない場合は単に $\vec{X}$ と書く．

2.1　構文

LMNtal の構文を図 3 に示す．LMNtal における**アトム・リンク**は，それぞれグラフ理論における節点（ノード）・辺（エッジ）と対応している．p という名前で，m 本のリンク $X_1, \ldots, X_m$ をもつアトムを $p(X_1, \ldots, X_m)$ と書く．これらのリンクをアトムの引数といい，順序がついている．英字の大文字から始まる名前はリンク名，そうでないものはアトム名として解釈される．アトム名と引数の本数（価数）の組を**ファンクタ**といい，p/m のように書き，「m 価の p アトム」のように読む．プロセス P 中に出現するアトムのファンクタの集合を Funct(P) と書く．

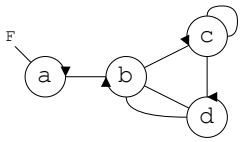

図 4　LMNtal グラフの例

LMNtal においては，アトムの多重集合によって無向多重グラフを表現する（つまり，多重辺や自己ループの出現を許す）．例えば，

```
a(A,F), b(A,C,L1,L2), c(C,D,S,S), d(D,L1,L2)
```

というアトムの多重集合は，図 4 に示すような無向グラフを意味する．

ルールは，部分グラフから部分グラフへの書換えを表す書換え規則である．例えば，`a(X) :- b(X)` は 1 価の `a` アトムを，1 価の `b` アトムに書き換えるルールである．便宜上，ルールの名前として *RuleName* をルールの前に付すこともある．

LMNtal では，アトムとルールからなる多重集合であるプログラム全体のことを**プロセス**と呼んでいる．これは，LMNtal が並行計算のモデリング手法であるプロセス計算に由来することによる．以下，ルールを含まないプロセスのことを単に**グラフ**と呼ぶ．

なお，構文の解釈に曖昧性が生じる場合，必要に応じ括弧を使用する．また便宜上，ルールの結合子 “`:-`” より優先度の低い区切り記号として，“`,`” のかわりに “`.`” を用いる．すなわち，区切り記号の優先順位は結合の強い方から順に “`,`”, “`:-`”, “`.`” となる．

2.1.1　略記法

より簡潔な記述を可能にするため，次の二つの略記法を認める．

1. アトムの引数にアトムを書いた場合は，そのアトムの最終引数に繋がっているものと解釈される．つ

まり，$p(X_1,\ldots,q(Y_1,\ldots,Y_n),\ldots,X_m)$ は，$p(X_1,\ldots,L,\ldots,X_m)$，$q(Y_1,\ldots,Y_n,L)$ と解釈される．ただし，L は使用されていない適当なリンク名とする．例えば，`a(b(c),d)` は `a(B,D),b(C,B),c(C),d(D)` を意味する．

2. 引数リストを省略した場合は引数が 0 個であると解釈される．つまり，p は $p()$ の意味である．

2.2 構文上の制約条件

以下，各構文要素が満たすべき制約条件を述べる．

定義 2.1 (グラフのリンク条件)．グラフおよびプロセステンプレートに同名のリンクは高々2 回出現できる．このうち，2 回出現するリンクは両端がアトムと繋がっている**局所リンク**であり，1 回のみ出現するリンクは一端が無接続である（または，外部と接続している）**自由リンク**である．自由リンクのないグラフは**閉じている**という．G 中の自由リンクの集合を $\mathcal{F}(G)$ と書く．2 つのグラフをカンマで結合する際は，局所リンク名が衝突しないように（後述の構造合同規則 (E4) により）必要に応じて α 変換する． ◇

定義 2.2 (ルールのリンク条件)．ルール T_1 :- T_2 は，リンク条件 $\mathcal{F}(T_1) = \mathcal{F}(T_2)$ を満たす必要がある．ルールには自由リンクは存在しないため，すべてのルールについて $\mathcal{F}(T_1 \text{ :- } T_2) = \emptyset$ とする．従って，プロセス P における自由リンク集合 $\mathcal{F}(P)$ は，P からすべてのルールを取り除いたグラフ G の自由リンク集合 $\mathcal{F}(G)$ と同一となる． ◇

2.3 意味論

LMNtal の意味論は，LMNtal プロセス上の 2 つの二項関係からなる．

2.3.1 構造合同関係

まず，LMNtal プロセス同士の構文的な差異を吸収する同値関係である構造合同関係の定義を与える．図 5 に示す構造合同規則 (E1)–(E9) を満たす最小の同値関係 “$\equiv$” を構造合同関係という．これは，グラフ理論におけるグラフ同型にあたる同値関係であり，構文上の差異を吸収するものである．なお，$P[Y/X]$ はプロセス P に出現するリンク X をリンク Y に置き換えることを意味する．ただし，(E6), (E10) は膜に関する規則であるので省略した．

(E1)	$\mathbf{0},P \equiv P$
(E2)	$P,Q \equiv Q,P$
(E3)	$P,(Q,R) \equiv (P,Q),R$
(E4)	$P \equiv P[Y/X]$
	（X は P の局所リンク）
(E5)	$P \equiv P' \Rightarrow P,Q \equiv P',Q$
(E7)	$X\text{=}X \equiv \mathbf{0}$
(E8)	$X\text{=}Y \equiv Y\text{=}X$
(E9)	$X\text{=}Y,P \equiv P[Y/X]$
	（P はアトム，X は P の自由リンク）

図 5 LMNtal の構造合同規則

(E1)–(E3), (E5) はプロセス計算にもみられる一般的な規則であり，(E4) は局所リンク名の α 変換である．(E7) から (E9) までは特別な 2 価のアトムである**コネクタ**に関する規則である．`=`(X,Y) は二つのリンク X, Y を接続するという意味を持ち，中置記法で X `=` Y とも書く．(E7) は一つのリンクの両端が繋がって輪になっているものは空とみなして良いこと，(E8) はリンクが対称である（つまり無向である）こと，(E9) は直接接続されている二つのリンクが一つのリンクに縮約できることを表している．

2.3.2 遷移関係

次に，LMNtal における本質的な計算ステップを表す二項関係である遷移関係の定義を与える．図 6 に示す遷移規則 (R1)–(R6) を満たす最小の二項関係 “$\rightarrow$” を遷移関係という．ただし，(R2), (R4), (R5) は膜に関する規則であるので省略した．

最も本質的で重要な規則は (R6) である．これは，「ルールと，その左辺にマッチするプロセスが存在したら，その左辺を右辺に書き換えてよい」ということを言っている．

3 軽量な型検査手法 LMNtype

本節では，LMNtype の形式的な定義（構文・意味論）を与えるが，まずは基本的な型であるリストを例

$$\text{(R1)}\quad \frac{P_1 \to P_1'}{P_1, P_2 \to P_1', P_2}$$

$$\text{(R3)}\quad \frac{P_2 \equiv P_1 \quad P_1 \to P_1' \quad P_1' \equiv P_2'}{P_2 \to P_2'}$$

$$\text{(R6)}\quad T, (T \texttt{ :- } U) \to U, (T \texttt{ :- } U)$$

図 6　LMNtal の遷移規則

として，直観的なアイディアから紹介していく．

LMNtype において型付けの対象となるアトムは，すべての引数について入出力の方向性と型をあらかじめ定めておく．例えば，1.3 節でも紹介したように，リストコンストラクタを表す cons/3 アトムには，第 1 引数に int 型が入力され，第 2 引数にリスト型が入力され，第 3 引数からリスト型が出力されるアトムであると定めておく．これを，LMNtype では次のように表記する．

$$\texttt{cons(X,L,R)} : \texttt{int(X)},\texttt{list(L)} \rightarrowtail \texttt{list(R)}.$$

これは，アトム cons(X,L,R) は，リンク X から int 型を，L から list 型を受け取り，R へ list 型を出力するアトムであると言明している．同様に，リスト終端を表す nil/1 アトムについては，

$$\texttt{nil(R)} : \mathbf{0} \rightarrowtail \texttt{list(R)}.$$

ここで **0** は，nil には入力がないことを表している．このように，LMNtype では型自体を定めるのではなく，コンストラクタの入出力関係を設定することによって，余帰納的に型の意味が定められる点が特徴である．

このもとで，例えばリスト $[3,1,4]$ を表す次のグラフの型を考える．

```
cons(3,cons(1,cons(4,nil)),R)
```

このグラフは，リンク R からリストを出力していると捉えられるので **0** $\rightarrowtail$ list(R) 型である．このようなグラフ全体の型は，先ほど定めた cons/3 アトムと nil/1 アトムの型をもとにして，入出力の型を相殺しながらグラフ同士を併合していくことで求めることができる．

$$\begin{aligned}
\textbf{型} \quad & \tau ::= \alpha \rightarrowtail \alpha \\
\textbf{単純型} \quad & \alpha ::= \mathbf{0} \mid p(\overrightarrow{X}) \mid \alpha, \alpha \\
\textbf{型環境} \quad & \Gamma ::= \mathbf{0} \mid p(\overrightarrow{X}) : \tau \mid \Gamma, \Gamma \\
\textbf{型判定式} \quad & \mathcal{J} ::= \Gamma \vdash G : \tau
\end{aligned}$$

図 7　LMNtype の構文

加えて注目すべきは，差分リスト（list segment, difference list）のような穴あきのグラフにも型が与えられる点である．例えば，

```
cons(3,cons(1,cons(4,X)),R)
```

は，list(X) $\rightarrowtail$ list(R) 型である．

これと同様に，LMNtype ではリスト以外の様々なデータ構造の差分や，穴あきの文脈 (context) のような不完全な構造に対しても型を与えることができる．特に LMNtal においては，関数型言語のように決まった場所から計算が進むということはなく，データ構造の中途が無作為かつ非決定的に書き換わっていく場面が多くある．そのため，グラフが穴あきの状態であっても型の情報が得られることは非常に重要である．

3.1　構文

LMNtype の構文を図 7 に示す．ただし，誤解なき場合は $\mathbf{0} \rightarrowtail \alpha$ を α と，$\mathbf{0} \vdash G : \tau$ を $\vdash G : \tau$ とそれぞれ略記する．

単純型中に出現する $p(\overrightarrow{X})$ を**型アトム**と呼ぶ．直観的には，型 $\alpha \rightarrowtail \beta$ において α は**入力** (source)，β は**出力** (sink) に対応する．ただし，入出力ともに単純型に含まれる型アトムは 0 個以上の任意の個数でよいとしているため，入力のないアトムや出力のないアトムがあってもよいし，入出力が複数あってもよい．

なお，構文の解釈に曖昧性が生じる場合，必要に応じ括弧を使用する．また便宜上，“$\rightarrowtail$” より優先度の低い区切り記号として，“,” のかわりに “.” を用いる．すなわち，区切り記号の優先順位は結合の強い方

から順に “,”, “$\rightarrowtail$”, “.” となる．

3.2 構文上の制約条件

各アトムのすべての引数に対して過不足なく型制約が与えられるようにするため，以下の制約条件を各構文要素に課す．

定義 3.1 (単純型のリンク条件)．一つの単純型中に同名のリンクは 1 回のみ出現できる．単純型 α 中のリンクの集合を $\mathcal{F}(\alpha)$ と書く． ◇

定義 3.2 (型のリンク条件)．型 $\alpha \rightarrowtail \beta$ は，リンク条件 $\mathcal{F}(\alpha) \cap \mathcal{F}(\beta) = \emptyset$ を満たす必要がある．記法を濫用し，$\mathcal{F}(\alpha \rightarrowtail \beta) \triangleq \mathcal{F}(\alpha) \cup \mathcal{F}(\beta)$ と書く． ◇

定義 3.3 (型定義のリンク条件)．型定義 $G : \tau$ は，リンク条件 $\mathcal{F}(G) = \mathcal{F}(\tau)$ を満たす必要がある． ◇

定義 3.4 (型環境の無矛盾性)．型環境 Γ は，以下の条件を満たす必要がある．

$$\forall (a_1 : \tau_1), (a_2 : \tau_2) \in \Gamma.$$
$$(a_1 : \tau_1) \neq (a_2 : \tau_2) \Rightarrow$$
$$\mathrm{Funct}(a_1) \neq \mathrm{Funct}(a_2)$$

すなわち，同一のファンクタに異なる型が割り当てられてはならない． ◇

3.3 意味論

LMNtype における型判定式の公理・推論規則を図 8 に示す．これらの規則から $\Gamma \vdash P : \tau$ が証明可能であるとき，型環境 Γ のもとでプロセス P は τ 型をもつといい，そのような τ が存在するとき P は Γ のもとで well-typed であるという．以降，文脈から明らかなときは型環境を省略し，$\Gamma \vdash P : \tau$ を単に $P : \tau$ と書く．

なお，$\tau[\overrightarrow{Y}/\overrightarrow{X}]$ は τ 中のリンクの置換を表す．例えば，$\tau[A, B/X, Y]$ は τ 中のリンク X を A，Y を B で置換したものである．ただし，これらの置換は同時並行に行われることに注意が必要である．例えば，$[X, Y/Y, X]$ はリンク名 X と Y の交換を表し，$[X/Y][Y/X]$ とは意味が異なる．

また，(T-Atom) に出現する “$\cdots \in \Gamma$” は Γ を集合とみなしたときの所属関係を表す．(T-Mol) に出現する型の間の二項演算 “$\oplus$” は以下の通り定義する．

$$(\alpha_1 \rightarrowtail \beta_1) \oplus (\alpha_2 \rightarrowtail \beta_2)$$
$$\triangleq (\alpha_1 \setminus \beta_2) \cup (\alpha_2 \setminus \beta_1) \rightarrowtail (\beta_1 \setminus \alpha_2) \cup (\beta_2 \setminus \alpha_1)$$

ただし，ここでの集合記法は，$\alpha_1, \alpha_2, \beta_1, \beta_2$ をそれぞれ多重集合とみなして取り扱う．(T-Mol) は厳密にはプロセスの分子 P_1,P_2 に関する規則であるが，プロセステンプレートの分子 T_1,T_2 についても同一の規則を準用する．(T-Rule) に出現する型の間の構造合同関係 $\tau \equiv \tau'$ は，既に定義済みの構造合同関係から次のように定義されるものとする．

$$(\alpha_1 \rightarrowtail \beta_1) \equiv (\alpha_2 \rightarrowtail \beta_2) \text{ iff } \alpha_1 \equiv \alpha_2 \wedge \beta_1 \equiv \beta_2$$

なお，以前に定義した構造合同関係は厳密にはプロセスに関するものであるが，型アトムの多重集合である単純型に対しても同様に定義されるものとする．

以下，各公理・推論規則の直観的な意味を述べる．(T-Zero) は空プロセス **0** に関する規則で，これは入力も出力も行わないため **0** 型とする．(T-Atom) は型環境の情報をもとにしてアトムに型付けを行う規則である．ただし，アトムと型アトムとの間の対応さえ変わらなければリンク名は何でも良い（α 変換）．(T-Conn1), (T-Conn2) はコネクタに関する規則であるが，この規則の存在意義は少々複雑であるため，5.2 節で詳述する．(T-Mol) は分子（プロセスの併合）に関する型付け規則で，併合されるプロセスの一方の出力が他方の入力になっている場合に，これらを相殺するための規則である．(T-Rule) はルールに関する型付け規則で，ルールの適用前後において型が不変である，すなわち両辺の型がバランスしていることを要求している．ただし，ルールはグラフとしては意味を持たない（書換えの対象とならない）ため，well-typed であれば必ず **0** 型となる．

また，閉じたグラフも well-typed であれば必ず **0** 型である．閉じたグラフは書換え対象として意味を持たないわけではないが，外部との接続関係がないため，ブラックボックスであって入出力をもたない．ただし，そのグラフ内部の接続関係が正しく整合している（well-typed である）ことは重要である．

3.3.1 組込みデータ型

これまでの定義によれば，型環境 Γ において型検査の対象となるプロセス P 中に出現するすべてのア

$$\frac{}{\Gamma \vdash \mathbf{0} : \mathbf{0}}\ \text{(T-Zero)} \qquad \frac{(p(\overrightarrow{X}) : \tau) \in \Gamma}{\Gamma \vdash p(\overrightarrow{Y}) : \tau[\overrightarrow{Y}/\overrightarrow{X}]}\ \text{(T-Atom)}$$

$$\frac{}{\Gamma \vdash X\texttt{=}Y : t(X) \rightarrowtail t(Y)}\ \text{(T-Conn1)} \qquad \frac{}{\Gamma \vdash X\texttt{=}Y : t(Y) \rightarrowtail t(X)}\ \text{(T-Conn2)}$$

$$\frac{\Gamma \vdash P_1 : \tau_1 \quad \Gamma \vdash P_2 : \tau_2}{\Gamma \vdash P_1\texttt{,}P_2 : \tau_1 \oplus \tau_2}\ \text{(T-Mol)} \qquad \frac{\Gamma \vdash T_1 : \tau \quad \Gamma \vdash T_2 : \tau' \quad \tau \equiv \tau'}{\Gamma \vdash T_1 \texttt{ :- } T_2 : \mathbf{0}}\ \text{(T-Rule)}$$

図 8　LMNtype の公理・推論規則

トムに対して型が与えられていることが，Γ のもとで P が well-typed となるための必要条件である．しかし，実用上 int, float, string のような基本的なデータ型には組込みで型が与えられていることが望ましい．

なお，LMNtal においてこうした基本的なデータ型は 0 価ではなく 1 価 (unary) である．例えば，整数 3 は LMNtal においては `3(X)` という整数をアトム名とする 1 価のアトムによって表現される．そこで，LMNtype においては `3(X)` アトムは `int(X)` 型をもつというように，従来の LMNtal において t 型 ($t \in \{\text{int}, \text{float}, \text{string}\}$) に分類されるアトム $p(X)$ については，いずれも $p(X) : t(X)$ が成り立つという前提条件を型環境へ暗黙に追加することとする．

さらに，実用的な LMNtal プログラムではこれらの基本的なデータ型に対する制約・演算を行うための**ガード**という拡張機能が用いられることが多い．このガードの扱いについては 5.3 節で後述する．

3.4　型検査の実装

以上で定義した LMNtype の検証用環境として，型検査器のプロトタイプ実装を作成した．

実装にあたり，グラフやルールの解釈を行う部分など，汎用的な部分については，以前に作成した LMNtal ShapeType のプロトタイプ実装 [10]（使用言語は TypeScript+Node.js）を流用し，追加実装する形で作成した．最終的に，全体のコード量は約 3,300 行，追加実装分は約 250 行となった．

4　例題

4.1　基本例題：リスト・二分木

まず，基本的な例題としてリストに関する LMNtal プログラムの型検査を紹介する．

はじめに，int 型を要素とするリストに関する型環境を次のように定義する．なお，以降の例題においては $\rightarrowtail$ を単に `->` で表記する．

```
cons(X,L,R): int(X),list(L) -> list(R).
nil(X): list(X).
```

3 節冒頭で触れた通り，リスト $[3, 1, 4]$ は次のような ($\mathbf{0} \rightarrowtail$) `list(R)` 型のグラフで表現可能である．

```
cons(3,cons(1,cons(4,nil)),R)
```

この型付けに関する証明木を図 9 に示す．注目すべきは，証明の過程で `cons(3,cons(1,cons(4,X)),R)` のような差分リストにも型の証明木が与えられている点である．

差分リストが型付け可能であることの利点を活かした例題として，隣接するリストの要素の入れ替えを行うルールは次のように書ける．

```
cons(X,cons(Y,L),R)
  :- cons(Y,cons(X,L),R).
```

これは，両辺ともに `int(X),int(Y),list(L)` $\rightarrowtail$ `list(R)` 型であるから，well-typed なルールであり，**0** 型となる．

このような無限に繰り返し適用可能なルールは，一見すると実行が終了せず不都合なように思われるが，非決定的な状態空間探索機能を有する LMNtal においては有用であり，あるリストに対してこのルールを非決定的に適用することで全通りの順列を得る，といった使い方がある．

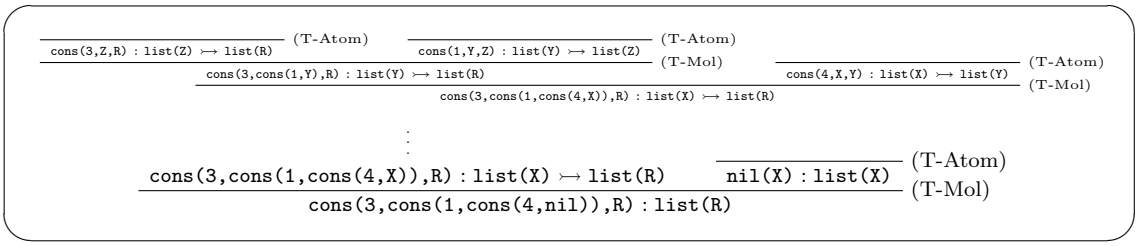

図 9　リストおよび差分リストの型付け証明木

加えて，このルールの発火に「リンク X の先に繋がる数値が Y の先の数値より大きい」という制約を課すことで，ソートを行うプログラムに変化させることもできる．こうしたルールの発火条件は，5.3 節で後述するガード機能によって制御できる．

重要な注意点として，LMNtype の型ではリストが環状となることを排除しない．例えば，次のグラフは要素 3, 1, 4 からなるリストの先頭と末尾を接続して環状にしたものである．

```
cons(3,cons(1,cons(4,X)),X)
```

なお，このグラフは，閉じているため **0** 型である．このようなグラフを型によって排除しないのは，LMNtype があくまで局所的なリンクによる接続とその前後の型の整合性にのみ着目しているためであるが，グラフ全体の形状に関して制約を掛けるための機能の案についても 5.4 節で議論する．

続いて，もう一つの基本例題として二分木を紹介する．

```
node(E,X,Y,R):
  int(E), tree(X), tree(Y) -> tree(R).
leaf(R): tree(R).
```

このもとで，例えば次のような型付けが成り立つ．

```
node(3,leaf,Y,R): tree(Y) -> tree(R).
```

4.2　スキップリスト

1 節で紹介したスキップリストのうち，2 レベルのものは次のような型環境のもとで型付けできる．

```
nil(L1,L2): exp(L1),local(L2).
n1(E,X1,L1):
  int(E),local(X1) -> local(L1).
n2(E,X1,X2,L1,L2):
  int(E),local(X1),exp(X2)
  -> exp(L1),local(L2).
```

ここでは，local(·) と exp(·) という 2 種類の型アトムを用いている．前者はすべてのノードを通過する「各駅停車」のリンクを，後者は一部のノードを通過する「急行」のリンクを表す．

この型環境のもとで，1 節で紹介した 2 つのルールについて順に見ていく．

```
n2(X,A,B,C,D), n2(Y,E,F,B,A)
  :- n2(X,A,F,C,D), n1(Y,E,A).
```

このルールは両辺とも

```
int(X),int(Y),local(E),exp(F)
                    ⟼ exp(C),local(D)
```

型であるので，well-typed である．

続いて，もう一つのルール例を見ていく．

```
n2(X,A,B,C,D), n2(Y,E,F,B,A)
  :- n2(X,A,E,C,D), n1(Y,F,A).
```

こちらは，左辺は前掲のものと同一であるが，右辺が

```
int(X),int(Y),exp(E),local(F)
                    ⟼ exp(C),local(D)
```

型であり，リンク E と F の役割が入れ替わっているため，ill-typed である．

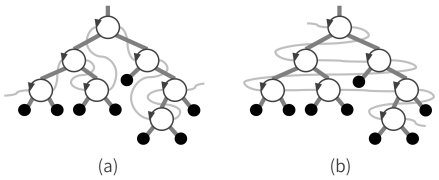

図 10　(a) 標準的な Threaded tree，(b) 幅優先の Threaded tree

4.3　Threaded Tree

Threaded tree とは，図 10 (a) に示すように，二分木のノードに対し中間順で直前・直後にあたるノードへのエッジを追加したデータ構造をいう．なお，通常は左右の部分木へのポインタが null となる箇所（外部節点）のみをバックエッジに置き換えることが多いが，LMNtal では全ノードの価数が一定であったほうが都合がよいので，すべてのノードに中間順のエッジを追加することとする．

LMNtype において，Threaded tree は次の型環境のもとで型付けできる．

```
n(H,L,R,T,P):
  thl(T),tht(L),tht(R)
  -> thl(H),tht(P).
l(P): tht(P).
```

ここでは，`tht(·)` と `thl(·)` という 2 種類の型アトムを用いている．前者は通常の二分木としての接続関係を，後者は中間順の接続関係を表している．つまり，ノードを表す `n`/5 アトムの第 1 引数は中間順で直前のノードへ，第 2 引数は左部分木へ，第 3 引数は右部分木へ，第 4 引数は中間順で直後のノードへ，第 5 引数は親ノードへとそれぞれ接続されている．

この Threaded tree に対して右回転を行うルールは以下の通りである．

```
n(H1,n(H2,X,Y,T2),Z,T1,R)
  :- n(H2,X,n(H1,Y,Z,T1),T2,R).
```

このルールは，両辺とも

```
thl(T1),tht(Z),thl(T2),tht(X),tht(Y)
                    ⟙ thl(H1),tht(R),thl(H2)
```

という型であるため，well-typed である．

ただし，LMNtype は接続関係の整合性のみを検査するため，`thl(·)` で管理されている接続が真に中間順の関係にあるかまでは保証できず，例えば図 10 (b) に示すように幅優先探索の順になっていたとしても排除できない．これは 5.4 節で議論する添字付きの型の領分であるが，これらの型を判別可能とするための具体的な制約条件については今後の課題とする．

5　議論

5.1　LMNtype の性質

LMNtype における主部簡約定理を示すため，まず以下の補題が必要となる．

補題 5.1 (型の一意性)**.** 型環境 Γ，コネクタを含まないプロセス P, P'，型 τ について，$P \equiv P'$ かつ $\Gamma \vdash P : \tau$ ならば $\Gamma \vdash P' : \tau'$ かつ $\tau \equiv \tau'$ を満たす τ' が存在する．

この補題の証明は本論文では取り扱わないが，$P \equiv P'$ の導出に関する構造的帰納法により証明可能と期待される．なお，型の演算 $\oplus$ の結合則の証明が補題として必要となることが予想される．

以下，主部簡約定理およびその証明を示す．

定理 5.1 (主部簡約)**.** 型環境 Γ，コネクタを含まないプロセス P, P'，型 τ について，$\Gamma \vdash P : \tau$ かつ $P \to P'$ ならば，$\Gamma \vdash P' : \tau'$ かつ $\tau \equiv \tau'$ を満たす τ' が存在する．　◇

Proof. LMNtal のルール適用においては，一度に一つのルールのみが書換えに関与し，それ以外のルールは書換えに影響を及ぼすことも及ぼされることもない．また，LMNtype の型付け規則から，well-typed なルールは必ず **0** 型となる．

以上のことから，型環境 Γ およびグラフ G, G'，プロセステンプレート T, U について，$\Gamma \vdash G, (T :\!\!- U) : \tau$ かつ $G, (T :\!\!- U) \to G', (T :\!\!- U)$ ならば $\Gamma \vdash G', (T :\!\!- U) : \tau'$ かつ $\tau \equiv \tau'$ を満たす τ' が存在することを示せば十分である．

このとき，$G,(T :\!\!- U) \to G',(T :\!\!- U)$ より，$G \equiv C,T$ かつ $G' \equiv C,U$ を満たすグラフ C が存在する．$T :\!\!- U$ が well-typed であることから，$T : \tau_T, U : \tau_U$ かつ $\tau_T \equiv \tau_U$ なる τ_T, τ_U が存在する．well-typed なルールは必ず **0** 型であるから，$G : \tau$ であり，$C : \tau_C$ なる τ_C が存在する．型付け規則 (T-Mol) から $C,U : \tau_C \oplus \tau_U$ であり，$\tau \equiv \tau_C \oplus \tau_T \equiv \tau_C \oplus \tau_U$ である．補題 5.1 および $T :\!\!- U : \mathbf{0}$ より，$G',(T :\!\!- U) : \tau_{G'}$ かつ $\tau \equiv \tau_{G'}$ なる $\tau_{G'}$ が存在する． □

一般的な型システムではこの主部簡約定理に加えて進行定理が必要な性質とされることがあるが，LMNtype ではこれを敢えて課していない．これは，LMNtal においては計算が終了した「値」の概念が定式化されておらず，「適用可能なルールがなくなったとき」が計算の終了と定義されているため，プログラマの意図した正常終了と，計算の行き詰まり状態とを区別するすべがないためである．ただし，[10] で行ったように，後天的に関数的アトム (functional atom) の概念を導入することで，計算の終了および進行に関して論ずることは可能であると考えられる．これについては本論文では扱わず，今後の課題としたい．

5.2 コネクタの扱い

前節の証明においては，対象となるプロセスはコネクタを含まないという仮定を置いていた．実際のところ，閉じたグラフ中に含まれるコネクタはすべて構造合同規則により吸収可能であり，ルールの左辺にコネクタが出現する場合は任意のリンクとマッチしてしまうため本質的な意味をなさない．しかし，ルールの右辺においてはコネクタが有用である場合がある．

例えば，リストの連結を行うプログラムは次のように書ける．

```
append(cons(X,L1),L2,R)
  :- cons(X,append(L1,L2),R).
append(nil,L,R) :- L=R.
```

ここで，append/3 アトムは第 1 引数と第 2 引数に cons/3 アトム（cons セル，第 1 引数は要素（head），第 2 引数は残余リスト（tail）と接続）および nil/1 アトム（リスト終端）からなる 2 本のリストを受け取り，連結して第 3 引数へ返却する．上記の 2 本目のルールで「最終的に第 1 引数のリストが nil になった際に append/3 アトムとともに消滅する」ことは，コネクタなしでは表現不可能である．

このルールの左辺は， append(nil,L,R) : list(L) $\rightarrowtail$ list(R) と型付けできる．そのため，左辺と右辺の型がバランスするためには右辺が L=R : list(L) $\rightarrowtail$ list(R) と型付けできる必要がある．以上の考察から，コネクタ X=Y は一方のリンクから入力された任意の 1 価の型アトムを他方のリンクへとそのまま伝播させると考え，規則 (T-Conn1), (T-Conn2) を導入した．

5.3 検査対象言語の拡張

本論文では，膜やガードといった高度な機能を除外した LMNtal のサブセット言語の上で議論を行ってきたが，これらの機能への対応は実用上重要と考えられる．

ガードについては，型制約（int 等）と LMNtype による型が無矛盾であることを検査すればそれで十分である．例えば，4.1 節で触れた隣接要素の交換によるソートを行うルールは次のように書ける．

```
cons($x,cons($x,L),R)
  :- $x>$y | cons($y,cons($x,L),R).
```

ここで，$x, $y はプロセス文脈と呼ばれるワイルドカード記法である．プロセス文脈は複数の引数を持ってもよく，その場合は $p[A,B] のように引数リストを角括弧で示す．また，アトムと同様の略記法が許されており，例えば $p[A,B],a(B,C) は a($p[A],C) と略記できる．

:- と | で挟まれた部分を**ガード**と呼び，カンマ区切りで左辺に出現するプロセス文脈に関する制約を記述する．ここでは$x>$y と書かれており，$x と $y は 1 価の int 型アトムであり，$x より $y の方が値が大きい，という意味になる．そのため，$x[X] および $y[X] が int(X) 型であるという情報が LMNtype の型付け情報と矛盾しなければ well-typed である．

また，ガードにユーザー定義の型による制約を掛けられるよう拡張した CSLMNtal [12] [13] という拡張言語も存在する．これとの融合についても模索しているところであるが，CSLMNtal の型は LMNtal ShapeType と同様に文脈自由グラフ文法に基づいて定義されるものであり，これと LMNtype との関連付けは自明でない．これら 2 つの型定義手法の差異については 5.5 節でも議論する．

5.4 添字付きの型について

4.1 節で触れた通り，これまでの LMNtype だけではグラフ全体の形状に関する保証は不可能である．環状のリストなど，通常の言語では扱うことの難しい形状のグラフを定義できることは LMNtal の長所である一方で，意図せずこのような構造が生じることを避けたい場面も多い．

そのため，型アトムに整数の添字を付し，型環境において型アトムの添字の間に制約を課すことが考えられる．例えば，リストが環状となることを禁じるには次のように型環境を変更する．

```
cons(X,L,R):
  int(X),list($x,L) ->
    $y=$x+1 |
  list($y,R).
nil(X): list(0,X)
```

ここでは，前節で触れたガードやプロセス文脈の記法を流用している．型アトム `list` は第 1 引数にその時点でのリストの長さを保持し，`cons` アトムはリストの長さを 1 だけ増加させる．これにより，リストが well-typed となるためには線形であることが要求されるようになった．なお，添字付きの型のもとでは，ルール両辺の型は多相的となる．

```
cons(X,cons(Y,L),R)
  :- cons(Y,cons(X,L),R).
```

このルールの両辺は整数パラメタ x を用いて次のように型付け可能である．

`int(X),int(Y),list(`x`,L)` $\rightarrowtail$ `list(`$x+1, R$`)`

この場合はパラメタまで含めて完全に一致するが，実際の型検査ではまず左辺の型検査を行い，そこで得られた情報が右辺と矛盾しないことを検査する必要がある．

同様に，スキップリストについても次のように拡張することで線形性が保証される．

```
nil(L1,L2): exp(0,L1),local(0,0,L2).
n1(E,X1,L1):
  int(E),local($e,$l,X1) ->
    $l1=$l+1 | local($e,$l1,L1).
n2(E,X1,X2,L1,L2):
  int(E),local($e,$l,X1),exp($e,X2) ->
    $e1=$e+1,$l1=$l+1 |
  exp($e1,L1),local($e1,$l1,L2).
```

ここでは `local` および `exp` の第 1 引数はその時点までで通過されていない（「急行」が停まる）ノードの数を，`local` の第 2 引数はその時点までのすべてのノードの数を表す．

さらに，赤黒木において，赤ノードが連続することがなく，かつ各ノードの黒高さ（葉からそのノードまでの黒ノードの数）が一定である，という 2 つの制約を満たしているかを確認することも可能である．

```
b(X,L,R,P):
  int(X),rbtree($lc,$lh,L),
  rbtree($rc,$rh,R) ->
    $lh=$rh, $lh1=$lh+1 |
  rbtree(1,$lh1,P).
r(X,L,R,P):
  int(X),rbtree(1,$lh,L),
  rbtree(1,$rh,R) ->
    $lh=$rh | rbtree(0,$lh,P).
l(P):
  rbtree(1,0,P).
```

ここで，型アトム `rbtree` の第 1 引数は root の色（1 ならば黒，0 ならば赤）を表し，第 2 引数は黒高さを表す．

上記の手法は，インデックス文法 [1] のアイディア

を LMNtal ShapeType に応用したインデックス型の概念に由来する [10]．なお，これと同様に型に対して数値制約をかける手法としては，関数型言語をベースとした型体系における篩型（refinement types [9]）が知られている．ただし，篩型では既存の型を元にして述語による制約をかけることで限定された型を得るのに対し，上記手法では型自体にパラメタが埋め込まれており，それらの間の関係に数値制約をかけているという違いがある．

5.5 型アトムの引数の数

本論文での定義では，型アトムは任意の個数の引数を持って良いこととしている．これは LMNtype が，LMNtal ShapeType に関する研究に端を発するものであることによる．LMNtal ShapeType では，型は複数の引数を持ってよく，例えば 2 レベルのスキップリストの型は 2 つの引数をもつ，というように設計されていた．

しかし，LMNtype では本質的にはアトム（もしくはその集合体であるグラフ）に対して型を付けているというより，むしろ各アトムの引数に対して型を付けているため，状況が少々異なる．例えば，4.2 節では 2 レベルのスキップリストに対し `local(·)` と `exp(·)` という 2 種類の型アトムで型付けを行ったが，LMNtal ShapeType と同様に 1 つの型アトム `skiplist(·,·)`（第 1 引数が「各駅停車」，第 2 引数が「急行」）で型付けを行おうとすると，次のようになる．

```
nil(L1,L2): skiplist(L2,L1).
n1(E,X1,L1):
  int(E),skiplist(X1,?)
  -> skiplist(L1,?).
n2(E,X1,X2,L1,L2):
  int(E),skiplist(X1,X2)
  -> skiplist(L2,L1).
```

`?` で示した部分では `n1` アトムの横を通過していくリンクに言及する必要があるが，直接接続関係のないリンクに言及するのは容易でない．

以上の考察から，型アトムの引数の数については，設計レベルにおいて 1 つと決め打ったほうが多くの場合で都合がよいとも考えられる．その場合，次のように型環境の表記を簡略化することも考えられる．例えば，

```
n2(E,X1,X2,L1,L2):
  int(E),local(X1),exp(X2)
  -> exp(L1),local(L2).
```

を，$\rightarrowtail$ の左（入力）は `+`，右（出力）は `-` を付すと決めることで，

```
n2(+int,+local,+exp,-exp,-local)
```

のように略記できる．なお，この略記法は Typed Prolog [5] に倣ったものである．

6 関連研究

5.5 節で少々言及した通り，重要な関連研究として Typed Prolog [5] がある．LMNtal 自体が Prolog 系の並行論理型言語に由来する言語であることもあり，LMNtal と Prolog とは構文上酷似しているが，Prolog では関数記号と述語の間に構文レベルで区別があるのに対し，LMNtal では構文上も意味論上もデータとプロセスとを区別せずに扱うという点で大きく異なっている．特に，型という観点から言えば，Prolog では「述語を起点として，関数記号からなるデータの型を考える」という考え方になるが，LMNtal ではそのような起点となる構文要素が存在しない．

加えて，Prolog が対象とするデータ構造は，差分リストのように穴あきの構造は扱えるものの，基本的には代数的データ型を出発点として設計されているのに対し，LMNtal ではもとより一般のグラフを対象としているため，環状構造なども許容している．

また，Prolog 系の論理型言語においては入力と出力の両方を述語の引数に並べて表記するため，各引数に入出力モードとして後天的に入出力の区別を与え，これを広義の型概念として扱っている．Typed Prolog では入出力の方向性についてはこのモード解析に委ねており，Type の側では扱わないという立場を採っているのに対し，LMNtype では入出力のデー

タの種類と方向性を同時に扱っている．

一方，通信を扱う型という観点では Session Type [4] がよく知られている．Session Type では各参加者 (participant) が送受信すべきデータ型を時系列順に並べたものを型として扱うのが基本思想であるが，LMNtype では各コンストラクタは固定の入出力をもち，計算の進行に伴ってグラフ構造が変容していくことによって，結果的にプロセスの各部が送受信する型が変容していくことになる．これらのより詳細な関連付けについては，今後の課題としたい．

7 まとめと今後の課題

本論文では，グラフ書換え言語 LMNtal を対象とした比較的軽量な静的型検査手法として LMNtype を提案し，例題に触れつつその性質について議論した．

これまでの節で触れなかった今後の課題のうち重要なものとしては，型推論への対応が挙げられる．本論文の範疇では，型アトムの命名は完全に人の手に依るものであったが，モード解析由来の解析手法 [8] と同様にアトムの引数（ポート）ごとにその役割をある程度推論することは可能であると考えられる．また，Typed Prolog における型再構築の手法も参考になるだろう．

謝辞 本研究の一部は科学研究費補助金 (23K11057)，早稲田大学特定課題研究費 (2023C-412) の補助を得て実施した．

参 考 文 献

[1] Aho, A. V.: Indexed grammars – An extension of context free grammars, *8th Annual Symposium on Switching and Automata Theory (SWAT 1967)*, 1967, pp. 21–31.

[2] Fradet, P. and Métayer, D. L.: Shape types, *Proc. POPL'97*, ACM, 1997, pp. 27–39.

[3] Fradet, P. and Métayer, D. L.: Structured Gamma, *Science of Computer Programming*, Vol. 31, No. 2(1998), pp. 263–289.

[4] Honda, K., Vasconcelos, V. T., and Kubo, M.: Language primitives and type discipline for structured communication-based programming, *Proc. ESOP'98*, LNCS, Vol. 1381, Springer, 1998, pp. 122–138.

[5] Lakshman, T. and Reddy, U. S.: Typed Prolog: A Semantic Reconstruction of the Mycroft-O'Keefe Type System, *Proc. ILPS'91*, MIT Press, 1991, pp. 202–217.

[6] Pugh, W.: Skip lists: A probabilistic alternative to balanced trees, *Commun. ACM*, Vol. 33, No. 6(1990), pp. 668–676.

[7] Ueda, K.: LMNtal as a hierarchical logic programming language, *Theoretical Computer Science*, Vol. 410, No. 46(2009), pp. 4784–4800.

[8] Ueda, K.: Towards a Substrate Framework of Computation, *Concurrent Objects and Beyond*, Agha, G. et al.(eds.), LNCS, Vol. 8665, Springer, 2014, pp. 341–366.

[9] Vazou, N., Seidel, E., Jhala, R., Vytiniotis, D., and Peyton Jones, S.: Refinement Types For Haskell, *ACM SIGPLAN Notices*, Vol. 49(2014).

[10] Yamamoto, N. and Ueda, K.: Engineering Grammar-Based Type Checking for Graph Rewriting Languages, *IEEE Access*, Vol. 10(2022), pp. 114612–114628.

[11] 後町将人, 堀泰祐, 上田和紀: LMNtal 実行時処理系の並列モデル検査器への発展, コンピュータ ソフトウェア, Vol. 28, No. 4(2011), pp. 4_137–4_157.

[12] 奈良耕太, 上田和紀: パターン定義によるマッチングを導入したグラフ書換え言語とその実装, 日本ソフトウェア科学会第 *31* 回大会 *(2014* 年度*)* 講演論文集, 2014.

[13] 白井涼也, 今川連, 山本直輝, 上田和紀: 再帰的なグラフパターンに基づく反復パターンマッチングの効率化手法, 日本ソフトウェア科学会第 *40* 回大会 *(2023* 年度*)* 講演論文集, 2023.