データ構造としてのグラフを対象とした型の表現力拡張

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．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が課題となっている．そこで我々は，グラフの型に関する枠組みとして，形式文法に基づいた LMNtal ShapeType を提案し，文脈自由を超えた広範な型に対応した型検査手法を定式化してきた．しかし，ラムダ式のグラフによる表現やグリッドグラフなど，ルートの本数が一定でない型については，型の定義や検査手法が他の型に比べても自明でない．そこで本論文では，LMNtal ShapeType の定義および検査手法に対して拡張を施すことで，以上のようなグラフ型を対象とする型検査手法の定式化を目指す．

1　はじめに

ネットワーク構造は，インターネットをはじめとして，情報網や交通網，あるいは人間関係など，様々な分野でみられる．あるいは，ミクロな視点で見ると，例えばコンピュータは各部品が接続関係にあるネットワークであり，それらの部品を構成する電子回路は素子が相互に接続されたネットワークである．こうしたネットワークの構造を抽象化・モデル化したデータ構造がグラフである．

一般のグラフは，代数的データ型（リストや木構造）よりも複雑な構造をもち，それ故に既存のプログラミング言語では扱うことが難しい．例えば，C 言語のようなポインタを扱える言語であれば，ポインタの接続構造としてグラフを表現することができるが，依然としてダングリングポインタなどの不正なポインタが発生する危険性がある．

そこで，グラフを第一級オブジェクトとしてもち，それ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．

LMNtal[20] は，グラフ書換え言語の一つであり，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．しかしながら，LMNtal は十分な表現力・計算能力をも持ち合わせている．実際，LMNtal はチューリング完全であり，ラムダ計算などの他の言語体系を LMNtal プログラムにエンコードする手法が存在している[11][12]．

また，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．実際，LMNtal の実行時処理系である SLIM[21][19] には，これらの 2 つの側面とそれぞれ対応する 2 種類の実行モードがある．すなわち，LMNtal をプログラミング言語とみなす場合には SLIM は実行時処理系となり，モデリング言語とみなす場合には SLIM はモデル検査器となる．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 モデル検査など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

---

* Extending the Expression Power of Types for Graphs as a Data Structure.

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.

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において不正なポインタを排除している（すなわち，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 [8] は線形リストに対して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，図 1 (a) に示すようなデータ構造である．これは LMNtal グラフとして図 1 (b) のように表現できる．

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,E,C,D), n1(Y,F,A).
```

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

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をひと目で判断することは難しい．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．

こうした問題点を解決するため，これまでにも LMNtal を対象とする幾つかの静的型検査手法が提案されている．例えば，[12] の手法ではノード間の局所的な繋がりに着目し，入出力の方向性を解析し，それを -1 から 1 までの実数値（極性）として表現することで型を付ける．

一方，本研究で扱う **LMNtal ShapeType** [17] は，Structured Gamma [5] およびそのサブセットである Shape types [4] を基にしてグラフの型を定義・検査する手法である．この手法では，LMNtal のルールを生成規則として，生成文法により型の定義を行う．そのため，SLIM のモデル検査機能を用いることによって，LMNtal 自身で LMNtal の型検査を行えるという利点がある．ある言語の型をその言語自身で記述し，その言語自身によって型検査を行うという取り組みが統一性の観点から興味深いだけでなく，LMNtal の柔軟な表現力をそのまま型の定義に活かすことができるのは大きな強みである．

我々はこれまでの研究 [18] [15] において，LMNtal ShapeType の表現力拡張を模索し，文脈自由を超えた広範な型に対応した型検査手法を定式化してきた．しかし，ラムダ式のグラフによる表現やグリッドグラフなど，ルートの本数が一定でない型については，型の定義や検査手法が他の型に比べても自明でない．そこで本論文では，LMNtal ShapeType の定義および検査手法に対して更なる拡張を施すことで，以上のようなグラフ型の定義を可能にし，型検査手法の定式化を目指す．

2 グラフ書換え言語 LMNtal

本節では，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．しかしながら，これらの言語機能をもたない LMNtal もまたチューリング完全であり，プログラミング言語としてもモデリング言語としても十分強力である．

2.1 構文

LMNtal の構文を図 3 に示す． LMNtal プログラムは，**グラフ**（アトムの多重集合）と**ルールセット**（ルールの多重集合）の組として表現され，これを**プロセス**と呼ぶ．

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

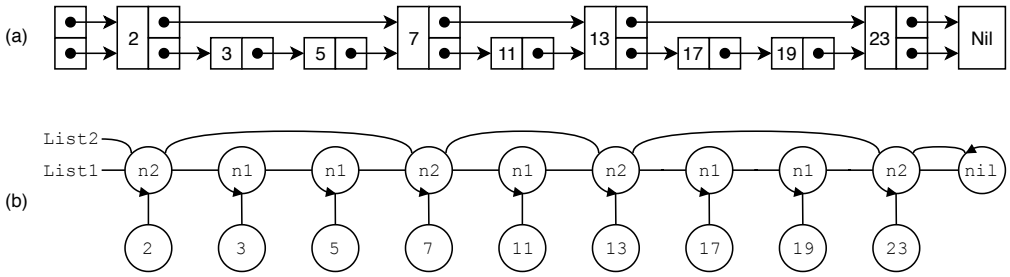

図 1　スキップリスト：(a) グラフによる表現 および (b) LMNtal グラフによる表現

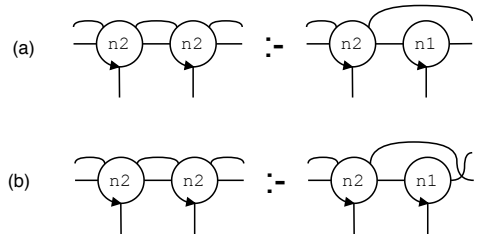

図 2　スキップリストの書換え規則：(a) 正しいもの および (b) 誤ったもの

プロセス　$P ::= G, R$

グラフ

$G ::= \mathbf{0}$　（空）

| $p(X_1, \ldots, X_m)$　（アトム, $m \geq 0$）

| G, G　（分子）

ルールセット

$R ::= \mathbf{0}$　（空）

| $[\,RuleName\,@@\,]\; G :- G$　（ルール）

| R, R　（分子）

図 3　LMNtal の構文

書き,「m 価の p アトム」のように読む.

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．図中において矢印は第 1 引数の

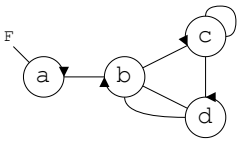

図 4　LMNtal グラフの例

場所を指しており，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示す.

グラフはいずれも次の**リンク条件**を満たす：

> グラフ中に同名のリンクは高々 2 回出現する.

グラフ中に 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．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 α 変換されたものとみなす.

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 †1．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．また可読性のために，ルールはカンマで区切るかわりに，ピリオドで終わる形で書かれることもある．なお，書換えの前後において自由リンクの本数が増減することはあり得ないので，ルール中に同じリンクはちょうど 2 回だけ出現する.

†1　本論文では簡単のため，コネクタはルールの左辺・右辺のいずれにも出現しないものとする.

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

図 5　LMNtal グラフ上の構造合同関係

2.2　意味論

LMNtal の意味論は，**構造合同**と**遷移規則**からなる．これらについて順に説明する．

2.2.1　構造合同

まず，グラフ間の構造合同関係 “$\equiv$” を，図 5 の規則を満たす最小の同値関係として定義する．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，構造合同な項は同じグラフを意味する．なお，$P[Y/X]$ はグラフ 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)–(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．

2.2.2　遷移規則

LMNtal における本質的な計算ステップを表す，グラフ間の二項関係として，**ルール** T `:-` U **によるグラフの遷移関係** “$\xrightarrow{T\ \texttt{:-}\ U}$” が存在する．これは，図 6 に示す遷移規則をみたす最小の二項関係として定義

$$
\begin{array}{ll}
\text{(R1)} & \dfrac{G_1 \xrightarrow{T\ \texttt{:-}\ U} G_1'}{G_1,G_2 \xrightarrow{T\ \texttt{:-}\ U} G_1',G_2} \\[2ex]
\text{(R3)} & \dfrac{G_2 \equiv G_1 \quad G_1 \xrightarrow{T\ \texttt{:-}\ U} G_1' \quad G_1' \equiv G_2'}{G_2 \xrightarrow{T\ \texttt{:-}\ U} G_2'} \\[2ex]
\text{(R6)} & T \xrightarrow{T\ \texttt{:-}\ U} U
\end{array}
$$

図 6　LMNtal グラフ上の遷移関係

される．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，その左辺にパターン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．

また，この定義を自然にルールセットへと拡張することができる．すなわち，$\exists r \in R.\ G \xrightarrow{r} G'$ が成り立つとき「ルールセット R によって，グラフ G が（1 ステップで）G' に遷移する」といい，単に $G \xrightarrow{R} G'$ と書く．

なお，通常の LMNtal においては，アトムとルールの多重集合であるプロセスが，遷移関係に基づいて**おのずから書き換わっていく**．そのため,「ルールによる作用を受けてグラフが書き換わる」とは考えない．しかしながら，書換えに関する静的な性質について定義および議論をするにあたっては，書換えの主体たるルールと客体たるグラフとは明確に分離されていたほうが都合が良い．よって，本論文では LMNtal の構文・意味論を簡明な形に等価変換して扱っている．

2.3　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．

3 LMNtal ShapeType

本節では，LMNtal を対象とする静的な型検査手法である LMNtal ShapeType を紹介する．まず，形式的な定義を先に示しておく．なお，以下で “記号” と言うときは，LMNtal のファンクタのことを指す．

定義 3.1. LMNtal ShapeType における型（以下，これを単に “ShapeType” という）は，3 つ組 (S, P, N) である．ここで，

- $S = t/m$ は，**開始記号**と呼ばれるファンクタ
- P は，**生成規則**と呼ばれるルールの有限集合
- N は，**非終端記号**と呼ばれるファンクタの有限集合

とする．ただし，$S \in N$ とする． ◇

3.1 構文

ShapeType の構文を図 7 に示す．生成規則は，LMNtal のルールとして well-formed である必要がある．生成規則の左辺は 1 つ以上の（コネクタでない）非終端アトムのみからなる．また，LMNtal と同様の略記法（2.3 節）を認める．`nonterminal{`N`}` は S 以外の非終端記号の宣言であり，無ければ省略して良い．

ShapeType
::= `defshape` S `{` P `}` [`nonterminal{` N `}`]

開始記号
S ::= p/m

生成規則の有限集合
P ::= $\mathbf{0}$ | [$RuleName$ `@@`] A `:-` A | P`,`P

非終端記号の有限集合
N ::= $\mathbf{0}$ | p/m | N`,`N

アトムの有限集合
A ::= $\mathbf{0}$ | $p(X_1, \ldots, X_m)$ | A`,`A

図 7 ShapeType の構文

例えば，1 節で紹介したスキップリストの場合，次のように型定義できる．

```
defshape skiplist/2 {
  p0@@ skiplist(L2,L1) :- nil(L2,L1).
  p1@@ skiplist(L2,L1)
    :- n1(X1,L1), skiplist(L2,X1).
  p2@@ skiplist(L2,L1)
    :- n2(X1,X2,L2,L1), skiplist(X2,X1).
}
```

なお，図 1 (b) で示したスキップリストの各ノードには整数型の要素が繋がっているが，以降の本論文中ではグラフの形状を重視するために，こうしたデータ構造に含まれるデータは省略して考える．例えば，上記の型をもつグラフは図 8 に示すような形状をしている．

以上のように定義された型を，開始記号 $S = t/m$ を用いて t/m 型（あるいは m 価の t 型）と呼び，誤解のおそれがないときは価数を省略して単に t 型と呼ぶ．

3.2 意味論

型付け関係 “:” を定義するにあたって，補助的な型付け関係 “$\lhd$” を先に定義する．以下では，$\vec{L} = L_1, \ldots, L_m$ だけを自由リンクに持つグラフを $G[\vec{L}]$ と書く．また，グラフ G 中で使用されているファンクタ全体の集合を，$\mathrm{Funct}(G)$ と表す．

定義 3.2. 型 $(t/m, P, N)$ について，$t(\vec{L}) \xrightarrow{P}{}^{*} G[\vec{L}]$ であるとき，かつそのときに限り，$G[\vec{L}]$ は $t(\vec{L})$ 型によって**生成される**といい，$G[\vec{L}] \lhd t(\vec{L})$ と書く． ◇

これは直観的には，t 型の開始記号から生成規則によって 0 ステップ以上遷移した先のグラフだけが t 型によって生成されることを表す．

次に，型付け関係 “:” を定義する．

定義 3.3. 型 $(t/m, P, N)$ について，$G[\vec{L}] \lhd t(\vec{L})$ かつ $\mathrm{Funct}(G[\vec{L}]) \cap N = \varnothing$ であるとき，かつそのときに限り，$G[\vec{L}]$ は $t(\vec{L})$ 型を**もつ**といい，$G[\vec{L}] : t(\vec{L})$ と書く． ◇

これは直観的には，t 型によって生成されるグラフのうち，非終端記号を含まないグラフのみが，t 型をもつことを表す．

また，開始記号および（型付けの対象である）グラ

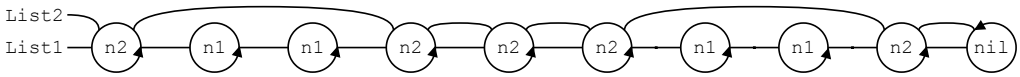

図 8　要素を無視したスキップリスト型グラフの例

フの自由リンクの名前と順序の対応関係には意味がある．例えば，

```
defshape t/2 { t(X,Y) :- a(X,Y) }
```

という型があったときに，`a(X,Y)` $\lhd$ `t(X,Y)` であるが，`a(Y,X)` $\lhd$ `t(X,Y)` ではない．

3.3　型検査

LMNtal ShapeType には，2 つの基本的な型検査手法がある．一つは，**グラフの型検査**である．これは，与えられたグラフが，与えられた型に属しているかを検査するものである．もう一つは，**ルールの型検査**である．これは，ルールと型が与えられたときに，ルールの適用によって型の構造が破壊されないことを検査するものである．

ルールの型検査について，もう少し形式的に解説する．まず，型 t に関する LMNtal ルール r の型保存性を定義する．

定義 3.4. ルール r と型 t/m およびリンク列 $\vec{L} = L_1, \ldots, L_m$ について，全ての $G : t(\vec{L})$ が，

$$G \xrightarrow{r} G' \Rightarrow G' : t(\vec{L})$$

を満たすとき，かつそのときに限り，r は t 型を**保存する**という．　◇

自然言語で表現すると，「t 型をもつ全てのグラフ G は，ルール r を（可能なら）適用した後も，t 型を持っている」という性質である．このルールの型保存性の検査のことを，単にルールの型検査という．

3.4　具体的な型検査手法の概略

以下では，2 つの検査手法について概略を述べる．より詳細な型検査の手法およびその正当性証明については文献[15] [18] で詳しく述べているため，そちらを参照されたい．

まず，LMNtal ルールおよびその逆実行について，以下の性質が成り立つ．

定義 3.5. LMNtal ルール $r = T$ `:-` U の**反転** r^{inv} を，$r^{\mathrm{inv}} \triangleq U$ `:-` T で定める．同様に，LMNtal ルールセット R の反転 R^{inv} を，$R^{\mathrm{inv}} \triangleq \{r^{\mathrm{inv}} \mid r \in R\}$ で定める．　◇

定理 3.1. 両辺ともにコネクタを持たない LMNtal ルール $r = T$ `:-` U について，

$$G' \xrightarrow{r} G \Leftrightarrow G \xrightarrow{r^{\mathrm{inv}}} G'$$

である．　◇

この定理は，ルールの左辺と右辺を入れ替えた反転ルールの適用によって，遷移関係を逆向きに辿ることができると主張している．

グラフの型検査の目標は，開始記号に生成規則を 0 回以上適用して検査対象グラフ G に遷移できるかを判定することである．そのため素朴には，定理 3.1 より，逆に検査対象グラフ G から出発し，生成規則の反転を 0 回以上適用して開始記号に遷移できるか否かを調べればよい．

次に，ルールの型検査の概略は以下の通りである．まず，各生成規則 α `:-` β について，$\beta \equiv \beta_1$`,`β_2 かつ β_1 が空でないようなグラフ β_1, β_2 をとる．ここで，状態 L `:-` R に対し，$L \equiv \beta_1$`,`L' であるならば，この状態を α`,`L' `:-` β_2`,`R へと書き換えるような書換え規則を作成する．以上のようにして得られる全ての書換え規則をルールセットとし，検査対象ルールを初期状態として状態空間を生成する．こうしてできた状態空間において，初期状態から出発して，左辺が開始記号であるような状態へと到達するような任意の経路上に，必ず reducible（左辺から右辺へと生成規則を 0 回以上適用して遷移可能）な状態が存在することを検証すれば，型保存性を検査できる．

なお，以上の手順で生成される状態空間はそのままでは一般に有限とならない．そのため，実際の実装においては左辺が同じ状態は同一とみなして状態空間を構築する．ただし，このように左辺の構造合同関係で商を取った状態空間で，初期状態から左辺が開始記号であるような状態へ至る経路上に閉路が出

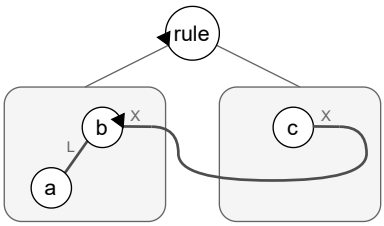

図 9　第一級書換え規則によるルールの表現

現する場合，その閉路に進入した最初の状態において reducible であるかの検査を行い，それ以上は遡らない．

以上の手法ではルール自体をメタ的に書き換えて型検査を進めているが，このような計算は LMNtal メタインタプリタ [10] における第一級書換え規則のアイディアを応用することにより実現できる．

まず，第一級書換え規則とは，LMNtal においてルールを第一級に，すなわち書換えの対象として扱うために考案された書換え規則の表現形式である．この形式においては，ルールは左辺と右辺を表す 2 つの**膜**と，それらを束ねるアトムの形で表現される †2．例えば，`a(L),b(X,L) :- c(X)` というルールを第一級書換え規則として表現すると，

```
rule({a(L),b(X,L)},{c(X)})
```

となり，これを図示したものが図 9 である．なお，テキスト表現では `{...}`，図中においては角丸四角形で表される**膜**とは，本論文における型付け対象言語には含まれない LMNtal の言語要素で，アトムをグループ化する機能を有する機構である．

以上のように第一級書換え規則により検査対象ルールをグラフとして表現することで，検査対象ルールを直接書き換えてルールの型検査を行うことができる．

3.5　生成規則及び型の文脈自由性

7 節で後述するように，グラフを対象とする既存手法の多くでは，生成規則の形に次のような制限を加えることで型検査の停止性を担保している．

定義 3.6. 生成規則 $p = \alpha$:- β について，α がただ一つのアトムからなり，β がコネクタ以外のアトムを一つ以上含むとき，p は**文脈自由**な生成規則であるという．また，生成規則が全て文脈自由である型を**文脈自由な型**という．　◇

グラフの型検査に関しては，型検査が必ず停止し，かつ false-negative とならないような，文脈自由より広い型のクラスが既に発見されている [15] が，ルールの型検査については，文脈自由より型定義のクラスを広げると容易に状態空間のサイズが無限となり，停止性を満たさなくなることがわかっている．これは，直観的にはルールの型検査が「必要なグラフを仮定として補った上で生成規則の逆適用を行う」ことに基づいているためである．

文脈自由な型に関するルールの型検査が停止する理由を簡単に述べる．文脈自由な型において，生成規則左辺は必ず 1 つのアトムのみからなるため，逆適用前にアトムをいくつ補ったとしても，逆適用後は必ずアトム 1 つとなる．よって，逆適用によってアトムの数が増加することはない．すなわち，逆実行による状態空間は（左辺の構造合同関係で商をとると）有限となる．

型の表現力を向上するために単に文脈自由の前提を崩してしまうと，上記の状態空間の有限性に関する議論までも成り立たなくなってしまう．したがって，ルールの型検査をより広範な型へと応用するにあたっては，生成規則左辺がひとまとまりであるという前提を崩さずに型定義を拡張することが重要となる．

4　インデックス型

4.1　背景

文脈自由な型により，二分木や線形リストといった代数的データ型についてはもちろん，根（自由リンク）が複数あるスキップリストや差分リスト，あるいはルートが一つもないリング構造のように，代数的データ型では表現ができないグラフの型も表現することができる．

しかし，高さ n の完全二分木や（平衡している）赤黒木，あるいは長さ n のリストといったような数値

†2　本論文における第一級書換え規則の表現は，実装上の都合により [10] で導入されたものと若干異なる．特に，原典では自由リンクはハイパーリンク（超辺）によって表現されていたが，本論文では通常のリンクにより表現している．

```
defshape rbtree3/1 {
  bb3@@ rbtree3(R) :- b(tree2,tree2,R).
  bb2@@ rbtree2(R) :- b(tree1,tree1,R).
  bb1@@ rbtree1(R) :- b(tree0,tree0,R).
  bl@@ rbtree0(R) :- leaf(R).
  tb2@@ tree2(R) :- b(tree1,tree1,R).
  tb1@@ tree1(R) :- b(tree0,tree0,R).
  tr2@@ tree2(R) :- r(rbtree2,rbtree2,R).
  tr1@@ tree1(R) :- r(rbtree1,rbtree1,R).
  tr0@@ tree0(R) :- r(rbtree0,rbtree0,R).
  tl@@ tree0(R) :- leaf(R).
} nonterminal {
  rbtree3/1, rbtree2/1, rbtree1/1,
  rbtree0/1, tree2/1, tree1/1, tree0/1
}
```

図 10　黒高さがちょうど 3 の赤黒木の型定義

制約の付いた型は表現することが難しい．もちろん，文脈自由でない型を許せばこうした型を表現することはできるが，その場合は型検査における状態空間の有限性が自明でなくなる．

一方で，数値制約が定数であるもの，例えばちょうど黒高さが 3 の赤黒木は図 10 のように文脈自由な型で表すことができる． ここで登場した非終端記号のうち，`rbtreeN` は高さ `N` かつ根が黒である赤黒木，`treeN` は黒高さ `N` の赤黒木（根は赤でも黒でもよい）を表し，終端記号の `r` は赤ノード，`b` は黒ノード，`leaf` は葉ノードを表す．なお，黒高さとは根から葉に至るまでの経路上に出現する黒ノードの数である．赤黒木の性質上，黒高さは葉によらず一定であることが要求される．この定義において，`bb3` から `bb1`，`tb2` から `tb1`，`tr2` から `tr0` はそれぞれほとんど同一の生成規則であり，非終端記号の添字が変わっているにすぎない．

以上の観察結果から，添字付きの非終端記号の出現を認めることにより，赤黒木のような数値制約付きの型の定義を簡略化することができると考えられる．本論文では，このような添字付きの非終端記号の出現を認める拡張を**インデックス型**と呼ぶ．なお，こうした添字付きの非終端記号という着想は，形式言語理論におけるインデックス文法（indexed grammar [1]）から得たものである．

4.2　インデックス型の形式的な定義

添字付きの非終端記号の出現を認めるインデックス型を定式化するにあたっては，まずアトムが添字を持つことができるように LMNtal グラフの定義を拡張する必要がある．

定義 4.1. 添字により拡張された LMNtal グラフの構文を以下の BNF で定める．

$$
\begin{array}{rcll}
G & ::= & \mathbf{0} & \text{(空)} \\
 & | & p\texttt{<}x_1,\ldots,x_n\texttt{>(}L_1,\ldots,L_m\texttt{)} & \\
 & & \text{(添字付きアトム, } n \geq 0,\ m \geq 0\text{)} & \\
 & | & G,G & \text{(分子)}
\end{array}
$$

ここで，

- $x_1,\ldots,x_n$ は変数を表す**プロセス文脈** `$`p または非負整数定数からなる列であり，これらを（アトムの）**添字**という[†3]．
- $p\texttt{<}x_1,\ldots,x_n\texttt{>(}L_1,\ldots,L_m\texttt{)}$ を**添字付きアトム**という．
- 添字付きアトムの名前と添字数と価数からなる三つ組を $p/n/m$ のように書き，これを（添字付きアトムの）**ファンクタ**という．

なお，(従来記法との整合性のため) $p\texttt{<>(}L_1,\ldots,L_m\texttt{)}$ のように添字が 0 個である場合は，添字リストの括弧を省略して $p\texttt{(}L_1,\ldots,L_m\texttt{)}$ と書いても良い．同様に，ファンクタ $p/0/m$ は p/m と書いても良い． ◇

次に，LMNtal ルールとそれによる遷移規則についても次のように拡張し，ルールが発火するための条件を記述できるようにする．

定義 4.2. ガード付きの LMNtal ルールの構文を以下で定める．

$$[\mathit{RuleName}\ \texttt{@@}]\ G\ \texttt{:-}\ [C\ \texttt{|}]\ G\,.$$

ここで，C は**ガード**と呼ばれる 0 個以上の等式・不等式制約の列であり，省略されている場合は空列とみなす．なお，今回使用できる演算はプレスバーガ算術の範疇のみとする（加減算・定数倍）． ◇

定義 4.3. 遷移規則 (R6) を以下の通り拡張する．

$$G\theta \xrightarrow{G\ \texttt{:-}\ C\ \texttt{|}\ G'} G'\theta \quad \text{iff} \quad \theta \text{ が } C \text{ を充足する}$$

†3　ここで新たに導入した `$`p といった記法および「プロセス文脈」という名称は，型付きプロセス文脈 [22] という任意性のあるグラフにマッチするための LMNtal の拡張機能に由来するものであるが，ここでは単に変数を表す記法として使用している．

```
defshape rbtree/1/1 {
  bb@@ rbtree<$n>(R)
    :- $m = $n-1 | b(tree<$m>,tree<$m>,R).
  bl@@ rbtree<0>(R) :- leaf(R).
  tb@@ tree<$n>(R)
    :- $m = $n-1 | b(tree<$m>,tree<$m>,R).
  tr@@ tree<$n>(R)
    :- r(rbtree<$n>,rbtree<$n>,R).
  tl@@ tree<0>(R) :- leaf(R).
} nonterminal { tree/1/1 }
```

図 11　インデックス型を用いた赤黒木の型定義

なお，θ はプロセス文脈に対する具体値（非負整数定数）の代入であり，C に含まれる全ての制約が θ により充足されるとき，θ は C を充足する代入であるという．◇

以上の準備のもとで，LMNtal ShapeType の定義を拡張し，インデックス型を扱えるようにする．

定義 4.4. 以下の三つからなる組 (S, P, N) を LMNtal ShapeType における型と呼ぶ．

- $S = t/n/m$：開始記号と呼ばれる，ファンクタ
- P：生成規則と呼ばれるルールの有限集合
- N：非終端記号と呼ばれるファンクタの有限集合

ただし，生成規則の左辺・右辺に登場するアトムのファンクタのうち N に含まれないもの（終端記号）は添字をもってはならない．◇

以下，$(t/n/m, P, N)$ は型，$\vec{L} = L_1, \ldots, L_m$ は自由リンクの列，$\vec{c} = c_1, \ldots, c_n$ は具体値（非負整数定数）の列であるとする．

定義 4.5.

$$t\texttt{<}\vec{c}\texttt{>(}\vec{L}\texttt{)} \xrightarrow{P}{}^{*} G[\vec{L}]$$

であるとき，グラフ $G[\vec{L}]$ は型 $t\texttt{<}\vec{c}\texttt{>(}\vec{L}\texttt{)}$ により**生成される**といい，$G[\vec{L}] \lhd t\texttt{<}\vec{c}\texttt{>(}\vec{L}\texttt{)}$ と書く．◇

定義 4.6.

$$G[\vec{L}] \lhd t\texttt{<}\vec{c}\texttt{>(}\vec{L}\texttt{)} \ \wedge\ \mathrm{Funct}(G[\vec{L}]) \cap N = \varnothing$$

であるとき，グラフ $G[\vec{L}]$ は型 $t\texttt{<}\vec{c}\texttt{>(}\vec{L}\texttt{)}$ を**もつ**といい，$G[\vec{L}] : t\texttt{<}\vec{c}\texttt{>(}\vec{L}\texttt{)}$ と書く．◇

以上のように型の定義を拡張した上で，任意の高さの赤黒木の型は図 11 のように記述できる．

4.3　インデックス型の性質に関する議論

インデックス型の定義において，添字は任意の非負整数値をとる可能性がある．よって，無限通りの値をとることが考えられ，状態空間が爆発する危険がある．そのため，状態空間構築においては非終端記号の添字だけの違いは無視し，同一状態とみなすこととする．

以上の仮定により，見かけ上は添字なしの文脈自由な型と同一の状態空間が生成されるため，型検査の停止性が保証される．一方で，前述のちょうど高さが 3 の赤黒木の例のように文脈自由な型の範囲で表現された型の場合と比べると，複数の状態が一つの状態に縮約されるため，結果が false-negative となる可能性が少し高まる．しかしながら，添字を導入することは型定義の記述の簡潔性からも重要であり，かつ添字を導入せずに任意の高さの赤黒木を表現することは不可能であるから，全体としては型の表現力は向上しているといえる．

4.4　具体的な型検査の手順

ここでは，例として以下のルールの型検査を行う．

```
b(S,leaf,R) :- b(S,r(leaf,leaf),R)
```

これは赤黒木へのノードの挿入のうち最も単純な（木の回転を必要としない）例である．まず，生成規則 `tl` を逆向きに使って，左辺の `leaf` を `tree<0>`に戻す．

```
b(S,tree<0>,R) :- b(S,r(leaf,leaf),R)
```

続いて，自由リンク `S` の先に `tree<0>`を補いつつ，生成規則 `tb` を逆向きに使って，左辺全体を `tree<1>`に戻す．

```
tree<1>(R) :- b(tree<0>,r(leaf,leaf),R)
```

このルールは reducible である．実際，次の手順で左

辺から右辺へと遷移できる．

tree<1>(R)
$\xrightarrow{\texttt{tb}}$ b(tree<0>,tree<0>,R)
$\xrightarrow{\texttt{tr}}$ b(tree<0>,r(rbtree<0>,rbtree<0>),R)
$\xrightarrow{\texttt{bl}}$ b(tree<0>,r(leaf,rbtree<0>),R)
$\xrightarrow{\texttt{bl}}$ b(tree<0>,r(leaf,leaf),R)

また，生成規則 tb の代わりに bb を逆向きに使うと，次を得る．

```
btree<1>(R) :- b(tree<0>,r(leaf,leaf),R)
```

これもまた reducible である．実際の遷移経路は，先ほどのものとほとんど同一であるので割愛する．これ以外に初期状態から bl を使って戻ることもできるが，その先は行き詰まり状態であって左辺が開始記号となることはないので，この経路は考えなくて良い．以上により，ルールの型保存性が示された．

補うグラフに添字付き非終端記号が含まれる場合は，その添字に任意性があるため上記のように単純ではない．例えば，次のルールを考える．

```
b(A,B,R) :- b(B,A,R)
```

これは単に黒ノードの左右の部分木を入れ替えるルールである．あまり実用的でなくかつ型保存性が直観的には明らかであるが，この型検査にあたっては添字の取扱いが難しい．まず自由リンク A と B の先に黒高さ n の tree を補いつつ，生成規則 bb を逆向きに使い，次を得る．

```
btree<n + 1>(R) :- b(tree<n>,tree<n>,R)
```

このルールは生成規則 bb により reducible である．同様に，初期状態から生成規則 tb を使った場合以下を得るが，これも生成規則 tb により reducible である．

```
tree<n + 1>(R) :- b(tree<n>,tree<n>,R)
```

以上によりルールの型保存性が検証できた．

4.5 インデックス型の実装

以上のような，具体値が必ずしも定まっていないが制約条件が存在するような数を含むグラフに対する操作は，一見すると自然なように思われる．しかし，こうした不定数を記号的に扱う機能は現在の LMNtal の組み込み機能には存在しない．そのため実装にあたっては，従来の型検査実装に用いていた第一級書換え規則 [10] を拡張し，数値に関する記号制約を持っておくことができるようにした．具体的には，ルールの左辺と右辺を表す膜に加えて，制約ストアを表す膜を追加した．左辺・右辺において添字が出現する部分はそれぞれ制約ストア膜内の対応する膜とリンクで接続し，それらの膜の間に，添字が満たすべき制約条件を表すアトムを橋渡しする形で制約を表現した．例えば，前節の型検査の例において登場した

btree<$n + 1$>(R) :- b(tree<n>,tree<n>,R)

という書換え規則は，実装上は図 12 のように表現される． 図中で下側にある角丸四角形が制約ストアを表す膜である．その膜の中に入れ子になっている膜が 3 つあり，それぞれが非終端記号の添字を表している．便宜上，図中左から順に x, y, z と呼ぶことにすると，右側にある eq/2 アトムは $y = z$ という制約を表しており，左側にある eq/2, +/3, 1/1 アトムは $x = y + 1$ という制約を表している．

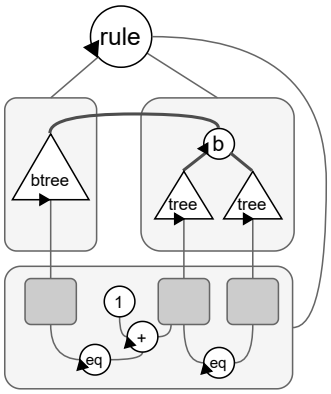

図 12　拡張された第一級書換え規則による検査対象ルールと制約ストアの表現

このように，制約をグラフとして管理することで，LMNtal が元来持っている構造合同による状態空間圧縮の恩恵に与ることができ，全く同一の制約をもつ全く同一のルールについては 2 度以上探索を繰り返さずに済むという利点がある．

```
defshape grid/0 {
  r1@@ grid :- bgn(L1,L1).
  r2@@ bgn(X,Y) :- f(e,L,X), bgn(L,Y).
  r3@@ bgn(X,Y) :- mid(Z,Z,X,Y,e).
  r4@@ mid(Z,W,f(H,X),Y,R)
    :- f(L1,L2,W), n(H,mid(Z,L1,X,Y),L2,R).
  r5a@@ mid(Z,W,X,X,R)
     :- e(R), mid(X,X,Z,W,e).
  r5b@@ mid(Z,W,X,X,R)
     :- e(R), end(Z,W).
  r6@@ end(f(H,X),Y) :- e(H), end(X,Y).
  r7@@ end(f(H,X),X) :- e(H).
} nonterminal {
  bgn/2, mid/5, end/4, f/3
}
```

図 13　グリッドグラフの文脈自由でない型定義

5　インデックス型における添字の拡張

5.1　背景：$m \times n$ グリッドグラフ

これまでの拡張により，非終端記号に添字として非負整数の情報を持たせられるようになり，赤黒木などの数値制約を伴う型を扱えるようになった．しかし，ラムダ式のグラフによる表現やグリッドグラフなどを型として表現するためには，これまでの拡張だけでは不十分である．

まず，大きさ $m \times n$ のグリッドグラフについて見ていく．m が一定であり n のみが動く場合は，一列分にあたる m 個のアトムを同時に生成するような規則を書けば文脈自由の範囲で生成可能である．m と n が逆の場合も同様である．一方で，m, n ともに動く場合については，文脈自由な生成規則では素朴に記述することが難しい．このような型は，文脈自由ではない生成規則を許すと図 13 のように記述できる．

この型による生成は以下のように進む．また，生成過程の例を図 14 に示す．

1. ルール `r1` により開始記号の `grid`/0 が自己ループ（空の deque）を伴う非終端記号 `bgn`/2 に変化する．
2. `r2` を 1 回以上適用することにより終端記号 `e`/1 の列からなる左の端が生成される．この際，左の端につながるリンクは非終端記号 `f`/3 からなる deque（`bgn`/2 の第 1 引数と第 2 引数）によって管理する．
3. `r3` により非終端記号が `mid`/5 に変化し，従前の deque はそのまま第 3 引数と第 4 引数に受け継いだ上で，同時に第 1 引数と第 2 引数にも空の deque をつくり，上の端を表す `e`/1 を 1 つ生成する．
4. `r4` により 2 つ目の deque から（左側へとつながる）リンクを 1 つ取り出して，グリッドグラフのノード（`n`/4）を一つ生成して，新しい左側へのリンクを 1 つ目の deque の末尾へと返却する手順を 1 回以上（縦の幅の分だけ）繰り返す．
5. 2 つ目の deque が空になったら，以下のいずれかを非決定的に選択し行う：
   (a) `r5a` により 1 つ目の deque を 2 つ目に移し替え，下の端を表す `e`/1 を生成して一列分の生成を終えるとともに，次の列の上の端を表す `e`/1 を生成し，手順 4 に戻る．
   (b) `r5b` により下の端を表す `e`/1 を生成するとともに，非終端記号が `end`/2 に変化し，従前の 1 つ目の deque を引き継ぐ．手順 6 に進む．
6. `r6` を 1 回以上（縦の幅の分だけ）適用することにより，右の端を表す `e`/1 の列を生成し，deque が空になったら `r7` により生成を終わる．

この型において，`f`/3 からなる deque は単にリンクを束ねるためのものであるので，`f`/3 は独立した非終端記号というよりは，`bgn`/2 や `mid`/5 のような他の非終端記号の付属物であるとみなすことができる．このように考えると，前掲の `grid`/0 型の定義は形の上では文脈自由な型と捉えることができる．

5.2　リンクを要素とする deque の導入

そこで，インデックス型における添字について，リンクを要素とする deque（差分リスト）を取れるよう拡張することを考える．以下，4.2 節において導入したプロセス文脈のうち，名前が `$D` から始まるものをリンクの deque として扱う．特に，`$Demp` は空の deque を表す予約語とする．加えて，deque に関する以下の制約をガードに記述してよいこととする．

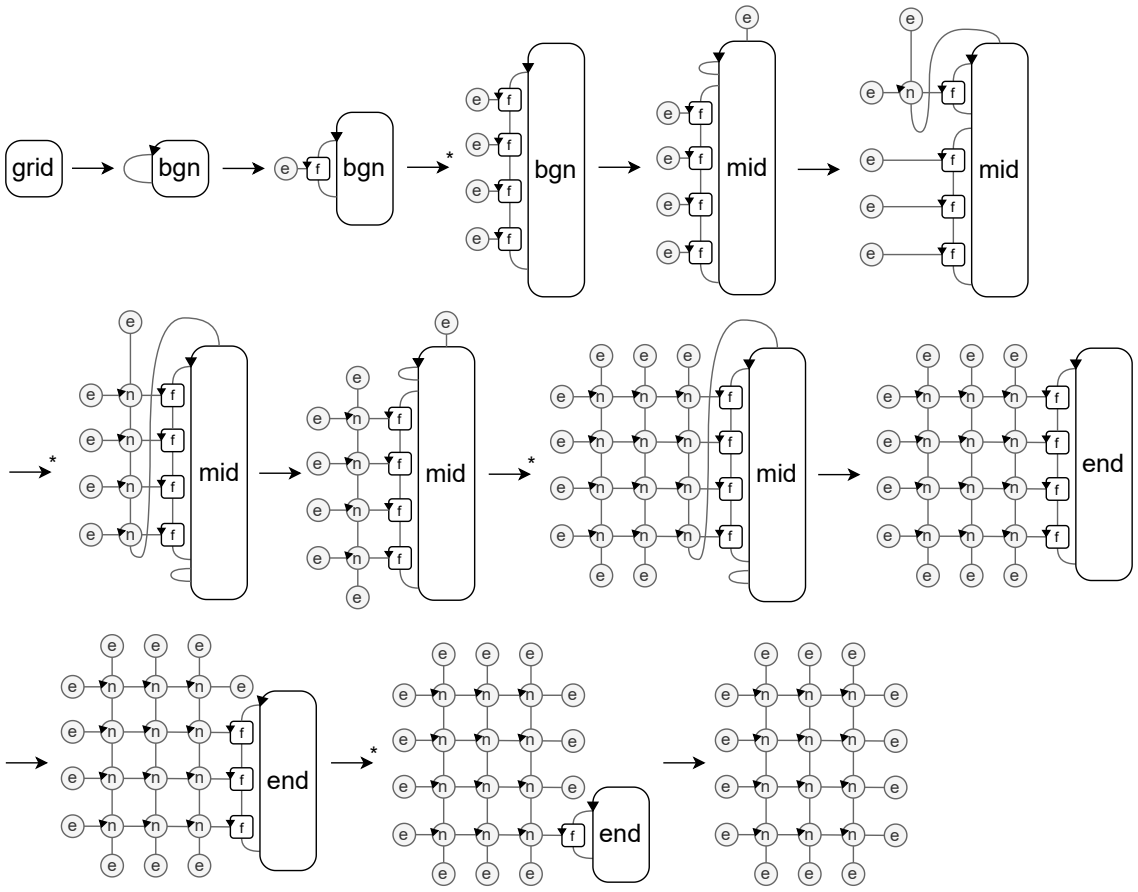

図 14　グリッドグラフの生成過程の例

`push(X,$D1,$D2)`

deque `$D1` の末尾に要素としてリンク `X` を追加した結果得られる deque が `$D2` である

`shift(X,$D1,$D2)`

deque `$D1` の先頭要素のリンク `X` を削除した結果得られる deque が `$D2` である

一般に deque に許される操作は他に `pop`, `unshift` などがあり，これらも上記と同様に導入できるが，本論文では割愛する．

次に，構文上ルールに課せられる制約条件について述べる．まず，`$Demp` 以外の deque は両辺においてちょうど 1 回ずつ出現する必要がある．ただしこの際，ガードにおいて `push` や `shift` の第 2 引数に出現する deque は右辺，第 3 引数に出現する deque は左辺に出現しているものとみなす．

以上の拡張のもとで，グリッドグラフの型は図 15 のように記述できる．

6　ハイパーグラフへの拡張

関数型プログラミング言語に対応する計算モデルであるラムダ計算においては，α 変換可能な名前により変数の束縛を表現する方式が一般的に用いられている．一方で，より直截的な表現方法として，ラムダ式をグラフとして表現し，束縛を辺の接続により表す手法が存在する [12] [16]．特に，[16] の方法では多対多接続を許す**超辺**により束縛を表現している．これは，1 対 1 接続の通常辺をベースとして補助的な頂点により分配を行う [12] などの他の手法と比べても，より直截的なラムダ式の表現であるといえる．この節では，[16] の方法でグラフとして表現されたラムダ

```
defshape grid/0 {
  r1@@  grid :- bgn<$Demp>.
  r2@@  bgn<$D1> :- push(X,$D1,$D2) | e(X), bgn<$D2>.
  r3@@  bgn<$D1> :- mid<$Demp,$D1>(e).
  r4@@  mid<$D1,$D2> :- push(X,$D1,$D3),shift(Y,$D2,$D4) | n(X,mid<$D3,$D4>,Y,R).
  r5a@@ mid<$D,$Demp>(R) :- e(R), mid<$Demp,$D>(e).
  r5b@@ mid<$D,$Demp>(R) :- e(R), end<$D>.
  r6@@  end<$D1> :- shift(H,$D1,$D2) | e(H), end<$D2>.
  r7@@  end<$D> :- shift(H,$D,$Demp) | e(H).
} nonterminal { grid/0, bgn/1/0, mid/2/1, end/1/0 }
```

図 15　拡張されたインデックス型を用いたグリッドグラフの型定義

式を型として表現することを目指す．

ハイパーリンクと呼ばれる超辺の出現を許す LMNtal の拡張を **HyperLMNtal**[13] という．言語仕様等の詳細は [13] を参照されたい．

以下，アトムの引数にハイパーリンクの出現を許す．ハイパーリンクは，プログラム中では `!X` のように，`!`とそれに続く英大文字から始まる名前により表現されるものとする．例えば，

```
lam(!F,lam(!X,
  app(var(!F),app(var(!F),var(!X)))),R)
```

は図 16 のようなグラフを意味する．

また，このグラフはラムダ式 $\lambda fx.f(fx)$ を表す．`lam`/3 は λ 抽象を表し，第 1 引数はこの λ が束縛する変数を表すハイパーリンク，第 2 引数は抽象の本体である．`app`/3 は関数適用を表し，第 1 引数が関数，第 2 引数が引数である．また，ここでは後ほどの型定義の都合上，（ラムダ計算における）変数の出現を表すハイパーリンクの直前に `var`/2 アトムを挟んでいる．

HyperLMNtal においては，型検査を行う上で重要となる，逆実行と反転ルールの関係を表した定理 3.1 が素朴には成り立たない．というより，そもそも HyperLMNtal のルールはハイパーリンクの生成及び破棄を行えるため，対称ではない．そのため，型検査の過程における反転ルール生成の際，ハイパーリンクを生成する生成規則については，生成された（右辺にのみ存在する）ハイパーリンクの要素数が，元のルールの右辺に出現する回数とちょうど一致することを検査する必要がある．例えば，

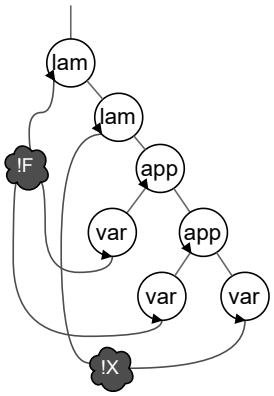

図 16　ハイパーグラフの例

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

の反転ルールは，

```
b(!Y,!Y,!X) :- num(!Y)=:=2 | a(!X).
```

とする．ここで，`num(!Y)=:=2` はグラフ全体における `!Y` のハイパーリンクの出現数がちょうど 2 であることを表す HyperLMNtal のガード制約である．また，生成規則においてはハイパーリンクの破棄を許さないこととする．すなわち，左辺に出現するハイパーリンクは必ず右辺においても出現しなくてはならない．

また，プロセス文脈のうち，名前が `$HD` から始まるものをハイパーリンクの deque として扱う．特に，`$HDemp` は空の deque を表す予約語とする．加えて，前節と同様の deque に関する制約をガードに記述してよいこととする．以上の拡張を認めた上で，ハイパーグラフとして表現されたラムダ式の型は図 17 のように表せる．なお，開始記号 `term`/1/1 の添字は自

```
defshape term/1/1 {
  t1@@ term<$HD>(R) :- lam_<$HD>(!X,R).
  l1@@ lam_<$HD1>(!X,R) :- push(!X,$HD1,$HD2) | lam_<$HD2>(!X,R).
  l2@@ lam_<$HD>(!X,R) :- lam(!X,term<$HD>,R).
  t2@@ term<$HD>(R) :- app_<$HDemp,$HDemp,$HD>(R).
  a1@@ app_<$HD1,$HD2,$HD3>(R) :-
    shift(!X,$HD3,$HD5), push(!X,$HD1,$HD4) | app_<$HD4,$HD2,$HD5>(R).
  a2@@ app_<$HD1,$HD2,$HD3>(R) :-
    shift(!X,$HD3,$HD5), push(!X,$HD2,$HD4) | app_<$HD1,$HD4,$HD5>(R).
  a3@@ app_<$HD1,$HD2,$HDemp>(R) :- app(term<$HD1>,term<$HD2>,R).
  t3@@ term<$HD>(R) :- shift(!X,$HD,$HDemp) | var(!X,R).
  t4@@ term<$HD1>(R) :- shift(!X,$HD1,$HD2), push(!X,$HD2,$HD3) | term<$HD3>(R).
} nonterminal { term/1/1, lam_/1/2, app_/3/1 }
```

図 17　拡張されたインデックス型を用いたラムダ式のグラフ表現の型定義

由変数を表すハイパーリンクからなる deque である．

以下，この型による生成の流れを説明する．まず，以下のいずれかを非決定的に選択し行う：

1. ルール `t1` により，非終端記号 `term`/1/1 が `lam_`/1/2 に変化し，その第 1 引数に新しい束縛変数を表すハイパーリンク `!X` を生成する．以降，ルール `l1` を繰り返し適用することにより，0 回以上 `!X` を deque に push した後，ルール `l2` により λ 抽象を表す `lam`/3 アトムを生成した上で抽象の本体に deque を引き渡す．
2. ルール `t2` により，非終端記号 `term`/1/1 が `app_`/3/1 に変化する．以降，ルール `a1`, `a2` により deque の中身を非決定的に 2 つの deque に分配し，分配が終わったらルール `a3` により関数適用を表す `app`/3 アトムを生成した上で 2 つの deque をそれぞれ関数側と引数側へと引き渡す．
3. ルール `t3` により，deque がシングルトンであることを確認した上で，変数を表す `var`/2 アトムを生成する．
4. ルール `t4` により，deque から一つ要素を shift してそのまま push する．すなわち，単に deque を rotate する．

特に，ルール `l1` の繰り返し適用によって先に束縛変数の使用回数を決め打ってしまうのが重要なポイントである．束縛された変数を各々一つだけ deque に保存して生成を進める場合，最終的に使用しなかった変数を破棄する必要があるが，ハイパーリンクの破棄は認めていないため，そのような生成規則は記述することができない．そのため，前もって使用する回数分だけハイパーリンクをコピーしておくことにしている．

この方式では，生成過程において束縛変数を保存した数と使用した数の辻褄が合わず，行き詰まりとなるケースが多発するため，一見すると効率的でないが，型検査を行う際の逆実行の効率だけを考えれば特に問題とはならない．

7　関連研究

本節では，まずグラフを対象とする既存の型検査手法を紹介する．Graph types[7] は正規表現を基にしたグラフの型検査手法である．また，Structured Gamma[5] は文脈自由文法を基にした，生成規則によって型を定義する型検査手法である．Shape Types[4] はこの Structured Gamma のサブセットであり，型検査の完全性が満たされるように型定義の範囲を制限したものである．本論文で扱っている LMNtal ShapeType は上記 Shape Types 及び Structured Gamma の技法をベースとして，LMNtal の書換え規則の型検査を行う手法である．以上の手法は，いずれも「グラフの書換え操作を 1 ステップ進めても，構造が破壊されない」ことの検査・保証を目的としたものである．しかし，その対象となる型の表現力には文脈自由性による制限（定義 3.6 参照）があり，完全二分木や赤黒木などの数値制約の入った型を表現することはできていなかった．

本論文においては赤黒木のような数値制約を含む型の表現にインデックス型を用いたが，関数型言語をベースとした型体系では，型に対して述語による数値制約を掛ける手法として篩型（refinement types [14]）が知られている．また，この篩型を述語抽象化によって自動的に推論する手法として Liquid Types [9] が存在する．他のアプローチとして，赤黒木の型を高階型などの機能を用いて表現する研究 [6] もある．

また，ハイパーグラフにおける超辺 †4 を文脈自由文法により書き換えることでグラフの言語を定める Hyperedge Replacement Grammar (HRG) [2] の分野においては，文脈自由より広い文法を表現できるようにする Contextual HRG という拡張が提案されている [3]．この拡張は，生成規則の左辺に，書換え対象となる超辺に加えて，その超辺に接続されていない頂点を記述できるようにすることで，任意のラベルへとジャンプする goto 文を許したコントロールフローグラフや，任意のハイパーグラフを生成する文法などを記述可能にしている．一方で，Contextual HRG では生成規則左辺に出現する頂点に（頂点の名前以外の）制約をかけることが困難であるため，本論文において例題として示したラムダ式のように，階層的な構造をなすハイパーグラフを生成する文法は記述が難しいと考えられる．

8 まとめと今後の課題

本論文では，形式文法に基づいてグラフの型定義・検査を行う枠組みである LMNtal ShapeType に対して，添字として非負整数や，辺・超辺を要素とするリストを伴う非終端記号の出現を許す拡張を施すことで，赤黒木を始めとする数値制約付きのデータ構造や，グリッドグラフ・ラムダ式のグラフ表現といった複雑なグラフ型を表現可能にした．

今後の課題としては，辺や超辺を要素とするリストが添字に出現する型に対応したルールの型検査アルゴリズムの再定式化が挙げられる．特に，ラムダ式のグラフ表現において，β 簡約を表す書換え規則 [16] の型検査を行うためには，検査対象ルールがガードをもつことを許す必要がある．本論文ではインデックス型を通して，ガードを持つ生成規則に関して議論してきたが，検査対象ルール，すなわち型付けの対象言語がガードを持つ場合については依然定式化も議論も十分でない．

謝辞 本研究の一部は，早稲田大学特定課題研究費 (2022C-421, 2022C-435, 2022Q-006) の補助を得て実施した．

†4 HRG では名前を持つ超辺が書換えの対象であり，HyperLMNtal では名前を持つアトム（頂点）が書換えの対象である．このことから，HRG における超辺は HyperLMNtal におけるアトムと，HRG における頂点は HyperLMNtal におけるハイパーリンクと，それぞれ対応づけられる．

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] Drewes, F., Kreowski, H.-J., and Habel, A.: *Hyperedge Replacement Graph Grammars*, World Scientific, 1997, pp. 95–162.
[3] Drewes, F., Hoffmann, B., and Minas, M.: Contextual Hyperedge Replacement, *Proc. AGTIVE'11*, Springer, 2011, pp. 182–197.
[4] Fradet, P. and Métayer, D. L.: Shape types, *Proc. POPL'97*, ACM, 1997, pp. 27–39.
[5] Fradet, P. and Métayer, D. L.: Structured Gamma, *Science of Computer Programming*, Vol. 31, No. 2(1998), pp. 263–289.
[6] KAHRS, S.: Red-black trees with types, *Journal of Functional Programming*, Vol. 11, No. 4(2001), pp. 425–432.
[7] Klarlund, N. and Schwartzbach, M. I.: Graph Types, *Proc. POPL'93*, 1993, pp. 196–205.
[8] Pugh, W.: Skip lists: A probabilistic alternative to balanced trees, *Commun. ACM*, Vol. 33, No. 6(1990), pp. 668–676.
[9] Rondon, P. M., Kawaguci, M., and Jhala, R.: Liquid Types, *Proc. PLDI 2008*, ACM, 2008, pp. 159–169.
[10] Tsunekawa, Y., Tomioka, T., and Ueda, K.: Implementation of LMNtal Model Checkers: A Metaprogramming Approach, *Journal of Object Technology*, Vol. 17, No. 1(2018).
[11] Ueda, K.: Encoding the Pure Lambda Calculus into Hierarchical Graph Rewriting, *Proc. RTA2008*, Voronkov, A.(ed.), LNCS, Vol. 5117, Springer, 2008, pp. 392–408.
[12] 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.
[13] Ueda, K. and Ogawa, S.: HyperLMNtal: An Ex-

tension of a Hierarchical Graph Rewriting Model, *KI - Künstliche Intelligenz*, Vol. 26, No. 1(2012), pp. 27–36.

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

[15] Yamamoto, N. and Ueda, K.: Engineering Grammar-based Type Checking for Graph Rewriting Languages, *Proc. Twelfth International Workshop on Graph Computation Models (GCM 2021)*, June 2021, pp. 93–114.

[16] Yasen, A. and Ueda, K.: Revisiting Graph Types in HyperLMNtal: A Modeling Language for Hypergraph Rewriting, *IEEE Access*, Vol. 9(2021), pp. 133449–133460.

[17] 吉元佑介, 上田和紀: グラフ書換え系における静的グラフ型検査, 日本ソフトウェア科学会第 *32* 回大会 *(2015* 年度*)* 講演論文集, 2015.

[18] 山本直輝, 上田和紀: グラフ書換え言語における数値制約を伴う型の静的型検査, 日本ソフトウェア科学会第 *37* 回大会 *(2020* 年度*)* 講演論文集, 2020.

[19] 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.

[20] 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.

[21] 石川力, 堀泰祐, 村山敬, 岡部亮, 上田和紀: 軽量な LMNtal 実行時処理系 SLIM の設計と実装, 情報処理学会第 *70* 回全国大会講演論文集, (2008), pp. 153–154.

[22] 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.