グラフ書換え言語におけるグラフ操作の静的型検査

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を併せ持つ言語である．また，関数的アトムというデザインパターンを用いて，型の変換を伴う演算をも記述できる．一方で，グラフ書換え言語ではリストや二分木よりも複雑な構造を扱えるため，静的な型体系の定式化は自明でない．グラフ書換えの前後においてグラフの構造が保たれることを検査する静的型体系の先行研究として，Structured Gamma が存在する．そこで本研究では，Structured Gamma を基に，LMNtal に静的型を導入することを考える．本論文では，Structured Gamma の型検査手法を用いて，関数的アトムの入力と出力の型が整合していることを検査する手法を提案する．

A graph rewriting language LMNtal has aspects of both a programming language which expresses calculation by graph rewriting and a modeling language which can handle complicated graph structures. Also, we can use a design pattern called functional atoms in order to describe calculations with type transformation. On the other hand, since graph rewriting languages can handle a broader class of graph structures than lists or trees, formulation of static type system for those languages is not obvious. Structured Gamma is known as a static type system which verifies that graph structures will not be destroyed by graph rewriting. Therefore we shall introduce a static type system based on Structured Gamma for LMNtal. This paper proposes a type checking method for calculations with type transformation, using the type checking algorithm in Structured Gamma.

1　はじめに

ネットワークという構造は，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．これを抽象化・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出来るが，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．このパラダイムに属する言語の例としては，GROOVE [3] や GP [5]，REGREL+ [11] などが挙げられる．

中でも，LMNtal [9] 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．

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

---

* Static Type Checking of Graph Operations in Graph Rewriting Languages.

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.

仕様のレベルに於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．しかし依然として，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，静的型付きの関数型言語である OCaml ではリストの結合を行う append 関数を次のように定義できる．

```
let rec append a b =
  match a with
  | [] -> b
  | h::t -> h::(append t b);;
```

この append 関数は引数に 2 つのリストが与えられることを期待し，それらを結合したリストを返却する．そのため，append 4 [1;2;3];; のようにリスト以外のものを引数に与えると型エラーとなり，処理系がその旨をプログラマに通知する．

同様に，LMNtal においても次のように append アトムを定義できる [†1]．

```
Ret = append([],B) :- Ret = B.
Ret = append([H|T],B)
  :- Ret = [H|append(T,B)].
```

しかし，list = append(4,[1,2,3]). のような不正なプログラムを処理系に与えたとしても，コンパイル時・実行時ともにエラーとはならず，ただ単に実行が進まなくなってしまうだけである．

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体系が提案されている．例えば，文献[6] の型体系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**[8] は，Shape types[1] を基とした型体系である．この型体系では，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，LMNtal 自身によって LMNtal の型検査を行えるという利点がある．ある言語の型体系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研究テーマとして非常に興味深いだけでなく，LMNtal の柔軟な表現力をそのまま型体系に活かすことができるのは大きな強みである．

本論文では LMNtal においてよく用いられるデザインパターンである**関数的アトム**に着目し，LMNtal ShapeType を応用することによって静的に型安全性を検査する手法について述べる．

2 関連研究

本節では，グラフを対象とする既存の型体系を紹介する．

Graph types[4] は正規表現を基にしたグラフの型体系である．また，Structured Gamma[2] 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体系である．Shape Types[1] 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．そして，[8] ではこの Shape Types を基にして LMNtal グラフおよびルールの型検査を行う型体系として LMNtal ShapeType が提案されている．

これらの型体系では，「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・保証を行うことに重点を置いていた．すなわち，「リストに対して挿入操作を行ったあとも必ずリストになっている」や「二分木に対して要素の削除操作を行ったあとも必ず二分木になっている」といった性質は，上述の型体系によって保証できる．

しかしながら，「整数のリストに対して，要素の総和をとる操作を行うと，整数が返ってくる」のように操作の前後で型が変化する場合や，前述の append のように複数の構造を入力にとる場合について対応していなかった．

また，このような型から型への変換を伴う計算には複数ステップを要することも多い．例えば，append の場合は一方のリストの要素数の分だけ計算ステップを要する．従来の型体系では small-step での性質を保証しているために，こうした big-step の性質についてはあまり注目されていなかった．

†1 ここで用いた角括弧によるリスト記法は単なる構文糖衣であり，実際には特別な名前 '.' を持った 3 価アトムの鎖に展開される．

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）

| $[\,RuleName\texttt{@@}\,]\ G$:- G （ルール）

| R,R （分子）

図 1　LMNtal の構文

3　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，本論文では簡単のためこれらを除外したサブセット言語を扱う．また，LMNtal の拡張言語である HyperLMNtal[7] において導入された拡張機能である，超辺を表すハイパーリンクについても扱わない．しかしながら，これらの高度な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．

3.1　構文

LMNtal の構文を図 1 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．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，

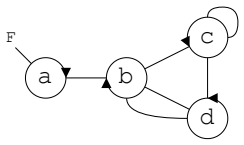

図 2　LMNtal グラフの例

p/m のように書き,「m 価の p アトム」のように読む．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．

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 2 に示すような無向グラフを意味する．ただし，図中において矢印は第 1 引数の場所を指しており，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． グラフ中に同じリンクは高々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．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．

3.2　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．

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

図 3　LMNtal グラフ上の構造合同関係

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

図 4　LMNtal グラフ上の遷移関係

3.2.1　構造合同

グラフ理論における “同型” にあたるものとして，LMNtal にはグラフ間の構造合同関係 “$\equiv$” が存在する．この関係は，図 3 の規則を満たす最小の同値関係として定義される．構造合同な LMNtal プロセスは互いに 0 ステップで変換可能である．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．$\texttt{=}(X,Y)$ 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 $X\texttt{=}Y$ 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．

3.2.2　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 4 に示す遷移規則をみたす最小の二項関係として定義される．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 $G \xrightarrow{R} G'$ とかく．

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，書換えに関する性質を静的に保証する型体系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．

3.3　略記法

便宜上，次の二つの略記法が認められている．

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．

4 LMNtal ShapeType

本節では，[8] で提案された LMNtal を対象とする静的な型体系である 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 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． ◇

4.1 構文

ShapeType の構文を図 5 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（3.3 節）を認める． 可読性の理由から，S と

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

開始記号
S ::= $p(X_1, \ldots, X_m)$

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

非終端記号の有限集合
N ::= A

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

図 5 ShapeType の構文

N の中でファンクタをアトムの形で記述しているが，引数のリンク名に意味はなく，価数が同じであれば名前は何でもよい．また，N の定義は省略することができ，その場合は $N = \{S\}$ とみなす．N の定義に S が含まれていない場合も，暗黙に含むものと考える．

以上のように定義された型を，開始記号 $S = t/m$ のアトム名 t と，適当なリンク名 $L_1, \ldots, L_m$ を用いて $t(L_1, \ldots, L_m)$ 型などと呼ぶことにする．通常の代数的データ型（二分木やリスト等）ではデータ構造の根は一つであるが，LMNtal ShapeType が扱うデータ構造の根は一つとは限らない．そのため，ShapeType として定義される型の根と，型付け対象のグラフの根（自由リンク）とが，どのように対応関係を持つのかを明示するために，根にそれぞれリンク名を持たせている．

ただし，リンク名が重要でないときは開始記号のファンクタを用いて 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も省略して単に t 型と呼ぶ．

4.2 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，$L_1, \ldots, L_m$ だけを自由リンクに持つグラフを $G[L_1, \ldots, L_m]$ 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，Funct(G) と表す．

定義 2. 型 $(t/m, P, N)$ について，

$$t(L_1, \ldots, L_m) \xrightarrow{P}{}^{*} G[L_1, \ldots, L_m]$$

であるとき，かつそのときに限り，$G[L_1, \ldots, L_m]$ は $t(L_1, \ldots, L_m)$ 型によって**生成される**といい，$G[L_1, \ldots, L_m] \lhd t(L_1, \ldots, L_m)$ 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. 型 $(t/m, P, N)$ について，

$$G[L_1, \ldots, L_m] \lhd t(L_1, \ldots, L_m)$$

かつ

$$\mathrm{Funct}(G[L_1, \ldots, L_m]) \cap N = \varnothing$$

であるとき，かつそのときに限り，$G[L_1, \ldots, L_m]$ は $t(L_1, \ldots, L_m)$ 型を**もつ**といい，$G[L_1, \ldots, L_m] : t(L_1, \ldots, L_m)$ 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．

これらの定義において，グラフの自由リンクをわざわざ明示しているのは，前節で述べた通り自由リンクの名前と順序が重要であるためである．例えば，

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

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

4.3 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，ルールの実行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 4. ルール r と型 t/m およびリンク列 $L_1, \ldots, L_m$ について，全ての $G : t(L_1, \ldots, L_m)$ が，

$$G \xrightarrow{r} G' \Rightarrow G' : t(L_1, \ldots, L_m)$$

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． $\diamond$

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．

LMNtal ShapeType においては，生成規則が LMNtal 自身のルールとして記述されていることから，LMNtal を対象言語とするモデル検査器 SLIM [10] を用い，生成規則の逆実行により実際にグラフの型検査・ルールの型検査を行うことが出来る．具体的な検査手法については，文献 [8] を参照されたい．

5 関数的アトム

LMNtal プログラムを書く上で，よく用いられるデザインパターンとして，**関数的アトム**というものがある．これは，関数型言語における関数のように振る舞うアトムである．例えば，以下のような `append`/3 アトムは関数型言語における `append` 関数のように，リストの結合を行う．

```
a1@@ R=append(c(L1),L2)
       :- R=c(append(L1,L2)).
a2@@ R=append(n,L) :- R=L.
```

これらのルールは図 6 に示すように可視化できる．

関数的アトムは，関数と同様に引数に対し入力されたデータに基づいて，適切に出力を返すことが期待される．例えば，`append`/3 アトムは第 1 引数と第 2 引数に入力として 2 本のリストが入ってくることを期待し，それらのリストをルールによって扱い，結合したリストを出力として第 3 引数に向かって返却する（図 7）．

通常の関数型言語における関数とは異なり，関数的アトムが返す型のルートは，一つとは限らない．そのような型を返す関数的アトムの例として，次の `t2l`/3（tree-to-list）がある．

```
t1@@ t2l(n(N,L,R),T,P)
       :- t2l(L,c(N,t2l(R,T)),P).
t2@@ t2l(l,T,P) :- T=P.
```

これらのルールは図 8 に示すように可視化できる．これは，二分木を受け取って，中間順に走査し，差分リストに変換する関数的アトムである（図 9）．

以上のように関数的アトムとは，入力を受け取って出力を返す関数のように振る舞うアトムのことを指す．一方で，通常の関数型言語においては関数が演算の主体であり，データ構造が演算の客体であるのに対し，LMNtal においては演算の主体たるプロセスやメッセージと，客体たるデータを区別することなく，それら全てをアトムという唯一の言語要素によって統一的に表現する．そのため，「関数的アトム」というものが，他のアトム（データアトム）とア・プリオリに類別されているわけではない．

また，LMNtal における演算の主体は，アトムではなくむしろルールであるので，アトムが自身の振る舞いについて責任を負っているわけではない．アトムの振る舞いを規定するのは，ルールである．そして，ルールが発火するか否かはアトムそのものに依存しているというよりは，各アトムのもつファンクタとそれらの間の接続構造に依存している．

図 6　append/3 アトムに関する関数的ルール

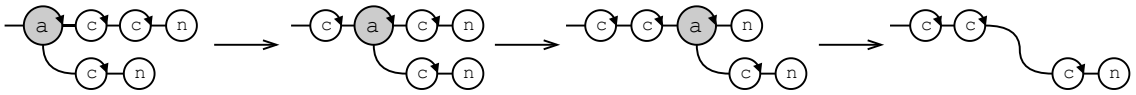

図 7　append/3 アトムによる計算過程

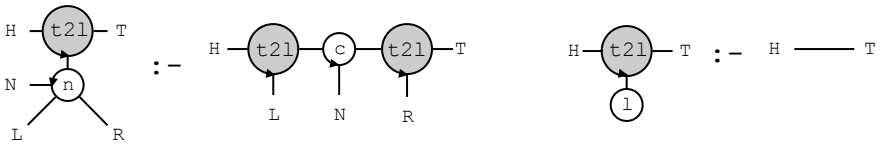

図 8　t21/3 アトムに関する関数的ルール

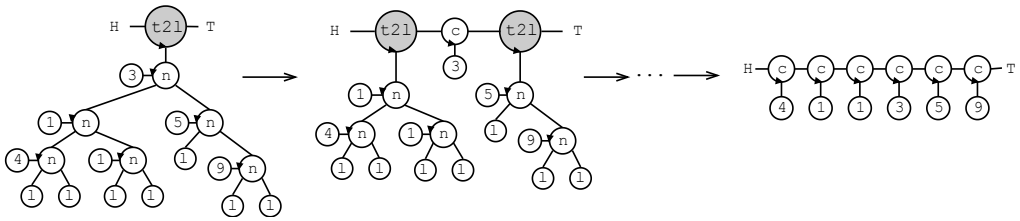

図 9　t21/3 アトムによる計算過程

以上の事実を踏まえると，“関数的” という性質を定義する際には，個々のアトムではなくむしろルールやファンクタに注目すべきである．よって，“関数的” という性質を以下のように定義する．

定義 5. 型 $t_1,\ldots,t_n,T$ と，f/m アトム一つからなるグラフ F および 左辺に f/m アトムをちょうど 1 つもつルールからなるルールセット R について，$t_1,\ldots,t_n$ 型をそれぞれもつグラフ $G_{t_1},\ldots,G_{t_i}$ に対し，

$$F,G_{t_1},\ldots,G_{t_n} \xrightarrow{R}{}^{*} G$$

を満たし，かつ F と同じファンクタのアトムを含まないような G について，$G:T$ を（$G_{t_1},\ldots,G_{t_n}$ のとり方によらず）満たすとき，F は**関数的**であるという．また，これを $t_1,\ldots,t_n \vdash_R F:T$ とかく．

$t_1,\ldots,t_n$ を**入力型**，T を**出力型**，R を F に関する**関数的ルールセット**という．また，F をもつアトムのことを単に関数的アトムという．誤解の恐れがないときは，R を略す．　◇

直観的に言えば，関数的アトム F に入力として $t_1,\ldots,t_n$ 型のグラフを与えて，計算が終了すると T 型のグラフが得られるということを示している．ただし，ここではグラフ中に関数的アトムが残っていないこと（すなわち計算の正常終了）を仮定しているので，計算が行き詰まらないことを保証するわけではない．

この記法を用いて，append アトムと t2l アトムの性質を図 10 のように記述できる．

$$\texttt{list(L1),list(L2)} \vdash_{\{\texttt{a1},\texttt{a2}\}} \texttt{append(L1,L2,R)} : \texttt{list(R)}$$
$$\texttt{tree(T)} \vdash_{\{\texttt{t1},\texttt{t2}\}} \texttt{t2l(T,X,Y)} : \texttt{dlist(X,Y)}$$

図 10 append アトムと t2l アトムの性質（ここでは，二分木の型を tree，差分リストの型を dlist と表記している）

6 関数的アトムの型検査

与えられたファンクタが関数的であるか否かを，LMNtal ShapeType におけるルールの型検査を用いて検証することを考える．関数的という性質は上述のように大ステップ的に定義されるが，ルールの型検査によって検査可能であるのは「1 ステップのルール適用」のみである．そこで，以下では図 7 や図 9 に示したような関数的アトムによる計算過程を小ステップ的に解釈していく．

以下では,「関数的アトムに入力型のグラフが繋がったもの」を**計算途中グラフ**と呼ぶ．例えば，図 7 の最左のグラフは関数的アトム append に入力型 list のグラフが 2 つ繋がった形をしているので，計算途中グラフそのものといえる．これに 1 ステップルールを適用したグラフ（図 7 の左から 2 つ目）は c アトムの先に計算途中グラフが繋がった形をしている．続いて図 7 の左から 3 つ目のグラフは，c アトムが 2 つ繋がった先に計算途中グラフが繋がった形をしている．

同様に，図 9 の最左のグラフは全体が計算途中グラフになっており，次のグラフは c アトムの前後に計算途中グラフが繋がった形をしている．

これらのグラフは，いずれも計算途中グラフが最終的に出力型のグラフになると仮定すると，全体として出力型のグラフになっている．つまり，図 7 の c アトムが 0 個以上繋がった先に計算途中グラフが繋がった形は，計算途中グラフを list 型の任意のグラフで置き換えてやると，全体として list 型のグラフとなる．また，図 9 の左から 2 つ目のグラフは c アトムの前後に計算途中グラフが繋がった形をしているので，計算途中グラフを差分リストで置き換えてやると，全体として差分リストになる．

関数的ファンクタ F をもつアトムを（0 個以上）含むグラフ G に対し，F に関する関数的ルールセット R を 1 ステップ適用すると，グラフ G' を得たとする．このとき G' は，

1. 関数的アトムが 1 個以上残っている（計算の途中である）
2. 関数的アトムが残っておらず，出力型をもつグラフになっている（計算が終了している）

のいずれかであるが，この 2 つの性質は上記の議論により,「計算途中グラフが出力型をもつと仮定すると，G' は出力型をもつ」という性質によって統一的に表すことが出来る．

「計算途中グラフが出力型をもつと仮定する」というのは，具体的には出力型の開始記号から計算途中グラフを生成するような生成規則を追加することで実現できる．すなわち，append の場合であれば

```
list(R) :- append(L1,L2,R),list(L1),list(L2)
```

という生成規則を list 型の定義に，t2l の場合であれば

```
dlist(X,Y) :- t2l(T,X,Y),tree(T)
```

という生成規則を dlist 型の定義にそれぞれ加えることを指す．

また，この仮定の上で考えると，計算の開始時のグラフ（図 7 および図 9 の最左）は全体が計算途中グラフとなっていることから，これも出力型のグラフになっているといえる．つまり，関数的ルールセット R が，生成規則の追加によって拡張された出力型を保存するならば，F は関数的であると言える．

以上の議論を定理の形にまとめると次のようになる．ただし，S_t, P_t, N_t は型 t の開始記号と生成規則の集合と非終端記号の集合をそれぞれ表すとする．

定理 1. 生成規則の集合 $P = P_T \cup P_{t_1} \cup \cdots \cup P_{t_n} \cup \{T \texttt{ :- } F\texttt{,}t_1\texttt{,}\ldots\texttt{,}t_n\}$ と非終端記号の集合 $N = N_T \cup N_{t_1} \cup \cdots \cup N_{t_n}$ について，すべてのルール $r \in R$ が，型 (S_T, P, N) を保存するとき，$t_1, \ldots, t_n \vdash_R F : T$ が成り立つ． ◇

ただし，それぞれの型の非終端記号には重複がないように予め α 変換を施しておくものとする．

7 まとめと今後の課題

本論文では，関数的アトムというデザインパターンに着目し，複数ステップを要し，また型の変換を伴うようなグラフ操作について型検査を行う手法について述べた．

今後の課題としては，入力や出力のグラフが実質的に無限である場合に対応することが挙げられる．今回扱ったグラフはいずれも有限であることを暗黙に仮定しているが，6 節の議論は，グラフが実質的に無限である場合にも適用可能である．例えば，無限リスト（ストリーム）を出力するような関数的アトムや，無限のメッセージ列を受け取るサーバのような関数的アトムといったものが考えられる．実際のところ，LMNtal はプロセス計算を基に設計されていることから，こうした無限の性質をもったモデルを容易に記述出来る．そのため，無限グラフに対応した型体系を用意しておくことは重要である．

謝辞 本研究の一部は，科学研究費基盤研究 (B) 18H03223 の助成を受けて実施した．

参 考 文 献

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

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

[3] Ghamarian, A., de Mol, M., Rensink, A., Zambon, E., and Zimakova, M.: Modelling and analysis using GROOVE, *STTT*, Vol. 14, No. 1(2012), pp. 15–40.

[4] Klarlund, N. and Schwartzbach, M. I.: Graph Types, *Proc. POPL'93*, 1993, pp. 196–205.

[5] Manning, G. and Plump, D.: The GP programming system, *Proc. Graph Transformation and Visual Modelling Techniques (GT-VMT 2008), volume 10 of Electronic Communications of the EASST*, 2008.

[6] 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.

[7] Ueda, K. and Ogawa, S.: HyperLMNtal: An Extension of a Hierarchical Graph Rewriting Model, *KI - Künstliche Intelligenz*, Vol. 26, No. 1(2012), pp. 27–36.

[8] 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.

[9] 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.

[10] 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.

[11] 東達軌, 武田正之: 動的な階層構造に基づくグラフ書き換え言語 REGREL+, 第 *10* 回情報科学技術フォーラム *(FIT2011)* 講演論文集（第 *1* 分冊）, 2011, pp. 159–166.