A-032

グラフ書き換え言語 LMNtal プログラムのトークンに基づく書き換え制御
Token-based rewrite control of graph rewriting programs in LMNtal

岡村 朋佳[1)]　山本 直輝[1)]　上田 和紀[1)]
Tomoka Okamura　Naoki Yamamoto　Kazunori Ueda

1　はじめに

本研究は，グラフ書換え言語 LMNtal [1] における書換え制御手法としてトークン機構を導入することにより，非決定的な書換えの制御を実現し，効率的かつ再現性のある実行を可能にすることを目的とする．

グラフは実世界に存在する多様な構造を自然に表現できるモデリング手段であり，高い表現力を持つ．グラフ構造の書換えによって計算を進行するグラフ書換えは，動的な構造変化を含むシステムのモデリングに適しており，LMNtal では階層構造や非決定性，並列性のある書換えも扱うことができる．しかし，LMNtal では書換え対象が動的に変化するため，計算の進行状況や実行効率を予測・制御することが困難である．また，複数の書換え可能な構造が同時に存在する場合において，どの構造を優先的に書き換えるかといった制御構造を持たない．

本論文では，グラフ構造上を巡回する制御機構であるトークンを導入し，トークンが存在する箇所においてのみ書換えを可能とする．これにより，書換えの対象となる構造を可視化・決定しやすくし，実行戦略の動的な構造制御を目指す．

2　LMNtal

LMNtal は，規則に基づいてグラフを書き換えていくことで計算を進める言語である．グラフはアトムとその接続関係を表すリンクで構成される．アトムは順序のある 0 個以上のリンクを持ち，リンクは 2 つのアトムが接続関係にあることを表す．LMNtal におけるグラフ書換え規則は，左辺と一致する部分グラフを右辺に置き換えるという形で記述される (構文と意味論については文献 [1] を参照)．プログラム例を以下に示す．

```
a(L1),b(L1,L2),b(L2,L3),a(L3).
b(X,Y) :- c(X,Y).
```

プログラムは初期状態のグラフ構造と書換え規則によって構成される．図 1 左のグラフが初期状態を表しており，アトム a，b がリンク接続されている．このとき，規則が適用可能なグラフ構造は 2 箇所存在する．

LMNtal は非決定的なモデルを記述することができる言語である．LMNtal の実行時処理系 SLIM [2] には，いずれか一つの書換えパスを実行する通常実行に加え，すべての実行経路を探索し，状態空間全体を構築する非決定実行が実装されている．

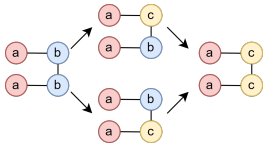

図 1　非決定的な書換えの例

1)　早稲田大学 Waseda University

2.1　例題：二分探索木

二分探索木は，最大 2 つの子要素を持ち，左の子は親より小さく，右の子は親より大きいという制約を持つ木構造である．以下に初期状態である二分探索木のグラフ構造と，書換え規則の LMNtal における実装例を示す．挿入操作（ins）と探索操作（sea）を表すアトムは木構造上を移動し，目的の位置を見つけたら操作を行う．

```
head=sea(10,ins(3,ins(1,ins(5,INP)))),
INP=n(4,n(0,z,n(2,z,z)),n(8,n(6,z,z),n(10,z,z))).

B = ins(D,z) :- B = n(D,z,z).
B = ins(D,n(F, L, R)) :- D < F |
      B = n(F, ins(D,L), R).
B = ins(D,n(F, L, R)) :- D > F |
      B = n(F, L, ins(D,R)).
B = ins(D,n(F, L, R)) :- D = F |
      B = n(F, L, R),  fail(ins,D).

B = sea(D,z) :- B=z,fail(sea,D).
B = sea(D,n(F, L, R)) :- D < F |
      B = n(F, sea(D,L), R).
B = sea(D,n(F, L, R)) :- D > F |
      B = n(F, L, sea(D,R)).
B = sea(D,n(F, L, R)) :- D = F |
      B = n(F, L, R), succ(sea,D).
```

各規則の左辺には，書換え対象となる木構造の一部のほかに，ins のようなアトムが含まれている．ins アトムは，挿入対象の値を持ってデータ構造上を移動し，挿入位置を見つけたら消滅する．よって，挿入にかかわる規則の左辺は必ず ins アトムを含む．探索操作を表す sea アトムについても同様である．こうした関数のようなふるまいをするアトムを，本論文では**関数**と呼ぶ．

3　トークンを使用した LMNtal プログラムのグラフ書換え制御

LMNtal プログラムに対する書換えの制御手法として，グラフ構造上を巡回するトークン（token）を導入する．グラフ中のトークンが存在する箇所でのみ書換え規則を適用可能とする制御手段であり，一意な書換え順序の戦略的制御を実現することが可能となる．

3.1　トークンによる実行制御

グラフ書換えをトークンによって制御する手法として，Interaction Nets において，グラフ表現されたラムダ計算のトークンを用いた戦略制御の手法が提案されている [3]．これを応用して，LMNtal プログラムにおいてもトークンを用いた書換え制御が可能である．

3.2　LMNtal プログラムのトークン制御の概要

LMNtal では，複数の部分グラフが同時に書換え可能な時，それらをどの順序で適用するかの戦略を指定する手段が存在しない．そこで，トークンを用いて構造内の位置に基づく書換えの実行順序を制御する．具体的には，トークンが接続されている部分グラフのみを書換え可能にし，トークンが各書換え箇所に到達する順番をコントロールすることで，書換えの順序を調整する．

よって，与えられた LMNtal プログラムに対してトークン制御を適用するには，次の手順が必要となる：

1. 書換え対象となるグラフにおけるトークンの巡回規則を決定する
2. 書換え規則をトークンを付加した形に変換し，実行をトークンが接続されている箇所に限定する
3. トークンの巡回規則に従って，トークンの遷移規則を追加する

3.3 トークンの巡回規則の決定

トークンは，書換え対象となる連結なグラフ構造内を巡回する．この巡回において，トークンはグラフ中の全てのアトムを訪れる必要はなく，書換え対象となるアトムのみを訪れることができればよい．トークンの巡回規則は，グラフの構造に依って決定する．例として木構造では，図 2 のように深さ優先探索順にトークンを巡回させることで，グラフ構造全体をトラバースできる．このとき，外部リンク (図中 head，木の根に相当) から末端へ向かうトークンを**順行トークン**，末端から外部リンクへ向かうトークンを**逆行トークン**と呼ぶ．

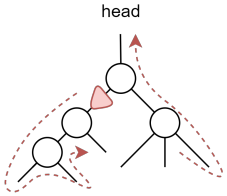

図 2 木構造におけるトークンの巡回

3.4 トークンの付加と戦略の決定

トークンによる制御において，まずはトークンの初期配置を行う必要がある．初期化操作では，グラフの外部リンクに対して，開始位置兼終了位置を示すアトム goal と，巡回用のトークンとなるアトムを付加する．この時，goal とトークンはリンクで接続し，任意のタイミングでグラフ巡回を最初からやり直せるようにする．

次に，与えられた LMNtal プログラムの書換え規則をトークンを含む規則に変更する．規則左辺においてトークンが接続された位置のみで書換えを許可し，規則右辺においても同様のトークン接続を維持または更新することで，書換えの実行順序を制御することができる．

最後に，巡回規則に従って，トークンの遷移規則を追加する．これは書換えが起こらない箇所をトークンが通過し，書換え可能部分グラフに到達するための規則である．よって，基本的には，トークンの巡回規則と同一の規則を追加することになるが，トークン付き書換え規則が適用される箇所では，トークンが遷移せず，書換え規則が適用されない場所では遷移する必要がある．

以下は木構造について，具体的な戦略によるトークンの付加方法を以下に示す．根に相当する外部リンクを木構造グラフにおける外，末端を内とする．図 2 に示す巡回規則を使用する．

3.4.1 最左最外

外部リンクから巡回規則に従ってトークンを巡回させたとき，最初に順行トークンが訪れる書換え可能部分グラフを書き換える．単純には，書換えのたびにトークンを開始位置から再出発させることで，常に最左最外を選択できる．よって，規則左辺のグラフ構造における最左最外の位置に順行のトークンを接続し，規則右辺においては，初期化時同様に goal とトークンを接続する．これは，関数を用いない書換えプログラムにおいても適用可能である．

この制御によって，外部リンク付近の木構造の書換えが優先され，外部リンクに接続された構造がより早く書換え可能部分を含まない部分グラフとなる．

3.4.2 最左最内

最左最外と異なる点として，書き換えを制御するトークンを逆行トークンに変更する．すなわち，最初に逆行トークンが訪れる書換え可能部分グラフを書き換えることで最左最内となるため，規則左辺のグラフ構造における最左最内の位置に逆行トークンを接続し，規則右辺においては，初期化時同様に goal とトークンを接続する．

3.5 効率化

基本的なトークン制御手法として，書換え操作実行のたびに開始位置から巡回を再開する方式を紹介した．しかし、この手法では書換えごとにトークンが開始点に戻る手法では，関数を含まない部分グラフや，不活性な関数しかない部分グラフの巡回が多く発生する．このような書換え可能部分が存在しないことが既知であるグラフ構造への複数回の巡回を避けるために，トークンの巡回規則を工夫する効率化手法を提案する．

3.5.1 外部探索の効率化

関数による書換えが実行プログラムに適用可能な，外部探索の効率化手法について述べる．書換え操作の実行箇所を出発点として，次の書換え可能部分グラフを探索するための補助的なトークンとして，**バックトークン**を導入する．バックトークンは書換え操作のトリガーとなることはなく，書換え操作間のトークンの移動を制御するためにのみ使用される．連結なグラフでは，ある構造が書き換えられた時，その外部構造に書換えの影響が及んでいなければ，更に外側が書換え可能となることはなく，再巡回の必要性も生じない．木構造のような明確な階層を持つグラフにおいては，常に外部，すなわち根の方向にバックトークンを遷移させ，関数に遭遇した時，順行トークンとして接続することで通常の書換え探索を再開する．この手法により，書換えの影響が及ぶ可能性のある領域の最も外側から再探索を開始することが可能となり，無関係な構造への巡回を削減し，効率的な探索が可能となる．特に最外戦略は，グラフ外縁部に書換え可能部分グラフが存在しない状態を早期に得る手法であるため，この効率化が有効であると考えられる．

バックトークンの動作例を図 3 に示す．図中 rotR は指定したノードを中心に右回転の書換えを行う関数である．トークンによって制御された ins の書換え実行後，トークンはバックトークンに変化する．バックトークンは右部分木と左部分木のどちらにつながるリンク上にあっても，外部リンクへ向かって移動する．関数 rotR に遭遇したバックトークンは順行トークンに変化し，rotR の書換えを制御する．rotR が実行可能な関数であったため，書換えが実行され，ふたたびバックトークンに変化する．もし rotR が不活性な関数であった場合，順行トークンは探索を再開することになる．

3.5.2 内部探索の効率化

書換え可能部分グラフが存在しないことが既知の部分木に対して，トークンの再進入を抑制するための**ストッパーアトム**を導入し，巡回を行わないようにすること

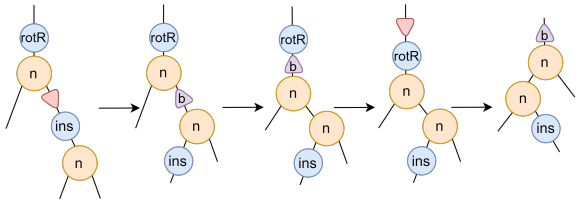

図 3　バックトークンの動作例

で，内部探索を効率化する．具体的には，逆行トークンが通過した箇所に対しては，その部分木内には書換え可能部分グラフが存在しなかったということができるため，図 4 に示すようにストッパーアトムを接続する．これにより，次回以降にその部分木をトークンが訪れた際に，ストッパーの内部を探索せずに引き返すことができる．このストッパーは書換えの実行時には無視され，トークンの巡回時のみに機能するよう設計する．実際の LMNtal プログラムとしては，左辺にストッパーを付加した書換え規則を追加で記述することで実現する．

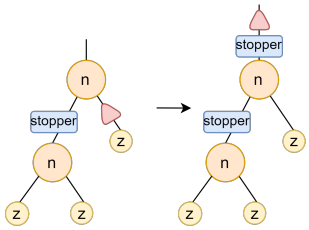

図 4　探索済み部分木に対するストッパー付加の実行例

4　ハイパートークンによる実行制御

本節では，LMNtal ShapeType [4] の型検査における山田らのトークン [5] を応用し，より一般のグラフ構造に対する書換えのトークン制御を目指す．

4.1　ハイパートークン

山田らの研究では，LMNtal ShapeType において，型定義からトークンの巡回規則を自動生成する手法を提案している．LMNtal ShapeType は，グラフ構造の形状を定義した型定義に基づいて，グラフ構造を型検査可能なグラフの型付け手法である．

トークンを用いた型検査手法では，型定義に基づいてトークンの巡回規則を生成し，トークンが検査対象のグラフ構造全体を巡回することができれば，型を満たしていると判定する．これは，従来の型検査の非決定性を抑止することを目的としている．複数本のリンクを入力として受け取るハイパートークンを使用することが特徴であり，これは実装上はリンクで優先度順に接続された複数のトークンとして記述される．

通常のトークンが一対のリンクを通じて単一経路を辿るのに対し，ハイパートークンは，トークン列を構成する各トークンが異なる方向に進み，連結なグラフ構造上の離れた箇所に同時にアクセスすることで，並行的にグラフ構造を探索できる．

4.2　書換え制御への応用

3 節で述べたように，トークンが書換え対象のグラフ構造上を巡回する規則を定義したうえで，実際のプログラムにトークンを付加し，書換えの実行順序を制御する．本節では，型を満たすグラフの書換えプログラムについて，トークンの巡回規則として，型定義から自動生成されるトークン巡回規則を応用することを提案する．

型検査と実行制御では，トークンがグラフ全体を巡回するという点は共通している．異なる点として，型検査では反例が入力された場合にトークンが巡回しきらないことを保証する必要があるのに対し，実行制御では巡回対象のグラフ構造が型を満たしていることは前提として良い．よって，実行制御においては型を満たさない入力に対するふるまいは考慮する必要がないほか，グラフ形状以外の情報，例えばリストにおける要素の順序などを検査する必要はない．

4.3　トークン列の例題 bst

二分探索木 (bst) の書換えを例に，ハイパートークンを用いた書換えの制御を示す．

4.3.1　トークンの巡回規則の決定

グラフにトークンを付加する初期化規則と，トークンが末端までトラバースする巡回規則を以下に示す．

```
start,head(INP) :-
  head(goal(token(INP,active(S),RTN),RTN,S)).

R=token(n(X,LEFT,RIGHT),active(S),RTN1) :-
  R=n(X,
    token(LEFT,S,RTN2),
    token(RIGHT,RTN2,E)), RTN1=active(E).

R=token(z,active(S),RTN) :- R=z, RTN=active(S).

H=goal(INP,active(RTN),S), A=token(B,C,RTN)
  :- H=goal(INP,RTN,S1),S=active(S1),A=token(B,C,RTN).
```

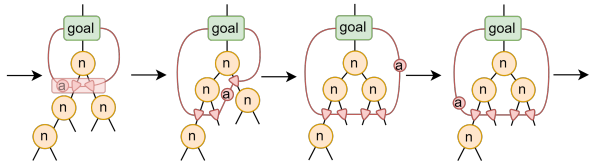

図 5　ハイパートークンの幅優先巡回

ハイパートークンがグラフ構造上を遷移する様子の一部を図 5 に示す．赤いリンクで接続されたトークン列がハイパートークンであり，トークン列の先頭と末端は goal アトムに接続されている．active アトムはトークン列の中でアクティブなトークンを表し，これがトークン列中を移動することで実行位置が制御される．初期化では，グラフの外部リンクに対して，開始位置兼終了位置であることを示す goal と，アクティブな巡回用のトークンを付加する．二分探索木は外部 (根) へのリンクを 1 本，内部 (子) へのリンクを 2 本持つ．よって，アクティブな 1 つのトークンはノードを通過して子要素へと移動するとき，2 つのトークン列となる．このとき，トークンがアクティブであることを示すアトムを移動させ，後続のトークンを実行することで，幅優先で二分探索木を巡回することができる．アクティブなトークンを変更せず，トークン列先頭のトークンを実行し続ければ，深さ優先での巡回となる．

4.3.2　トークンの付加

素朴には，書換えを行いながら各トークンを末端まで移動させ，トークン列を構成するトークンが空になったら再度開始位置からトークンを流し込むことで，実行を順を一意に制御することができる．ハイパートークンが開始位置から出発して末端に移動するまでに，関数が一度も出現しなかった場合は実行を終了する．

以下に，トークン制御による実行プログラムを示す．

```
1 B=token(ins(D,z),active(S),RTN)
2   :- B=n(D,z,z), RTN=active(S).
3 B=token(ins(D,n(F,L,R)),active(S),RTN)
4   :- D < F | B=token(n(F,ins(D,L),R),active(S),RTN).
5 B=token(ins(D,n(F,L,R)),active(S),RTN)
```

```
6     :- D > F | B=token(n(F,L,ins(D,R)),active(S),RTN).
7   B=token(ins(D,n(F,L,R)),active(S),RTN)
8     :- D = F |
        ↪ B=token(n(F,L,R),fail(ins,D),active(S),RTN).
9
10  B=token(sea(D,z),active(S),RTN)
11    :- B=z,fail(sea,D), RTN=active(S).
12  B=token(sea(D,n(F,L,R)),active(S),RTN)
13    :- D < F | B=token(n(F,sea(D,L),R),active(S),RTN).
14  B=token(sea(D,n(F,L,R)),active(S),RTN)
15    :- D > F | B=token(n(F,L,sea(D,R)),active(S),RTN).
16  B=token(sea(D,n(F,L,R)),active(S),RTN)
17    :- D = F | B=n(F,L,R),succ(sea,D), RTN=active(S).
18
19  continue(0).
20  B=token(ins(D1,ins(D2,L)),active(S),RTN),continue(C)
21    :- int(C) | B=ins(D1,token(ins(D2,L),active(S),RTN))
22                 ,continue(1).
23  B=token(ins(D1,sea(D2,L)),active(S),RTN),continue(C)
24    :- int(C) | B=ins(D1,token(sea(D2,L),active(S),RTN))
25                 ,continue(1).
26  B=token(sea(D1,ins(D2,L)),active(S),RTN),continue(C)
27    :- int(C) | B=sea(D1,token(ins(D2,L),active(S),RTN))
28                 ,continue(1).
29  B=token(sea(D1,sea(D2,L)),active(S),RTN),continue(C)
30    :- int(C) | B=sea(D1,token(sea(D2,L),active(S),RTN))
31                 ,continue(1).
32
33  H=goal(INP,active(S),S),continue(0) :- H=INP.
34  H=goal(INP,active(S),S),continue(1)
35    :- H=goal(token(INP,active(S),RTN),RTN,S),continue(0).
```

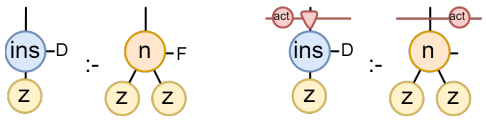

図 6　与えられた書換え規則 (左)　とトークン付き書換え規則の例 (右)

まず，書換え規則をトークン付き書換え規則に変更する．書換え規則は左辺に 1 つ以上の関数を含む．よって，規則左辺の関数のうち，最も外部リンクに近い関数に，巡回規則に現れる形でアクティブなトークンを接続し，規則右辺の同位置にも接続する．

次に，トークンの遷移規則を追加する．トークンの遷移規則は，トークンの巡回規則と，トークンが実行不可能な関数を通過する規則からなる．巡回規則は 4.3.1 節と同様であり，通過規則はそのうち 18 行目から 25 行目に示すように，この例題においては関数が連続して接続されている場合に，一つ目の関数を通過し，次の関数へとトークンを移動させる規則である．このとき，トークンが末端まで進んだ時に未実行の関数が残ることになるため，continue アトムを用いて，実行の継続を制御する．トークン列が空になった時，continue アトムに接続された値が 1 であれば開始位置から再度トークンを流し込む．値が 0 であれば，実行を終了する．

4.3.3　最外最左/最左最外

トークン遷移と書換え実行後のアクティブなトークンの制御によって，最外最左戦略，すなわち，実行可能な関数のうち，最も深さが浅い中で，最も左に位置するものを実行することができる．巡回規則は 4.3.1 節と同じ幅優先を使用する．次に，書換え規則にトークンを付加するにあたって，書換えの実行後にハイパートークンを削除し，開始位置から再出発させるようにする．このとき，巡回規則を深さ優先にすると，最左最外となる．

4.3.4　関数の実行制御

関数による書換えを制御する場合，実行の可否を判断しなければならない場所は関数の存在する場所に限られている．よって，グラフ構造上に存在する関数を探索し，関数をたばねるハイパートークンを導入することで，関数の実行制御を行う手法を考える．まず，書換えの対象となるグラフに対して，いずれかの巡回手法によって探索用のトークンを巡回させる．アクティブな探索用トークンが関数を発見した場合，関数用のハイパートークンを生成し，図 7 右に示すように，関数の位置に接続する．関数用のハイパートークンは探索が終了するまではアクティブなトークンを持たない．探索が完了して探索用トークンが消滅すると，関数用のハイパートークンによる関数の実行制御が開始される．

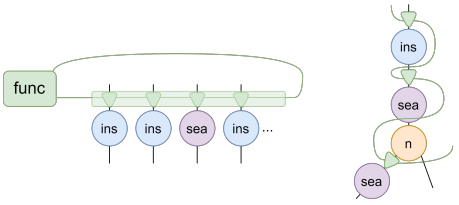

図 7　ハイパートークン (左) と データ構造中に配置されたトークン列 (右) の例

関数用のハイパートークンは，関数の位置に接続されているため，グラフの探索よりも効率的にトークンを移動させることができる．ハイパートークンに対する関数の接続順序は，関数実行の優先度とみなすことができる．探索用トークンの巡回方法によって，関数の接続順序を変更できるほか，関数の種類順に並べて接続することで，特定種類の関数を優先的に実行する等の制御もできる．また，関数による書換えでは，図 7 右の ins アトムと sea アトムのように連続して接続されている場合，内側の関数が実行された場合，すぐ外側の関数が実行可能になることが多い．よって，連続した関数はハイパートークンの列としても連続して接続しておき，いずれかの関数の実行後には，直前のトークンをアクティブにし，関数の実行可否を判断することで，効率よく関数の実行が行える．探索用トークンの巡回方法によって，ハイパートークンに対する関数の接続順序，すなわち関数の優先度を制御できる．また，探索用トークンによって発見された順序そのままではなく，関数の種類順に入れかえて接続することで，特定種類の関数を優先的に実行する等の制御も行える．

5　まとめ

本論文では，グラフ書換え言語 LMNtal において，トークンおよびハイパートークンを導入し，書換えの実行制御を行う手法を提案した．今後の課題としては，トークンによる制御戦略の正当性を形式的に検証するための理論構築，および，実行効率の向上を目指したトークン使用の最適化が挙げられる．

本研究の一部は，科学研究費補助金 23K11057 の援助を得て実施した．

参考文献

[1] 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, pp. 126–142, 2004.
[2] 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, pp. 153–154, 2008.
[3] François-Régis Sinot. Call-by-name and call-by-value as Token-passing Interaction Nets. In TLCA 2005, Vol. 3461 of LNCS, pp. 386–400, 2005.
[4] Naoki Yamamoto, Kazunori Ueda. Engineering grammar-based type checking for graph rewriting languages. IEEE Access, Vol. 10, pp. 114612–114628, 2022.
[5] 山田啓太, 山本直輝, 上田和紀. グラフ書き換え系における token passing を用いたグラフ型検査. 情報処理学会第 86 回全国大会, pp. 113–114, 2024.