Programming with Logical Links

上田 和紀 †‡ 加藤 紀夫†
Kazunori UEDA Norio KATO

† 早稲田大学理工学部情報学科
Dept. of Information and Computer Science, Waseda University
‡CREST, 科学技術振興事業団
CREST, Japan Science and Technology Corporation
{ueda,n-kato}@ueda.info.waseda.ac.jp

論理変数をリンクとする階層的グラフの書換えに基づく簡単な並行言語モデル LMNtal を提案する．本モデルは，多重集合の書換えに基づく多くの計算モデルの統合を目指すとともに，広域分散計算から極小規模計算までを包含しうる真に汎用なプログラミング言語のベースとなることを目指している．それとともに，動的データ構造のプログラミングを大幅に平易にすることも目指している．

1 はじめに

本論文の目的は，(i) 計算，特に並行計算に関する多様なパラダイムを統合し，(ii) 極小規模および広域分散の両方向へ展開しつつある計算基盤の上で幅広く使えるような，言語モデルを提案することである．設計目標は次の 4 つである．

1. 理論計算モデルとしての簡潔さと，実用言語 (のベース) としての実際性を兼ね備えること．
2. 計算に関する多様な考え方をできるだけ統合すること．たとえば，プロセスとデータ，動的プロセス構造と動的データ構造，プロセスと (プロセスがやりとりする) メッセージ，同期通信と非同期通信などの統一的な扱いを目指す．
3. 書きやすく直観的であること．並行計算や複雑なデータ構造の理解には図示がしばしば有効であることを考え，プログラムが直截的な図形変換に対応するような言語とする．
4. 効率的実装を視野に入れること．

2 言語モデルの大枠と関連研究

提案する言語モデルの主な技術要素は，リンク，多重集合，階層化，書換えである．言語名 LMNtal (elemental と発音) は links, multisets, nested nodes, transformation の 4 要素からなる言語を意味する．

(1) **リンク** — プロセス構造 (並行プロセスの相互関係) は，プロセスをノード，プロセスの論理的隣接関係をリンクで表現したグラフで表現できる．データ構造も同様にノードとリンクで表現できる．LMNtal はこの両者を統一的に扱う．つまりリンクは，隣接プロセス間の 1 対 1 通信路とデータ構造の表現の両方の機能をもつ．

並行計算における代表的な通信メカニズムは，名前による通信 (例：π 計算) と論理変数 (単一代入変数) による通信 (例：並行論理プログラミング [9]) であるが，LMNtal におけるリンク通信は，通信の前後でリンクのアイデンティティが変わる点，およびリンクが常にプライベートである点で，論理変数による通信に近い．しかし，変数を具体化するという概念が存在しない点において，並行論理プログラミングや CHR [6] における通信と異なる．

LMNtal でのリンクは基本的に無方向リンクである．ただしこれを一方向にしか使わない場合，型体系を使って向きを表現・発見することが可能となる．

(2) **多重集合とその階層化** — 多くの計算モデルが，何らかの形で多重集合を備えている．並行計算モデルではプロセスが多重集合をなす．Gamma [2] や Chemical Abstract Machine は多重集合書換えに基づく並列計算モデルであるし，線形論理に基づく言語は，シーケントの両辺が多重集合であることを活用する．Linda はタプルスペースが多重集合をなす．

一級市民として多重集合というデータ構造を許している言語としては，並行制約言語の一つである Janus [8] がある．大多数の多重集合をもつ言語では，多重集合を他のデータ構造と別の形で導入しているが，一級市民にすることの利点は，多重集合の構造化など，柔軟な扱いが可能になる点である．

LMNtal では，ノードの多重集合がノードとなれるという形で，多重集合の局所化と階層化を実現している．多重集合の階層化は，Ambient 計算をはじめとして形式言語 [7] や知識表現 [4] などの分野でも見られる．これらとの最大の違いは，リンクによる構造化と多重集合階層化の組み合せかたにある．

多重集合の階層化の概念は，(i) 論理的な計算管理 (例：ユーザプロセスと管理プロセスの区別)，(ii) 物理的な計算管理 (例：リジョンベース記憶管理)，(iii) 計算の局所化，などに重要な役割を果たす．

(3) **書換え** — LMNtal は書換え規則ベースの構文をもつ．グラフ文法とグラフ書換えに関しては多くの研究があり [1]，階層グラフの書換えも研究されているが [3] ，LMNtal の特徴は，プログラミング言語の立場からの設計，特にリンクの機能設計にある．書換え規則は多重集合要素間の反応規則であるが，リンクされた要素間の反応は，そうでない反応よりも効率的なパートナー検索が可能である．

LMNtal は，channel mobility と process mobility の両方の mobility を備える．つまり，プロセス構造の動的変更が可能なだけでなく，階層化された計算自体の移動が可能である．

3 構文

3.1 リンクと名前

最初に，次の 2 種類の構文カテゴリを仮定する．

- X は**リンク**(**リンク変数**ともいう) を表わす．具体構文では大文字から始まる識別子で表記する．
- p は**名前**を表わす．数も名前の一種である．具体構文ではリンクと区別できる識別子で表記する．名前 = は，LMNtal 唯一の予約名である．

3.2 プロセス

プロセス P を以下のように定義する．

$P ::= \mathbf{0}$		(null)
	$p(\overrightarrow{X})$	(simple process, $\overrightarrow{X}$ は X の列)
	P, P	(composition)
	$\{P\}$	(compound process)(*)
	$\{P\}/$	(quiet compound process)(*)
	$(T :- T)$	(reaction rule)

P を *nested graph* とも呼ぶ．P 中の各リンク (reaction rule 内のリンクを除く) は，**たかだか 2 回** (つまり 1 回か 2 回) しか出現しないものとする．これを**リンク条件**という．Reaction rule 内に関するリンク条件は 3.3 節で述べる．P 中に 1 回しか出現しないリンクを，P の**自由リンク**とよぶ．自由リンクを持たないプロセスを**閉じたプロセス**と呼ぶ．

直感的には，$\mathbf{0}$ は空プロセス，$p(\overrightarrow{X})$ は X の個数だけのリンクをもった単純プロセス，P, P はプロセスの並列合成，$\{P\}$ および $\{P\}/$ は膜 { } によってグループ化されたプロセス，$(T :- T)$ は P のための書換え規則である．単純プロセス $X = Y$ は，リンク X の一端と Y の一端とを接続する機能をもつ．$\{P\}$ と $\{P\}/$ の違いは，後者は P の簡約が終了していることを表わす点である．

$\{P\}$ および $\{P\}/$ の (自由) リンクとは，P の自由リンクのことであると定める．Simple process のリンクは順序づけられているが，compound process のリンクは順序づけられていない．Compound process のリンクは，実際には compound process 自身ではなくてその構成要素から出る．

LMNtal のサブセットとして，階層化を許さない Flat LMNtal を考えることができる．Flat LMNtal では，構文規則の (*) 印の部分が不要となる．

3.3 プロセステンプレート

次に，書換え規則の中で用いるプロセステンプレート T の構文を与える．

$T ::= \mathbf{0}$		(null)
	$p(\overrightarrow{X})$	(simple process)
	$\$p(FVspec)$	(process variable)(*)
	$@p$	(rule variable)(*)
	T, T	(composition)
	$\{T\}$	(compound process)(*)
	$\{T\}/$	(quiet compound process)(*)

Flat LMNtal では，(*) 印の部分が不要となる．

$\$p(FVspec)$ は**プロセス変数**と呼び，書換え規則以外のプロセスとマッチするのが役割である．$FVspec$ は自由リンクの出現に関する条件を表わし，次の構文に従う ([ ] はオプショナル，* は 0 回以上の反復)．

$$[+]\,FVspec \;::=\; [+X]^*\,[+*\,[-X]^*\,]$$

$FVspec$ 中の変数はすべて相異なるものとする．+ に続く変数は自由変数として出現しなければならず，- に続く変数は自由変数として出現してはならない．+*

は，*FVspec* に言及されていない変数が自由変数として出現してもよいことを表す．

例： `$p(X+Y)`: 自由変数は `X` と `Y` のみ
`$p(X+*-Y)`: `X` が自由変数として現れ，`Y` は自由変数として出現しない
`$p`: 自由変数はない (閉じたプロセス)

`@`p は**ルール変数**と呼び，ルールの多重集合 (空でもよい) とマッチするのが役割である．

プロセス変数とルール変数は，次の出現規則を満たさなければならない．

1. すべてのプロセス変数およびルール変数は，ルール左辺にちょうど 1 回，compound process の中に出現しなければならない (したがって Flat LMNtal ではこれらは使えない)．
2. ルール左辺の compound process は，そのトップレベルに，プロセス変数およびルール変数をたかだか 1 個ずつ持つことができる
3. 空でない *FVspec* をもつプロセス変数は，ルール右辺にちょうど 1 回出現しなければならない．

4 節の意味論で明らかになるように，1. と 2. から，プロセス変数とルール変数が受け取る値は一意となる．3. は，自由変数をもつ (かもしれない) プロセスはコピーや廃棄ができないことを表わす．

各 reaction rule (L :- R) 中のリンクは，ちょうど 2 回現れるか，L と R 中にそれぞれ 2 回ずつ現れなければならない．ただし，プロセス変数中の負の出現は数えない．L 中にのみ現れるリンクはルールによって消費されるリンク，R 中にのみ現れるリンクはルールによって生成されるリンク，1 回ずつ現れるリンクは継承されるリンク，2 回ずつ現れるリンクはリサイクルされるリンクである．

T の構文規則には (T :- T) がない．これは，本論文の範囲では，プログラムの動的生成や検査を伴うプログラム操作機能がないことを意味する．

4 操作的意味論

本節では，プロセスの構造合同 (structural congruence) を定めて論じたあと遷移意味論を与える．

4.1 構造合同

図 1 の各式の両辺は等価な (0 ステップで互いに変換可能な) プロセスと見なす．ただしここで，$[Y/X]$ はリンク X をリンク Y に置き換える**リンク代入**を，$\$p(FVspec)$ は *FVspec* を満たすプロセスを表わすメタ記法である．

$$
\begin{aligned}
\mathbf{0}, P &\equiv P && (1)\\
P, Q &\equiv Q, P && (2)\\
P, (Q, R) &\equiv (P, Q), R && (3)\\
X = X &\equiv \mathbf{0} && (4)\\
X = Y &\equiv Y = X && (5)\\
X = Y,\ \$p(X+*) &\equiv \$p(X+*)[Y/X] && (6)\\
\{X = Y,\ \$p(*)\}, \$q(X+Y+*) & \\
&\equiv \{\$p(*)\},\ \$q(X+Y+*)[Y/X] && (7)\\
P \text{ と } P' \text{ が } \alpha \text{ 同値ならば } P &\equiv P' && (8)\\
P \equiv P' &\Rightarrow P, Q \equiv P', Q && (9)\\
P \equiv P' &\Rightarrow \{P\} \equiv \{P'\} && (10)\\
P \text{ が簡約不能ならば } \{P\} &\equiv \{P\}/ && (11)
\end{aligned}
$$

図 1: LMNtal プロセスの構造合同

(4) は自己完結的リンクの消去を，(5) は = の対称性を表わす．(6) は，= プロセスが，必要ならば膜を外から中へ通過しつつ任意の隣接プロセスのところまで移動して吸収されうることを表わす．(7) は，ある膜の中の = プロセスが膜の外の 2 地点を結んで (中継して) いるとき，その中継を膜から追い出せることを表わす．(9) では，$\equiv$ の両辺でリンク条件が成立しなければならない．(11) は，簡約済のプロセスが，フラグ “/” を必要に応じて呈示できることを表わす．

4.2 遷移意味論

遷移をつかさどる reaction rule の左辺は，プロセス変数およびルール変数を含むかもしれない．これらとプロセスとのマッチングを可能にするために，まず**プロセス代入**を以下のように定義する．

$$
[W_1/\$p_1(FVspec_1), \ldots, W_n/\$p_n(FVspec_n), R_1/@q_1, \ldots R_m/@q_m]
$$

ここで W_i は $FVspec_i$ を満たしかつルールを含まないプロセス，R_i はルールの多重集合とし，また $i \neq j$ ならば $p_i \neq p_j$ かつ $q_i \neq q_j$ とする．上のプロセス代入は，プロセス変数 $\$p_i(FVspec_i)$ を W_i に，またルール変数 $@q_i$ を R_i に，それぞれ一斉に置き換えることを表わす．以下ではプロセス代入を θ と書く．

遷移意味論は**図 2** の通りである．(i)–(iii) は構造規則である．(i) は並行性を，(ii) は膜内の計算の独立

$$\frac{P \longrightarrow P'}{P, Q \longrightarrow P', Q} \quad \text{(i)} \qquad \frac{P \longrightarrow P'}{\{P\} \longrightarrow \{P'\}} \quad \text{(ii)}$$

$$\frac{P \longrightarrow P'}{Q \longrightarrow Q'} \quad (\text{if } P \equiv Q \text{ and } P' \equiv Q') \quad \text{(iii)}$$

$$\frac{}{T\theta, (T :\!\!- U) \longrightarrow U\theta, (T :\!\!- U)} \quad \text{(iv)}$$

図 2: LMNtal の遷移意味論

性を表わす．膜内で独立して計算を進めるためには，膜内にルールがなければならない．ルールを持たない compound process の計算は，外側のルールが制御する．(iv) は，ルール自体は保存されることを示している．Flat LMNtal はプロセス変数やルール変数をもたないので，(iv) はさらに簡単になる．

$$\frac{}{P, (P :\!\!- Q) \longrightarrow Q, (P :\!\!- Q)} \quad \text{(iv}')$$

(Flat) LMNtal では，プロセスとルール左辺とのマッチングは α 変換によってとられる．書換え後の $U\theta, (T :\!\!- U)$ がリンク条件を満たさなければならないことに注意してほしい．書換えによって新たなリンクが生成される場合，あらかじめ $(T :\!\!- U)$ を α 変換しておくことで，既存のリンクとの衝突を避けなければならない．

5 プログラム例

5.1 リストとその連結

リスト (の骨格) は，要素プロセス c(ons) と終端プロセス n(il) を用いて $\mathtt{c}(A_1, X_1, X_0)$, ..., $\mathtt{c}(A_n, X_n, X_{n-1})$, $\mathtt{n}(X_n)$ と表現できる．A_i は第 i 要素へのリンクで，X_0 がリストの先頭へのリンクである．これは，論理型言語のリスト $X_0 = \mathtt{c}(A_1, X_1)$, ..., $X_{n-1} = \mathtt{c}(A_n, X_n)$, $X_n = \mathtt{n}$ に対応する．リストの連結プログラムは，下記の二つのルールで書ける．

```
append(X0,Y,Z0), c(A,X,X0) :-
        c(A,Z,Z0), append(X,Y,Z)
append(X0,Y,Z0), n(X0) :- Y=Z0
```

GHC の append との対応関係は明らかであろう．

```
append(X0,Y,Z0) :- X0=c(A,X) |
        Z0=c(A,Z), append(X,Y,Z).
append(X0,Y,Z0) :- X0=n | Y=Z0.
```

このように LMNtal では，プロセスとデータとの間に構文上の区別はない．実装上の区別がありうるのみである．

5.2 ストリームの併合

ストリーム (メッセージのリスト) の併合による n 対 1 通信は，膜を使って以下のように書ける．

```
{i(X0),o(Y0),$p(*)}, c(A,X,X0) :-
        c(A,Y,Y0), {i(X),o(Y),$p(*)}
```

ここで左辺の { } 内には n 本の入力ストリームが名前 i とともに登録され，出力ストリームは名前 o とともに登録されているものとする．$p(*) は，X0 以外の入力ストリームとマッチするためのものである．

5.3 プロセス移送

独立して計算を進めている二つの compound process の間で，リンクづたいにプロセスやメッセージを移送することを考える．移送のルールを与えるのは一段階上位の層であり．ここではプロセス $\mathtt{m}(S, D)$ (S が出発地側，D が目的地側のリンク) を内部にもつ compound process を用意すると上位層がそれを移送するものとする．

```
{$p(S+*), {@q,$q(*),m(S,D)}}, {$r(D+*)} :-
    {$p(S+*)}, {{@q,$q(*),m(S,D)}, $r(D+*)}
```

{@q,$q(*),m(S,D)} を囲む膜は，資源 (プログラムを含む) の移送範囲を明示するためのものである．@q が空でなければ能動プロセスの移送に，空ならば通常のデータの移送に対応する．送受信プロセス間の共有リンクが，移送前は D だったのが移送後は S になる (D は受信側の局所リンクになる) ことに注意してほしい．これが論理変数に基づくリンクの重要な特徴である．なお，受信後の膜は，受信側で必要に応じて消去する．

5.4 反応制御

二つの独立した計算を異なる膜 (外部からの識別用のリンクを X, Y とする) の中で進め，両方の反応が止まったら最終生成物を別の膜 Z の中に移したい．この移動操作は次のように書ける．

```
{id(X),$p,@p}/, {id(Y),$q,@q}/, {id(Z)} :-
     {id(X),@p}, {id(Y),@q}, {id(Z),$p,$q}
```

5.5 数値演算

数値演算はもちろん組込みで提供されるが，ここでは基本概念を述べる．まず LMNtal では，プロセス a の第 1 引数の値が 5 であるという状況は，$\mathtt{a}(X), \mathtt{5}(X)$ などと書ける．次に，a が自分の引数をインクリメントして b に化けるという計算は

```
a(X), 5(X)   ⟶   b(X1), s(X1,X), 5(X)
             ⟶   b(X1), 6(X1)
```

という遷移 (をプログラムすること) で実現できる．最初の遷移で a がメッセージ s を送出，2 番目の遷移で s と 5 が化合している．

数値演算においては，数値のコピー機能が重要となるが，これには名前自体のプロセス的解釈が必要となる．設計は完了しているが，本稿では詳細は省く．

5.6 循環データ構造

宣言型言語は，リストや木の扱いは得意だが，サイクルを含むデータ構造の扱いは一般に得意でない．LMNtal では，たとえば双方向環状バッファは

$$\mathtt{b}(S, L_n, L_0), \mathtt{n}(A_1, L_0, L_1), \ldots, \mathtt{n}(A_n, L_{n-1}, L_n)$$

というリンク構造で表現できる．ここで b はヘッダプロセス，A_i は要素へのリンク，S はユーザプロセスからのリンクである．バッファを操作するには，S に left, right などのメッセージを送る．メッセージとバッファとの反応規則は以下のように定義できる．

```
left(S1,S), n(A,L0,L1), b(S,L1,L2) :-
    b(S1,L0,L1), n(A,L1,L2)
right(S1,S), b(S,L0,L1), n(A,L1,L2) :-
    n(A,L0,L1), b(S1,L1,L2)
put(A,S1,S), b(S,L0,L2) :-
    n(A,L0,L1), b(S1,L1,L2)
...
```

動的データ構造の記述をわかりやすくする提案としては Shape Type [5] などがあるが，Shape Type では LMNtal と逆に，グラフのノードを変数で，リンクを名前で表現している．

6 まとめ

リンク，多重集合，階層化，書換えを四大要素としてもつ単純な言語モデル LMNtal を提案した．

LMNtal は，論理変数による通信から着想を得て，プロセス，メッセージ，データの統一をはかったものである．階層的多重集合の下でのリンクの機能設計に最大の特徴がある．紙面の都合上，言語設計の詳細は，続編に譲らなければならない．論理変数をもつ強力な多重集合書換え系として CHR があるが，CHR と LMNtal とでは，論理変数の用法，階層化の有無，反応制御機能などに多くの相違点がある．

言語設計上の最大の将来課題は型体系である．プロセスの形づくる構造，リンクの方向性 (単方向ポインタとしての実装の可能性)，compound process の自由変数の管理などに関する多くの重要な性質が，型体系によって保証できると考えている．他の言語設計上の課題には，例外処理機能およびメタレベルアーキテクチャの設計がある．

実装については，プロセスの表現，ルールのコンパイル方法，並列分散処理などが課題である．また，並行計算や制約プログラミング分野の関連言語や計算モデルの LMNtal への埋込みや LMNtal による実装も課題である．

謝辞 本研究の一部は，文部科学省科学研究費基盤 (C)(2)11680370，特定 (C)(2)13324050, 特定 (B)(2) 14085205 の補助を得て行った．

参考文献

[1] Andries, M. *et al.*, Graph Transformation for Specification and Programming. *Sci. Comput. Program.*, Vol. 34, No. 1 (1999), pp. 1–54.

[2] Banâtre, J.-P. and Le Métayer, D., Programming by Multiset Transformation. *Commun. ACM*, Vol. 35, No. 1 (1993), pp. 98–111.

[3] Drewes, F., Hoffmann, B. and Plump, D., Hierarchical Graph Transformation. *J. Comput. Syst. Sci.*, Vol. 64, No. 2 (2002), pp. 249–283.

[4] Engels, G. and Schürr, A., Encapsulated Hierarchical Graphs, Graph Types, and Meta Types. *Electronic Notes in Theor. Comput. Sci.*, Vol. 1 (1995), pp. 75–84.

[5] Fradet, P. and Le Métayer, D., Shape Types. In *Proc. POPL'97*, ACM Press, 1997, pp. 27–39.

[6] Frühwirth, T., Theory and Practice of Constraint Handling Rules. *J. Logic Programming*, Vol. 37, No. 1–3 (1998), pp. 95–138.

[7] Păun, Gh., Computing with Membranes. *J. Comput. Syst. Sci.*, Vol. 61, No. 1 (2000), pp. 108–143.

[8] Saraswat, V. A., Kahn, K. and Levy, J., Janus: A Step Towards Distributed Constraint Programming. In *Proc. 1990 North American Conf. on Logic Programming*, MIT Press, 1990, pp. 431–446.

[9] Ueda, K., Resource-Passing Concurrent Programming. In *Proc. TACS 2001*, LNCS 2215, Springer, 2001, pp. 95–126.