分散プロセス計算の LMNtal へのエンコーディング

Encoding Distributed Process Calculi into LMNtal

上田 和紀 †
Kazunori UEDA

† 早稲田大学理工学術院コンピュータ・ネットワーク工学科
Dept. of Computer Science, Waseda University
ueda@ueda.info.waseda.ac.jp

我々は，階層グラフ書換えモデル LMNtal と他言語モデルとの関連付けを進めるべく，関数・並行・分散・制約等の概念を表現する多様な計算モデルの LMNtal へのエンコーディングと処理系上での動作確認を行ってきた．本論文では，局所性や移動の概念を持つ ambient 計算のエンコーディングを動作例とともに紹介する．技術的な要点は名前の分散管理であり，自己調整に基づく管理方式を実装するとともに，名前管理と本来の計算とが並行動作できるようにした．

1 はじめに

LMNtal [2][3] は階層グラフ書換えに基づく言語モデルであり，接続構造の表現に論理変数を，階層構造の表現に膜を用いることを特徴としている．LMNtal は，多重集合や並行処理やモビリティなどの概念を持つ計算モデルの統合を目指すと同時に，階層グラフ書換えに基づく実用プログラミング言語と処理系[1]を提供してその有用性を示すことを目標としている．

統合計算モデルとしての LMNtal の記述力を検証するために，我々は λ 計算や非同期 π 計算などの計算モデルを LMNtal にエンコードしてきた [5]．λ 計算のエンコードは，グラフ構造で表現した λ 式に対して β 簡約，δ 簡約（λ 計算にとっての他言語インタフェース），およびグラフ構造の複製と廃棄のためのルールセットを与えることで実現した．π 計算のエンコードでは，π 計算の名前をセル（プロセスを膜で囲んだもの）とそのセルに入射するリンク（“+” アトムによって終端する）とで表現することによって，名前の参照を図形的に明示するエンコードを行った．

LMNtal の設計目的の一つに広域分散計算の表現がある．そこで本論文では，既存の広域分散計算モデルと LMNtal との関係づけの一環として，代表的な分散プロセス計算モデルである ambient 計算 [1] のエンコードについて検討する．

分散プロセス計算は，場所の概念を持つプロセス計算の総称である．場所を表現するためのコンストラクトとして多用されるのが膜 (membrane) である．膜は並行プロセス（プロセスの多重集合）を包んで局所性を実現する．Ambient の本質も階層化可能な膜であり，管理ドメインの表現を目的の一つとしている点で LMNtal の膜と共通している．また，遠隔の場所に存在するプロセスどうしが膜のトポロジを無視して直接相互作用することはなく，近接作用に基づいて計算が進む点でも，LMNtal と ambient 計算は共通している．一方，Ambient 計算では ambient の階層構造が動的に変化し，名前の有効範囲 (scope) もそれに伴って変化するので，それらの変化をいかにエンコードするかを検討するのが，本研究の主要な動機である．

[1] http://www.ueda.info.waseda.ac.jp/lmntal/

2 Ambient 計算

Ambient 計算 [1] は，ambient と呼ばれる階層化可能な計算主体が，ambient の階層構造の中を認証に基づいて移動することを特徴とする計算モデルである．Ambient 計算は移動と通信の両側面を持つが，核心は移動，つまり pure mobility calculus の部分にあるので本論文ではそのエンコードについて検討する．

Ambient 計算の式は，名前を表す構文カテゴリ n の存在を仮定して，以下の構文で定義される．

$$\begin{aligned}\text{(processes) } P &::= (\nu n)P \mid \mathbf{0} \mid P \mid Q \\ &\quad \mid\ !P \mid n[P] \mid M.P \\ \text{(capabilities) } M &::= \mathsf{in}\ m \mid \mathsf{out}\ m \mid \mathsf{open}\ m\end{aligned}$$

$(\nu n)P$ は名前の隠蔽（または新たな局所名の生成），$\mathbf{0}$ は空プロセス，$P \mid Q$ は並列合成，$!P$ は P の繰返し，$n[P]$ は名前 n をもつ ambient，$M.P$ は M を実

$$
\begin{array}{rl}
\text{(process)} & P ::= \mathbf{0} \mid p(X_1,\ldots,X_m) \mid P,P \mid \{P\} \mid T :\!\!- T \\
\text{(process template)} & T ::= \mathbf{0} \mid p(X_1,\ldots,X_m) \mid T,T \mid \{T\} \mid T :\!\!- T \\
& \quad \mid \texttt{@}p \mid \$p[X_1,\ldots,X_m \mid A] \mid p(*X_1,\ldots,*X_n) \\
\text{(residual)} & A ::= [] \mid *X
\end{array}
$$

図 1: LMNtal の構文

$$
\begin{array}{c}
\text{(E1)}\quad \mathbf{0},P \equiv P \qquad \text{(E2)}\quad P,Q \equiv Q,P \qquad \text{(E3)}\quad P,(Q,R) \equiv (P,Q),R \\
\text{(E4)}\quad P \equiv P[Y/X] \qquad \text{ただし } X \text{ は } P \text{ の局所リンク} \\
\text{(E5)}\quad P \equiv P' \Rightarrow P,Q \equiv P',Q \qquad \text{(E6)}\quad P \equiv P' \Rightarrow \{P\} \equiv \{P'\} \\
\text{(E7)}\quad X = X \equiv \mathbf{0} \qquad \text{(E8)}\quad X = Y \equiv Y = X \\
\text{(E9)}\quad X = Y,\ P \equiv P[Y/X] \qquad \text{ただし } P \text{ はアトムで } X \text{ は } P \text{ の自由リンク} \\
\text{(E10)}\quad \{X = Y,\ P\} \equiv X = Y,\ \{P\} \qquad \text{ただし } X \text{ と } Y \text{ のどちらか一方が } P \text{ の自由リンク} \\
\text{(R1)}\ \frac{P \longrightarrow P'}{P,Q \longrightarrow P',Q} \qquad \text{(R2)}\ \frac{P \longrightarrow P'}{\{P\} \longrightarrow \{P'\}} \qquad \text{(R3)}\ \frac{Q \equiv P \quad P \longrightarrow P' \quad P' \equiv Q'}{Q \longrightarrow Q'} \\
\text{(R4)}\quad \{X = Y,P\} \longrightarrow X = Y,\{P\} \qquad \text{ただし } X \text{ と } Y \text{ は } \{X = Y,P\} \text{ の自由リンク} \\
\text{(R5)}\quad X = Y,\{P\} \longrightarrow \{X = Y,P\} \qquad \text{ただし } X \text{ と } Y \text{ は } P \text{ の自由リンク} \\
\text{(R6)}\quad T\theta,(T :\!\!- U) \longrightarrow U\theta,(T :\!\!- U)
\end{array}
$$

図 2: LMNtal の操作的意味論

行して P になるプロセスを表す.

Ambient 計算の操作的意味論は構造合同と簡約関係からなる．構造合同の各規則と簡約関係の中の構造規則 (structural rules) は標準的なので省き，in m, out m, open m に関する簡約規則を下に掲げる.

$$
\begin{array}{rcl}
n[\mathsf{in}\ m.P \mid Q] \mid m[R] & \rightarrow & m[n[P \mid Q] \mid R] \\
m[n[\mathsf{out}\ m.P \mid Q] \mid R] & \rightarrow & n[P \mid Q] \mid m[R] \\
\mathsf{open}\ m.P \mid m[Q] & \rightarrow & P \mid Q
\end{array}
$$

in は ambient の兄弟関係を親子関係に変換する．out は逆に親子関係を兄弟関係に変換する．open は ambient の膜を消去して，その内容物を親 ambient 直属に変更する．いずれの操作も，名前 m をもつ ambient がなければ中断する.

3 LMNtal

本節では LMNtal の構文と意味を簡単に紹介する.

LMNtal の構文は，リンク名とアトム名の二つの構文カテゴリ（それぞれ X および p と表記）を前提として，図 1 のように表される．$\mathbf{0}$ は空プロセス，$p(X_1,\ldots,X_m)$ $(m \geq 0)$ は m 引数アトム，P,P は並列合成，$\{P\}$ はプロセス P を膜 { } で囲った**セル**，$T :\!\!- T$ はプロセスの書換え規則である．2 引数アトム = は，引数のリンクを相互接続する組込みプロセスで，**コネクタ**と呼ばれる.

プロセス P の各リンク名の出現回数はたかだか 2 回でなければならない（**リンク条件**）．1 回だけ出現するリンクを P の**自由リンク**と呼ぶ.

書換え規則の両辺のプロセステンプレートには，上記に加えて以下のものも指定できる．**ルール文脈** @p はセル内のルールセット（ルールの多重集合）にマッチし，**プロセス文脈** $\$p[X_1,\ldots,X_m \mid A]$ $(m \geq 0)$ はセル内のルール以外のプロセスにマッチする．プロセス文脈の引数は，自由リンクの出現に関する条件を指定する．$X_1,\ldots,X_m$ は必ず出現しなければならない自由リンクである．剰余引数 (residual) A が $*X$ の形のときは，$X_1,\ldots,X_m$ 以外のリンクが自由出現してもよく，$*X$ はそれらのオプショナルな自由リンクの列を表す．剰余引数 A が [] のときは他の自由リンクがないことを示す．最後の $p(*X_1,\ldots,*X_n)$ $(n > 0)$ は，同名のアトムの集団 (aggregate) を表す記法である（詳細は [2][3] を参照してほしい）．プログラムの実行過程でリンク条件が保存されるように，書換え規則にもいくつかの構文条件が課されている.

LMNtal の操作的意味論（図 2）は構造合同規則

(E1)–(E10) と簡約関係 (R1)–(R6) からなる．(E4) は α 変換を表し，(E9)–(E10) はアトムとセルによる = の吸収放出規則をそれぞれ表す．(R1)–(R3) は標準的な構造規則，(R4)–(R5) は = の移動規則である．LMNtal における中心的規則は (R6) で，代入 θ はプロセス文脈やルール文脈やアトム集団を具体的なプロセス，ルール，アトムに対応づけるものである．

4 Ambient 計算のエンコード方針

Ambient 計算と LMNtal の最大の共通点は，階層化可能な膜をもっている点である．そこで，ambient を LMNtal のセルに対応づけることを基本方針としてエンコードを行う．設計上の大きな考慮点は，ambient 計算における名前の表現と管理である．

4.1 名前の表現

π 計算と同様，ambient 計算では名前が非常に重要な役割を果たす．名前に関する基本操作として，(a) 新たな局所名を作る，(d) 通信機能を使って他プロセスに渡す，(c) ambient に命名する，(d) ケイパビリティを形作る，がある．(c)(d) は対をなしており，ambient 名（特に局所名）は ambient からの入退室や ambient 消去の認証のための秘密鍵として利用される．

Ambient 計算の名前を LMNtal で表現する方法として，(i) LMNtal のアトム名にマップする方法と (ii) セルとリンクによるグラフ構造にマップする方法とが考えられるが，我々は後者を採用する．つまり，名前はその同一性 (identity) を表現するセル（名前セルと呼ぶことにする）に対応させ，名前の各出現は，その名前セルに入射するリンクによって表現する．アトム名でなくてグラフ構造を用いるのは，次の動機と目的からである．

1. LMNtal のグラフ書換えの記述力を確認する．
2. 名前の参照トポロジやその変更を明示化する．
3. ν によって導入される局所名を扱う．
4. LMNtal のアトム名の利用は，ambient 計算の各コンストラクトのエンコード用に限定する．

なお，名前セルは，ambient セルのようにルールを包含する管理ドメインを形成するのが目的ではなく，当該名の参照の多重集合を若干の付加情報とともに管理するのが目的である．つまりフィールド名の多重性を許したレコードとしてセルを利用している．

我々は同様の方針で π 計算のエンコードも行ったが [5]，ambient 計算のエンコードが π 計算のエンコードと異なるのは ambient を表す膜の存在である．Ambient 計算における名前は ambient のさまざまな階層から参照されるが，それらの名前の同一性は，大域的に管理すると同時に，計算を局所的に進めるために各 ambient 内部で局所的に判定することもできなければならない．これらの要請を満たすために，名前は大域的な名前セルで一括管理するのでなく，各 ambient で，その内部で使われている各名前に対応するプロキシセルを保持することとした．つまり，**ルートセル**と**プロキシセル**（両者を**名前セル**と総称する）からなる木構造によって，(ambient 計算の）名前を表現することとした．これを**名前木**と呼ぶ．

ルートセルおよび個々のプロキシセルはそれぞれ

$$\{\texttt{id}, \texttt{name}(n), \texttt{+}L_1, \ldots, \texttt{+}L_m\} \quad \text{（大域名のルート）}$$
$$\{\texttt{id}, \texttt{+}L_1, \ldots, \texttt{+}L_m\} \quad \text{（局所名のルート）}$$
$$\{\texttt{id}, \texttt{-}L_0, \texttt{+}L_1, \ldots, \texttt{+}L_m\} \quad \text{（プロキシ）}$$

という形を持つ ($m > 0$)．L_0 は上位の名前セルにつながるリンクである．$L_1, \ldots, L_n$ は ambient 内部からの参照リンクで，リンクの他端は名前の出現箇所もしくは子 ambient のプロキシにつながる．`name(`n`)` は `name(L),`n`(L)` の略記法である．

名前木に対してその**正規形**を定義する．Ambient の階層は木構造をなすが，正規形の名前木は，ambient 木構造の連結部分グラフ (connected subgraph) の最上位の節点にルートセルを配置し，それ以外の各節点に一つずつプロキシセルを配置したものでなければならない．この定義と上述の条件 $m > 0$ から次の 2 つの性質が導かれる．

1. 名前木の葉が置かれる ambient はその名前の参照を持つ．
2. 名前木の節点どうしを接続するリンクは，ambient 境界をちょうど 1 回通過する．

さらに，大域的な名前は，ambient 階層の最上位にルートセルを置くのを正規形とし，局所名は，その局所名への参照をすべて含む ambient の中で最下位の ambient にルートセルを置くのを正規形とする（最小性）．また同一の大域名に対する名前木は一つにまとめたものを正規形とする．図 3 に，二つの大域名 (a, b) と三つの局所名に対応する正規形の名前木をもつ ambient 構造の例を示す．楕円が ambient,

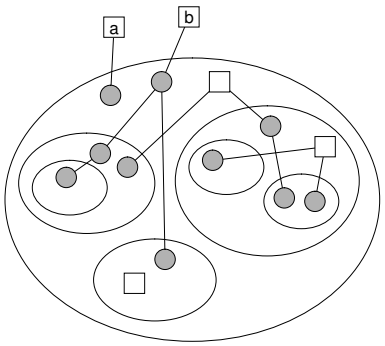

図 3: ambient 構造と名前木

正方形が名前木のルートセル，小円がプロキシセルである．

4.2 エンコード規則

Ambient 計算のプロセスから LMNtal プロセスへのエンコードを以下の関数 $[\![\cdot]\!]$ によって定義する（繰返し機能 $!P$ のエンコードは次節で論じる）．

$$
\begin{aligned}
[\![\mathbf{0}]\!] &\stackrel{\text{def}}{=} \mathbf{0} \\
[\![P \mid Q]\!] &\stackrel{\text{def}}{=} ([\![P]\!],[\![Q]\!])\downarrow \\
[\![(\nu n)P]\!] &\stackrel{\text{def}}{=} (\mathrm{hide}_n([\![P]\!]\downarrow))\downarrow \\
[\![n[P]]\!] &\stackrel{\text{def}}{=} \{@amb,\mathtt{amb(L)},[\![n]\!](\mathtt{L}),[\![P]\!]\}\downarrow \\
[\![M.P]\!] &\stackrel{\text{def}}{=} ([\![M]\!]([\![P]\!]))\downarrow \\
[\![op\ n]\!] &\stackrel{\text{def}}{=} [\![op]\!]([\![n]\!]) \quad (op \in \{\mathsf{in},\mathsf{out},\mathsf{open}\}) \\
[\![op]\!] &\stackrel{\text{def}}{=} \lambda f.\lambda p.(op(\mathtt{L},\mathtt{M}),\{\mathtt{+M},p\},f(\mathtt{L})) \\
&\quad (op \in \{\mathsf{in},\mathsf{out},\mathsf{open}\}) \\
[\![n]\!] &\stackrel{\text{def}}{=} \lambda l.\{\mathtt{id},\mathtt{name}(n),\mathtt{+}l\}
\end{aligned}
$$

ここで $\downarrow$ は名前木の正規化関数であり，グラフとしての接続関係を維持したままルートセルとプロキシセルの場所や個数を調整して正規形に変形する．具体的には次節の LMNtal コードによって示す．また (νn) は演算ではなくて構文機能として扱うべきであり，下記の補助関数 hide_n で大域名情報を消去する．

$$\mathrm{hide}_n((\{\mathtt{id},\mathtt{name}(n),P\},Q)) = (\{\mathtt{id},P\},Q)$$

$[\![n[P]]\!]$ の定義中の $@amb$ は次節に示す簡約規則の集合を表す．Ambient の内部で計算を局所的に進めるため，簡約規則は ambient 膜の中でも有効でなければならない．一方 LMNtal の簡約規則は，同一の場所に置かれたプロセスにしか作用しないので，個々の ambient 膜の中に ambient 計算のルールセットを明示的に展開しなければならない．

5 LMNtal による実現

前節に述べた方針にしたがった ambient 計算のエンコードを図 4 に示す．LMNtal の拡張構文 [5] を利用して，各ルールの先頭にルール名を付加している．引数をもたないプロセス文脈 `$p` は自由リンクの出現に制約条件のないプロセス文脈であり，`$p[|*X]` という形式の略記である．

ambient を新規に作るときは，内部に `amb.use` というアトムを置く．一般に *modulename*`.use` は，モジュール *modulename* に言及することによってそのモジュールのルールセットをその場所に展開するための標準的イディオムである．アトム `amb.use` 自体は計算には不要なので，展開されたルールセット中の `gc_amb` ルールによって消去している．図 4 のルール in, out, open には `amb.use` が出現しないが，計算を進めるためのルールセットはルール文脈（`@p`, `@q`, ...）が左辺で受け取って右辺で再利用している．

in, out, open の簡約規則に対応する LMNtal のルールはそれぞれ 1 本で書ける．これらのルールは，名前セルを ambient とともに移動させるため，名前木の正規性は必ずしも保たれない．正規性が満たされないと，上記の 3 本のルールを用いた次の簡約において，名前の同一性を認識できないことがある．正規性の問題は，in と out で直接言及している ambient 名にとどまらず，それらの ambient がもつすべての自由名について発生するので，図 4 の最初の 3 本のルールを補強することによって解決することは難しい．

5.1 名前の管理

そこで，名前管理のためのルールは，ambient の移動や消去のルールとは独立に与える．

`proxy_` から始まる名前をもつルールは，ambient の階層構造の変化に伴って各 ambient の参照する名前の集合が変化した場合に名前木構造を正規化するためのものである．`proxy_enter` は同一 ambient 内での同一名の参照を，その ambient 内部のプロキシで認識できるようにする．`proxy_resolve` は，同一名に対するプロキシが同一 ambient 内で直列につながっている場合に両者を併合する．`proxy_insert_middle` は，直接つながっている二つの名前セルが，in によって直接の親子関係にない ambient に置かれた場合に，中間の ambient にプロキシを挿入する．`proxy_insert_outer` は，あるプロキシが管理する子プロキシが out によって当該 ambient 外に輸出さ

```
{ module(amb).

/* n[in m.P | Q] | m[R] --> m[n[P|Q] | R] */
in@@
{amb(N0), {id,+N0,$n}, {id,+M0,-M1,$m0}, in(M0,{$p}), $q,@q},
{amb(M2), {id,+M2,-M3,$m1}, $r,@r},
{id,+M1,+M3,$m2} :-
   {amb(M4), {id,+M4,+M5,-M,$m1},
      {amb(N2), {id,+N2,$n}, {id,-M5,$m0}, $p,$q,@q},
   $r,@r},
   {id,+M,$m2}.

/* m[n[out m.P | Q] | R] --> n[P|Q] | m[R] */
out@@
{amb(M0), {id,+M0,+M2,$m1}, {id,+N1,$n2},
  {amb(N0), {id,+M1,-M2,$m0}, {id,+N0,-N1,$n}, out(M1,{$p}), $q,@q},
  $r,@r} :-
    {amb(N2), {id,-M3,$m0}, {id,+N2,-N3,$n}, $p,$q,@q},
    {amb(M4), {id,+M3,+M4,$m1}, {id,+N3,$n2}, $r,@r}.

/* open m.P | m[Q] --> P|Q */
open@@
open(M,{$p}), {amb(M1), {id,+M1,-M2,$mm}, $q,@q}, {id,+M,+M2,$m} :-
   $p, $q, {id,$m,$mm}.

proxy_enter@@
{$p[M0,M1|*P],@p}, {id,+M0,+M1,$m} :-
   {$p[M0,M1|*P],@p, {id,+M0,+M1,-M}}, {id,+M,$m}.

proxy_resolve@@
{id,-M,$m0}, {id,+M,$m1} :- {id,$m0,$m1}.

proxy_insert_middle@@
{{{id,-M,$m},$p,@p},$q,@q} :- {{id,+M0,-M}, {{id,-M0,$m},$p,@p},$q,@q}.

proxy_insert_outer@@
{{id,+M0,$m0},$p,@p} :- {{id,-M,$m0},$p,@p}, {id,+M0,+M}.

proxy_merge_outer@@
{id,+M0,$m0}, {id,+M1,$m1}, {{id,-M0,-M1,$m2},$p,@p} :-
     {id,+M,$m0,$m1}, {{id,-M,$m2},$p,@p}.

local_name_in@@
{$p[M|*P],@p}, {id,+M} :- {{id,+M}, $p[M|*P],@p}.

global_name_out@@
{{id,name($n),+M0},{$p[M0|*M],@p},$q,@q} :- unary($n) |
    {{id,+M0,-M},{$p[M0|*M],@p},$q,@q}, {id,name($n),+M}.

root_merge@@
{id,name($n0),$m0}, {id,name($n1), $m1} :- unary($n0), unary($n1), $n0=$n1 |
  {id,name($n0),$m0,$m1}.

gc_local_name@@ {id} :- .
gc_global_name@@ {id,name($n)} :- unary($n) | .
gc_proxy@@ {id,+X,$m}, {{id,-X}, $p,@p} :- {id,$m}, {$p,@p}.
gc_amb@@ amb.use :- .

}.
```

図 4: ambient 計算の LMNtal へのエンコード

れたときに，親の階層に新たなプロキシを作成する．proxy_merge_outer は，同一名に対するプロキシの親が複数個できた場合に両者を併合する．

また，local_name_in，global_name_out は，ルートセルの場所を正規化するためのものであり，前者はルートセルを可能な限り内側に，後者はルートセルを ambient の階層構造の最上位に向かって移動させる．root_merge は，同一の大域名をもつルートセルどうしを併合する．

gc_から始まるルールは使われなくなった名前の整理のためのものである．gc_local_name と gc_global_name は参照されなくなったルートセルの消去を，gc_proxy は参照されなくなったプロキシセルの消去を行う．

図 4 の各ルールは，各名前に対応する名前木に関する以下の不変条件を保存する．

1. その名前のすべての出現がいずれかの名前セルに接続されている．
2. 同一大域名をもつルートセルどうしは連結されているものと見なすと，(ambient 計算の）各名前は名前セルの各連結成分に一対一対応する．
3. 名前セルどうしを接続するリンクの一端は - アトムで，他端は + アトムで終端されている．
4. 名前セルから名前出現へのリンクは ambient 膜を横切らない．

正規化ルールは，上記の不変条件を保ちながら，名前木を正規形に近づけてゆく．どの正規化ルールも適用できなければ名前木が正規形であることと，正規形が一意的に定まることは容易に確かめることができる．

なお，正規化作業は ambient 計算本来の計算と並行して行うことができる．正規化が未完了の状態でも in, out, open ルールが適用可能な場合があるが，この場合も上記の不変条件は保たれ，本来の計算と正規化とが非同期的に進むことになる．

5.2 繰返し機能とそのエンコード

Ambient 計算に限らず，並行計算モデルの多くは繰返し機能 $!P$ を備える．$!P$ の意味は構造合同規則 $!P \equiv P \mid !P$ によって定義されるが，無限個の P を生成するのが目的ではなくて，計算を進めるのに必要になったときに P のインスタンスを生成することが目的である．またその用途も計算モデルごとに限られていて，ambient 計算においてはほぼ !(open $n.P$) の形でしか使われない．これは手続き呼出しのエンコードと見なすことができ，ambient n を作成するたびに手続き本体 P が実行される．π 計算でも ! の用途のほとんどは手続きのエンコードである．

そこで，!(open $n.P$) の形に限定したエンコードを図 5 に示す．!(open $n.P$) のエンコードで問題となるのは，P の複製に伴って P の自由名に対する新たな参照が発生する点である．自由リンクをもつ $[\![P]\!]$ の複製は，現在の処理系では nlmem (nonlinear membrane) ライブラリを使って記述する．nlmem.copy({P},a,R) は，セル{P}のコピーを二つ作り，それぞれへの入射リンク（+ で終端）を 3 価アトム a で併合して R と接続する．また，もともとの {P} の各自由リンクについて，{P} の二つのコピーの対応する自由リンクを，当該自由リンクの相手方と 3 価アトム a で相互接続する．nlmem.copy の意味は，aggregate を用いた以下のルールスキームによって記述できる．

```
nlmem.copy({$p[|*X]},a,R) :-
    {+R1, $p[|*Y]}, {+R2, $p[|*Z]},
    a(*Y,*Z,*X), a(R1,R2,R)
```

6 例題

我々は，文献 [1] の例題の大多数を LMNtal にエンコードし，実際に LMNtal 処理系上で動作させた．最初の例として，ファイアウォールアクセスのエンコードとりあげる．下記のように定義される *Firewall* と *Agent* の並列合成を展開したものが図 6 である．

$$\begin{aligned} \mathit{Firewall} &\stackrel{\text{def}}{=} (\nu w)w[k[\mathsf{out}\ w\,.\,\mathsf{in}\ kk\,.\,\mathsf{in}\ w] \\ &\qquad \mid \mathsf{open}\ kk\,.\,\mathsf{open}\ kkk\,.\,P] \\ \mathit{Agent} &\stackrel{\text{def}}{=} kk[\mathsf{open}\ k\,.\,kkk[Q]] \end{aligned}$$

上の定義は，非公開の ambient 名 w をもつ Firewall ambient の中に，鍵 k, kk, kkk を保有する Agent が外から入るためのプロトコルを表現している．w の中の ambient k が一旦外に出てエージェントを招き入れるというのが基本アイデアである．図 6 の pp, qq は入室後実行すべきプロセス P および Q をそれぞれ記号的に表したものである．このプログラムの実行結果は，ルールを表示しないオプションを与えると

```
{pp, qq, amb(L749), {id, +L749}}
```

```
/* !(open m.P) | m[Q] --> P | Q | !(open m.P) */
open_repl@@  /* special case of !open */
open_repl(M,{$p}), {amb(M1), {id,+M1,-M2,$mm}, $q,@q}, {id,+M,+M2,$m} :-
   nlmem.copy({$p},cp,Copies), copies(Copies,P),
   $q, {id,+M3,$m,$mm}, open_repl(M3,P).
open_repl_aux@@
copies(cp(C1,C2),P), {+C1,$p1} :- $p1, P=C2.
```

図 5: 繰返しのエンコード

```
// Firewall Access
//   Firewall =def (new w) w[k[out w.in kk.in w] | open kk.open kkk.P]
//   Agent    =def kk[open k.kkk[Q]]

{id,name(k),+K9,+K3}, {id,name(kk),+L3,+L9}, {id,name(kkk),+M9,+M3}, {id,+W9},
{amb.use. amb(W0),
  {id,+W0,+W8,-W9}, {id,+K8,-K9}, {id,+L1,+L8,-L9}, {id,+M0,-M9},
  {a.use. amb(K0), {id,+K0,-K8}, {id,+W1,+W2,-W8}, {id,+L0,-L8},
     out(W1,{in(L0,{in(W2,{})})})},
  open(L1,{open(M0,{pp})})
},
{amb.use. amb(L2), {id,+L2,-L3}, {id,+K2,-K3}, {id,+M2,-M3},
  open(K2,{{amb(M1), {id,+M1,-M2}, qq}})
}.
```

図 6: 記述例：Firewall Access

となる．これは $(\nu w)w[P \mid Q]$ を表し，非公開名をもつ ambient に P が入室できたことを示している．

次に，ambient 計算の主体的移動を用いて客体的移動 (objective move) をエンコードする例を図 7 に示す．客体的移動とは，ambient 膜に守られていないプロセスを入室または退室させる操作であり，ambient に守られた計算の移動と異なり，関係する ambient の許可が本質的に重要となる．

$$
\begin{aligned}
\mathsf{allow}\ n &\stackrel{\text{def}}{=} !(\mathsf{open}\ n) \\
\mathsf{mv\ in}\ n.P &\stackrel{\text{def}}{=} (\nu k)k[\mathsf{in}\ n.\mathit{enter}[\mathsf{out}\ k.\mathsf{open}\ k.P]] \\
\mathsf{mv\ out}\ n.P &\stackrel{\text{def}}{=} (\nu k)k[\mathsf{out}\ n.\mathit{exit}[\mathsf{out}\ k.\mathsf{open}\ k.P]] \\
n^{\Updownarrow}[P] &\stackrel{\text{def}}{=} n[P \mid \mathsf{allow}\ \mathit{enter}] \mid \mathsf{allow}\ \mathit{exit}
\end{aligned}
$$

mv in と mv out がそれぞれ in と out の客体版である．$n^{\Updownarrow}[P]$ は mv in と mv out の対象となることを許可した ambient である．

図 7 では，これら四つの補助定義をまとめたモジュール b を展開して，mv in $n.P \mid n^{\Updownarrow}[Q]$ を実行している．結果は

```
open_repl(L270,L535),
 {pp, qq, amb(L324), open_repl(L591,L601),
   {+L601},
   {id, -L557, +L591},
   {id, -L338, +L324}},
 {id, n(name), +L338},
 {id, exit(name), +L270},
 {id, enter(name), +L557},
 {+L535},
```

となる．これは $n[P \mid Q \mid \mathsf{allow}\ \mathit{enter}] \mid \mathsf{allow}\ \mathit{exit}$, つまり $n^{\Updownarrow}[P \mid Q]$ を表している．

7 考察とまとめ

Ambient 計算では，ambient の階層構造の動的変化とともに名前の参照の階層構造も動的に変化する．Ambient は管理ドメインを表現し，名前は管理ドメインへのアクセス権限を表現するが，ambient 計算の枠組みでは，名前参照という資源の管理ドメイン間の移動（という重要な操作）が ambient の移動に伴って暗黙のうちに行われていた．LMNtal による ambient 計算のエンコードでは，名前参照のトポロジ

```
// Objective Moves
//   allow n    =def !open n
//   mv in n.P  =def (new k) k[in n.enter[out k.open k.P]]
//   mv out n.P =def (new k) k[out n.exit[out k.open k.P]]
//   n_dnup[P]  =def n[P | allow enter] | allow exit

{ module(b).
  allow(N) :- open_repl(N,{}).
  mv_in(N,{$p}) :- {id,+K}, {id,name(enter),+E},
    {amb.use. amb(K0), {id,+K0,+K9,-K}, {id,+N0,-N}, {id,+E1,-E},
       in(N0,{{amb(E0), {id,+E0,-E1}, {id,+K1,+K2,-K9}, out(K1,{open(K2,{$p})})}})}.
  mv_out(N,{$p}) :- {id,+K}, {id,name(exit),+E},
    {amb.use. amb(K0), {id,+K0,+K9,-K}, {id,+N0,-N}, {id,+E1,-E},
       out(N0,{{amb(E0), {id,+E0,-E1}, {id,+K1,+K2,-K9}, out(K1,{open(K2,{$p})})}})}.
  n_dnup(N,{$p}) :- {id,name(enter),+E}, {id,name(exit),+Ex0},
    {amb.use. b.use. amb(N0), {id,+N0,-N}, {id,+E0,-E}, $p, allow(E0)}, allow(Ex0).

  b.use :- .
  {module(b), @b} :- .
}.

b.use.
{id,name(n),+N0,+N1}, mv_in(N0,{pp}), n_dnup(N1,{qq}).
```

図 7: 記述例：Objective Moves

を名前木の形で明示化し，その管理アルゴリズムを自律的かつ非同期的な木の書換え規則の形で与えた．

エンコードは全部で 15 本のルールからなり，3 本は ambient 計算の三つの基本操作の直訳，8 本は名前管理，残りはごみ集めである．LMNtal の特徴は計算の図形的な解釈にあり，ルールの本数が少ないだけでなく，いずれのルールも図形的に容易に理解できることが利点である．

名前のグラフ構造（名前木）へのエンコードは，ambient 計算における名前の挙動の明示化と理解に有用であった．プロキシセルを用いた名前管理は現実的な分散システムの実装においても必要となる技法である．プロキシセルは π 計算では不必要であったが，これは ambient 計算の方が π 計算よりも本質的に複雑であることを示唆している．

並行計算は一般にプロセスの多重集合の書換えであり，多重集合書換え言語でもある LMNtal は，並行計算の操作的意味論の簡潔なエンコードに適していると言える．Ambient 計算は boxed ambients, safe ambients, bioambients などいくつかの変種をもつが，図 4 のプログラムの中の基本操作に対応する部分を別のルールに置き換えることによって直ちにそれらの実装を得ることができる．今後も多様な計算モデルのエンコードを進め，統合計算モデルとしての LMNtal の有用性を高めてゆきたい．

謝辞 LMNtal 処理系やライブラリは，LMNtal 開発チーム全体の努力の賜物である．本研究の一部は，科学研究費補助金（基盤 (B)(2)16300009，特定 (C)(2)13324050，特定 (B)(2)14085205）の補助を得て行った．

参考文献

[1] Cardelli, L. and Gordon, A. D. : Mobile Ambients, in *Foundations of Software Science and Computational Structures*, Nivat, M. (ed.), LNCS 1378, Springer, 1998, pp. 140–155.

[2] 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. 44–60，2001.

[3] Ueda, K. and Kato, N., LMNtal: a language model with links and membranes. In *Proc. Fifth Int. Workshop on Membrane Computing (WMC 2004)*, LNCS 3365, Springer, 2005, pp. 110–125.

[4] 工藤晋太郎，加藤紀夫，上田和紀．LMNtal 処理系におけるグラフ構造の操作機能の設計と実装．情報科学技術レターズ, pp. 9–12, 2005.

[5] 乾敦行，原耕司，水野謙，上田和紀：階層グラフ書き換え言語 LMNtal 処理系とその応用例第 8 回プログラミングおよびプログラミング言語ワークショップ論文集，pp.119-133, 2006.