言語モデルLMNtal

上田 和紀　加藤 紀夫

階層的グラフの書換えを基本原理とするプログラミング言語モデル LMNtal について，設計の背景および関連研究を交えながら解説する．LMNtal（elemental と発音）は，並行計算や多重集合書換えをはじめとするさまざまな計算に関する概念の統合を目指して設計した言語モデルであり，(1) 計算モデルとして簡明であることと，(2) 多様なプラットフォームで利用可能な実用プログラミング言語のベースとなること，の両立を目指している．処理系も稼働を始めている．

本解説ではなるべく多くのプログラム例を交えながら，LMNtal の言語機能について解説するとともに，他の言語や計算モデルに見られる言語機能との関連付けを行なう．関連計算モデルに言及する部分は，並行計算および論理プログラミングに関する初歩的な知識があるとよりよく理解できるあろう．しかし，LMNtal 自体の言語機能やプログラム例は予備知識なしで十分理解できるように解説したい．

1　はじめに

21 世紀のコンピューティング環境は，広域分散化・大規模化の流れと微小化・組込み化の流れが同時並行的に進展しつつある．しかし，これらの展開を支えるソフトウェア技術，特に基本概念やプログラミング言語や理論の確立と普及は，コンピューティング環境の進展にますます引き離されつつあるように思われる．20 世紀を代表する計算モデルであるチューリングマシンやラムダ計算の意義が失われることはないであろうが，より広範囲な計算プラットフォームをカバーし，かつ一部の専門家だけでなく多くの人に受け入れられる計算モデルやプログラミング言語を確立することは，21 世紀のソフトウェア科学にとってのグランドチャレンジであろう．

LMNtal は，これまで提案されてきたさまざまな計算パラダイムをできるだけ簡明な形で統合することを目指した言語モデルである．これからの計算モデルは逐次的でない計算をモデル化することが必須となる．これまで，並行計算に関する統合理論として Action Calculi [24] などが提案されてきたが，LMNtal は特に並行計算モデルおよび多重集合 (multisets，要素の順序は意味を持たないが重複度は意味を持つ集まり) の書換えに基づく計算モデルを参考にして，それらに見られる主要な機能の統合を目指している．計算モデルでなく言語モデルと称するのは，計算モデルであると同時に，プログラミング言語としての基本機能を備えることを意味している．

LMNtal はリンクを持つノードの階層的多重集合，すなわち階層的グラフ構造の書換えに基づく並行言語であり，その名は Linked Multisets of Nodes transformation language に由来する．実は，L と N には二つの意味を重ねている．すなわち L は logical links，つまり論理変数をリンクとして用いることを

The Language Model LMNtal

Kazunori UEDA, 早稲田大学コンピュータ・ネットワーク工学科, Dept. of Computer Science, Waseda University および科学技術振興事業団, CREST, Japan Science and Technology Corporation.

Norio KATO, 早稲田大学コンピュータ・ネットワーク工学科, Dept. of Computer Science, Waseda University.

コンピュータソフトウェア, Vol.21 (2004) 掲載予定

表し，N は nested nodes，つまり階層が扱えることを表している．Elemental の原義は「四大」，つまり地・水・火・風であるが，LMNtal における四大要素は logical links，multisets，nested nodes，そして transformation ということになる．

2 簡単なプログラム例

詳細に入る前に，まずは簡単なプログラム例を二つ示そう．

2.1 リストの連結

図 1 はリストの連結プログラムとその実行の様子である．

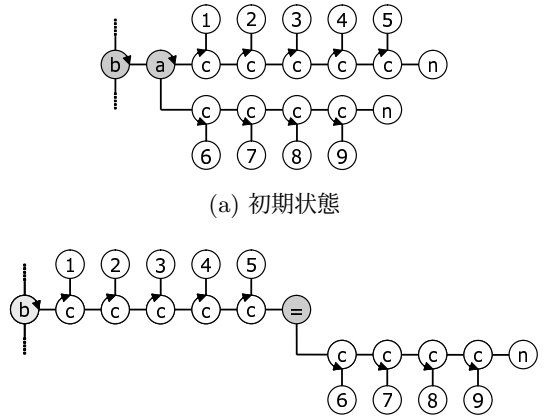
LMNtal のリストは，

- 0 個以上の `c` (cons) という名前をもつアトム，
- それと同数の要素（図 1 では自然数を名前とするアトム），および
- 1 個の `n` (nil) アトム

をリンクで接続した分子 (molecule) として表現できる．LMNtal における計算の主人公はプロセス（4.2 節）であるが，アトムはプロセスの最小単位であり，分子は一般に複数のアトムから構成されるプロセスである．リンクは 3.3.1 節で詳しく述べるが，アトムの引数どうしを無方向的に相互結合するものである．たとえば `c` は 3 価（3 引数）のアトムであり，第 1 引数として先頭要素へのリンク，第 2 引数として残りの要素のリストへのリンク，第 3 引数としてリスト全体を表すリンクを持つ．丸印の輪郭の矢印は，2 価以上のアトムの第 1 引数の場所と引数の順序とを示すためのものである．

$\texttt{A}_i$ を第 i 要素へのリンクとすると，n 個の要素をもつリストの骨格は，テキスト表現では $\texttt{c(A}_1\texttt{,X}_1\texttt{,X}_0\texttt{)}$, ..., $\texttt{c(A}_n\texttt{,X}_n\texttt{,X}_{n-1}\texttt{)}$, $\texttt{n(X}_n\texttt{)}$ となる．$\texttt{X}_0$ はこのリストを利用するプロセス（図ではプロセス `b`）につながるリンクである．

2 本のリストの連結は，`a` (append) プロセスに関する図 1 (d) の二本の書換え規則によって表現される．この規則にしたがってグラフを順次書き換えてゆくことにより，(a) の初期状態から (b) に至る．ここで `=` は二つのリンクを接続するための組込み機能であり，(b) と (c) は本質的に同一のものとみなされる[†1]．



(a) 初期状態

(b) 最終状態 1

(c) 最終状態 2

(d) 書換え規則

図 1　リスト連結プログラム

(d) の 2 本の書換え規則は，テキストでは

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

と表現される．このように LMNtal では Prolog 流の構文を援用し，論理変数を用いてリンクを表す．しかし Prolog とは (i) 左辺に複数のアトム (atomic formula) が書ける点と，(ii) 関数記号をもたない点で異なる．さらに，左辺に 2 個以上のアトムを書いた場合，それらが全部揃っている場合にのみ書換え規則が適用できるという点で，LMNtal の規則は一般に節形式 (clausal form) であるとも言えない．むしろ後述の CHR (Constraint Handling Rules) の単純化 (simplification) 規則と本質的に同じである．

†1　`=` は 2 価だが対称なので，矢印を省いている．

なお上記のプログラムは，c および n を 1 引数少ない関数記号としてもつ

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

という GHC (Guarded Horn Clauses) のプログラムと明白な対応をもっている．しかし GHC ではリストは $\mathtt{X}_0 = \mathtt{c}(\mathtt{A}_1, \mathtt{X}_1)$, ..., $\mathtt{X}_{n-1} = \mathtt{c}(\mathtt{A}_n, \mathtt{X}_n)$, $\mathtt{X}_n = \mathtt{n}$ という等式集合から構成される**データ**であって，論理型言語の特徴としてデータを書き換えるという概念が存在しないのに対し，LMNtal のリストはアトムとリンクから構成される**プロセス**である．LMNtal ではこのように，プロセスとデータとを統一的に表現する．

上記のプログラムは，実は Lafont の Interaction Nets [22] による append と非常に良く似ている．実際 Lafont は，

> “our rules are clearly reminiscent of clauses in *logic programming*, especially in the use of variables (see the example of difference-lists), and our proposal could be related to PARLOG or GHC” — [22]

と述べて論理プログラミングとの関係を予見している．しかし，Interaction Nets が 1 本のリンクで結ばれた 2 個のアトムの間の反応に限定しているのに対し，LMNtal は，3 個以上のアトムの書換えを許す点などにおいて Interaction Nets よりも一般的な枠組となっている．Interaction Nets の課した制限はプログラム記述の上では窮屈であるが，LMNtal のルールのコンパイル技法には示唆を与えるものである．

2.2 ストリームの併合

データやメッセージのリストはインクリメンタルに生成消費可能であり，**ストリーム**の表現として用いることができる．

ストリームは本質的に 1 対 1 通信のための機構であり，並行オブジェクト指向言語でのメッセージ送信が一般に多対 1 通信のための機構であることと対照をなす．ストリームに基づく並行言語で多対 1 通信を行うにはストリームの併合操作が必要となるため，その表現と効率的実現が長年の間議論と研究の対象になってきた．LMNtal では，**膜** (membrane) という概念を用いてストリーム併合を表現することができる（膜の由来については 3.3.2 節を参照）．**図 2** (a) のように，入力ストリーム（i という名前の膜内アトムにつながっていることで入力であると識別する）のうちの 1 本からメッセージが来ると，(b) のようにそれを出力ストリーム（o という名前で識別）に転送するのが併合器の役割である．ここでは膜（本解説では点線で表示）の中身はアトムの多重集合であり，リンクは膜を貫通して外部のアトム（図には示していない）とつながっている．プログラムは以下のようになる．

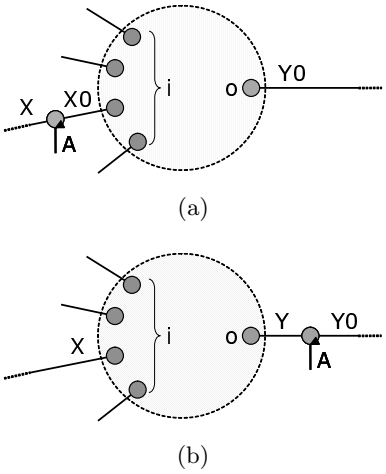


図 2　多対 1 ストリーム併合

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

テキスト表記では，膜はこのように`{ }`で表現する．`$p[|*Z]` は，膜内の残りの全要素を受け取って右辺に引き継ぐための**プロセス文脈** (process context) であり，引数の `*Z` は，受け取った「残り全要素」がもつすべての自由リンク（局所的でないリンク，この例ではすべて膜外に通じる）の束を表す．

3 LMNtal の大枠

3.1 設計目標

LMNtal が，さまざまな計算パラダイムをできるだけ簡明な形で統合することを目指した言語モデルであることは先に述べたが，具体的には次のようなことを目指している．

1. プロセス，メッセージ，データを統一的に扱う．
2. （1. の帰結であるが）プロセス構造とデータ構造を統一的に扱う．
3. リストや木などの循環しないデータ構造と循環するデータ構造とを統一的かつ容易に扱う．
4. 非同期通信と同期通信を統一的に扱う．

1. と 2. は，LMNtal においてはデータがプロセスと同様に資源としての側面をもつことを意味する．たとえば，プロセスがデータを観測するという事象はプロセスとデータの化合反応として扱われ，観測データは消費される．

3. に関して既存の宣言型言語を見ると，リストや木の扱いを得意としながら，循環構造の扱いになると非常に不器用となる．手続き型言語でもポインタ操作は難しく，誤りを引き起こしやすい．したがって，プログラミング言語設計において，循環データ構造を扱うための適切な抽象化機構の必要性は大きい．

4. の通信機構は，並行計算モデルの設計の歴史における大きな対立点であったが，LMNtal では双方を自然に表現することができる．

もちろん，LMNtal の目指す概念統合は，言語モデルとしての本質の追求を目的とするものであり，LMNtal の最適化処理系が上記 1.〜4. の諸概念を区別することや，LMNtal のための型体系が性質や用途の異なるプロセスを互いに区別することと矛盾するものではない．

LMNtal はまた，計算をグラフ書換えと見なすことができるように設計している．我々は，並行計算や動的データ構造の直感的理解のためにしばしば図を用いるが，それならば図と明らかな対応をもつことを設計指針にするのが，簡明な計算モデルを得る一つの道である．グラフ書換えやグラフ文法に関しては長い研究の歴史があり [2] [29]，多種多様な技法が提案されているが，プログラミング言語を指向した研究は多くない．LMNtal は，プログラミングのために必要かつ十分な表現力をもつ機能を盛り込むことを目指している．

3.2 背景

LMNtal の設計に至った背景には二つの流れがある．

一つは並行論理プログラミングとその並行制約プログラミングパラダイムへの発展 [31] である．1980 年代前半に提案された GHC などの並行論理型言語は，構成の動的に変化するプロセス群の記述能力を最大の特徴とし，かつ実用に供していた．論理変数（単一代入変数）を通信チャネルに利用して，π 計算におけるチャネルモビリティと同じ意味でのモビリティを実現していた．並行論理プログラミングはその後，有限木以外にもさまざまなデータ領域を許す並行制約プログラミングへと一般化されたが，多重集合はなかでも重要なデータ領域と認識され，並行制約言語 Janus [30] でも採用された．

もう一つの流れは，並行論理プログラミングから制約充足系記述言語 Constraint Handling Rules (CHR) [13] への発展である．CHR は，規則の頭部にアトムの多重集合を書けるようにするなどの拡張を施して，制約充足系 (constraint solver) をユーザが定義するための言語としたものである．また，論理変数や論理変数を含むデータ構造をアトムの引数にとれるおかげで，CHR は多重集合書換え言語のなかでもっとも強力なものとなっている．CHR では，多重集合の書換え (rewriting) のみならず，「要素 $c_1, \ldots, c_n$ が存在する場合は c も追加する」という形の伝播 (propagation) ないしは閉包 (closure) 操作も記述できる．この伝播機能のために，CHR は多重集合書換え言語だけでなく集合書換え言語としての側面も有している [18]．

並行制約プログラミングと CHR は，多重集合をサポートする点で共通しているが，前者が扱うのは一級データ（変数に代入したり，手続きの引数にできるデータ）としての多重集合，後者が扱うのはアトムの多重集合である．そこで一方を他方に埋め込んで統合できないだろうかという興味が湧いてくる．LMNtal は，これに対する回答を探す中で，膜の概念の導入によって両者を統合することを着想したものである．

3.3 LMNtal の言語要素

LMNtal の構文や意味を正確に与えるに先立ち，その四大要素である logical links，multisets，nested nodes，transformation を概観しておこう．

3.3.1 Logical Links

互いに通信しあうプロセス群は，プロセスをノードとし，通信チャネルをリンクとするグラフとして表現できる．同様に，動的データ構造もノードとリンクを用いて表現できる．LMNtal ではこれらを統一的に扱う．リンクは論理的に隣接するプロセス間の 1 対 1 通信チャネルを表すとともに，データセルの間の隣接関係も表す．

並行計算における二つの重要な通信メカニズムは，π 計算に見られるような**名前に基づく通信**と，並行制約プログラミング [32] に見られるような**論理変数（単一代入変数）に基づく通信**である．前者は，チャネルやメッセージやメッセージ受信のための仮引数などのプロセス間通信の諸機構を，名前 (name) という統一概念で説明することを特徴としている．後者は，並行プロセスが共有する単一代入変数の値に関する制約条件 (constraints) を付加（tell, 送信に対応）したり観測（ask, 受信に対応）したりすることによって通信を行うことを特徴としており，**制約に基づく通信**ともいう．LMNtal のリンクは，

- リンクづたいにメッセージを送信すると送信前後でリンクのアイデンティティが変化する点と，
- リンクがすべてプライベートである（つまり第三者はアクセスできない）点

において，オリジナルの π 計算と異なっており，論理変数に基づく通信に近い．しかし，LMNtal のリンクは具体化 (instantiate) するという概念を持たないので**無代入変数** (zero-assignment variable) とも呼ぶことができる．この点で，LMNtal のリンクは論理プログラミングや CHR における変数とも異なっている．

LMNtal のリンクは，化学結合と同じように無方向的である．しかし，リンクでつながったパートナーを探すのに常に一定の方向にしかリンクをたどらないようなプログラムでは，型体系を用いてその方向を表現したり推論（再構成）したりできるであろう．

3.3.2 多重集合とその階層化

これまでに，多重集合の概念をもつさまざまな計算モデルが提案されてきた．

- もっとも初期の代表例はペトリネットである．ペトリネットはプレースおよびトランジションと呼ばれる二種類のノードをもつグラフ（正確には重みつき有向 2 部グラフ）であり [26]，あるトランジションが発火 (fire) するとは，そのトランジションの入力プレース（一般に複数）に存在するトークンを指定個数ずつ消費して，出力プレースにトークンを指定個数ずつ追加することである．したがってペトリネットは本質的に，プレースの多重集合の書換え系である．
- 人工知能分野における問題解決機構として長い歴史をもつプロダクションシステムも，多重集合書換え系の一つである．RETE [10] をはじめとする多重集合のパターンマッチ（ルール左辺とワーキングメモリとのマッチング）のコンパイル方式，ルールの実行制御，並列化などの多くの研究がなされてきた [19]．
- 一般に並行計算モデルは，並行プロセス群が多重集合を形成するので，必然的に多重集合の概念を有している．並行プロセス群が多重集合を形成することと，プロセス間通信に多重集合を利用することとは独立であるが，Linda [15] のタプルスペースは，プロセス間通信における多重集合の利用例としてよく知られる．当初のタプルスペースは（少なくとも概念的には）大域的な一枚岩であったが，これを多重化，局所化してモバイル計算に適した機能とする試みもある [7] [27]．
- 多重集合の書換えに基づく代表的な計算モデルとしては Gamma [3] および Chemical Abstract Machine [4] などがある．どちらも化学反応に着想を得ているが，前者は超並列計算を，後者は並行計算を指向していると言える．同様に化学反応に着想を得た Chemical Casting Model (CCM) [21] は，自己組織化原理に基づく問題解決を指向している．
- 線形論理に基づくプログラミング言語 [23] は，原子論理式の多重集合という概念を何らかの形で持っており，これを継承やメッセージ通信などのプログラミング概念の表現やアルゴリズムの表現にうまく利用できることを標榜している．たとえば最初の線形論理型言語 LO [1] は，フィールドの多重集合として表現したオブジェクトの書換

え系と見なすことができる．Lolli のように，線形含意演算子を多重集合要素の追加と取出しに利用する言語もある [17].

- 多重集合書換えは，セキュリティプロトコルの記述と解析にも応用されている．文献 [6] が提案する言語は原子論理式の多重集合の書換え言語である点で LMNtal と共通であるが，局所論理変数の動的生成機能をノンス (nonce) の表現に利用している．

しかし，Gamma, Linda, CHR をはじめとする計算モデルや言語の多くは，多重集合を一級データとせず，他のデータ構造と異なる扱いをしてきた．多重集合というデータ構造を一級市民として許した言語としては，並行制約言語の一つである前述の Janus がある．一級市民にすることの利点は，多重集合の構造化やモバイル化など，さまざまな柔軟な扱いが可能になる点である．

LMNtal は，プロセスの多重集合を膜で囲ったものがひとまとまりのプロセスになるという形で，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．計算の局所化は，ある階層に置いた書換え規則はその階層にのみ適用されるという形で実現される．

多重集合の階層化は，Chemical Abstract Machine (CHAM), Ambient 計算 [5] や bigraphical reactive system (BRS) [25] [20] をはじめとして，膜計算 (membrane computing) [28] という一種の形式言語モデルや，知識表現 [9] などの分野でも見られる．この中で膜という用語と概念をいち早く取り入れたのは CHAM であるが，膜を用いて反応を局所化するということと，膜ごとに異なる反応規則を定義できることとは同じではない．後者の概念を有するのは，膜計算を別にすると，Ambient 計算，BRS, LMNtal など，広域計算の表現を指向しているモデルたちである．さらに LMNtal と BRS は，リンクによる構造化機能と多重集合階層化の双方を備えている点で他よりも強力である．LMNtal と BRS との比較は 6.5 節で行う．

3.3.3 書換え

計算モデルは，(i) λ 計算や π 計算のように，簡約の対象となる式自体がプログラムと見なされるモデルと，(ii) 論理プログラミングや項書換え系のように，式に対する書換え規則の集合がプログラムと見なされるモデルとに大別できる．両者の違いは，計算の実行に伴ってプログラムが消費されるか否かの違いであるとも言える．LMNtal は，消費されない書換え規則を持つという点で (ii) に近いが，書換え規則がプロセスと別個に与えられるのでなくてプロセスの一部をなすという点で (i) の側面もあわせ持つ．書換え規則は，プロセス中に存在してプロセスの反応を可能にする触媒であると考えるとわかりやすい．

グラフ文法とグラフ書換えの分野では階層グラフの書換えも研究されているが [8] ，LMNtal は，プログラミング言語の立場から，膜を通過するリンクの管理機能を注意深く設計した点を特徴とする．

書換え規則は多重集合要素間の反応を規定するが，リンクでつながった要素間の反応は，つながっていない要素間の反応よりも効率的なパートナー検索が一般に可能である．

LMNtal は，チャネルモビリティとプロセスモビリティの両方を備える．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．

4 LMNtal の構文

4.1 リンクと名前

まず最初に，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．

4.2 構文規則

LMNtal の主要な構文カテゴリは，**プロセス** P および**プロセステンプレート** T の二つである．前者は LMNtal プログラムの実行で作られるプロセスを表現するものである．後者はプロセスの書換え規則の表現

P ::=	$\mathbf{0}$	(空 / null)
	$p(X_1,\ldots,X_m)$	($m \geq 0$) (アトム / atom)
	P,P	(分子 / molecule)
	$\{P\}$	(セル / cell) †
	T :- T	(ルール / rule)
T ::=	$\mathbf{0}$	(空)
	$p(X_1,\ldots,X_m)$	($m \geq 0$) (アトム)
	T,T	(分子)
	$\{T\}$	(セル) †
	T :- T	(ルール)
	@p	(ルール文脈) †
	\$$p$[$X_1,\ldots,X_m$\|$A$]	(プロセス文脈) †
	$p(*X_1,\ldots,*X_m)$	($m > 0$) (アトム集団 / aggregate) †
A ::=	[]	(空) †
	$*X$	(リンク束 / bundle) †

図 3　LMNtal の構文

に用いるもので，局所文脈（特定のセルの内部での文脈）を扱う機能をもつ．

LMNtal の構文は**図 3** の通りである．ただし構文上のあいまいさをなくす目的で，必要に応じて括弧 ( ) を用いるものとする．分子を構成する “,” は，ルールを構成する :- よりも強い結合力をもつものとする．P および T にはいくつかの構文条件が存在するが，それについては 4.2.1 節および 4.2.2 節で詳しく説明する．

どのルールにも含まれない部分を**ルール外**という．

セルは入れ子になることができるが，セル $\{P\}$ または $\{T\}$ において，入れ子になったセルに含まれない部分をその $\{P\}$ または $\{T\}$ の**最外部**と呼ぶ．

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．

以下，4.2.1 節でルール以外の部分を説明し，4.2.2 節でルールの構文の詳細を説明する．

4.2.1　プロセス

プロセス P は，リンク条件と呼ばれる以下の構文条件を満たさなければならない．

> **リンク条件：** プロセスのルール外には，同じリンクが 2 回を越えて出現してはならない．

プロセス P に 1 回だけ出現するリンクを P の**自由リンク**と呼び，P に 2 回出現するリンクを P の**局所リンク**と呼ぶ[†2]．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，$\mathbf{0}$ は中身のないプロセス，$p(X_1,\ldots,X_m)$ は m 本のリンクを引数にもつアトム，P,P はプロセスの並列合成，$\{P\}$ 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．アトム 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．セルの自由リンクは，セルの膜からではなくて内部の構成要素から出ていることに注意する．

空，アトム，セル，ルールは互いに排他的な概念であるが，これらと分子とは必ずしも排他的ではない．なぜならば，構造合同関係によって，どんなプロセス P もそれを構成要素として持つ分子 $\mathbf{0},P$ と同一視されるからである．

プロセス P_1 と P_2 から分子 P_1,P_2 を合成しようとする場合，両者に同名のリンクが出現していると，リンク条件に抵触する場合がある．しかし，P_1 と P_2 に合計 3 回以上出現するリンクは，P_1 と P_2 の少なくとも一方に 2 回出現する．2 回出現するリンクは局所リンクなので，α 変換によって他に出現しないリンクに置き換えることができる（5.1 節）．したがって，分子 P_1,P_2 の合成がリンク条件によって本質的に妨げられることはない．なお，ルール中のリンクをリンク条件のカウント対象に含めないのは，それらはすべてルールに局所的であるからである．

4.2.2　ルールとプロセステンプレート

ルールは T :- T という構文をもつが，左と右の T をそれぞれ**左辺**，**右辺**と言う．直感的には，左辺は書き換えるプロセスがマッチすべきテンプレートを，右

†2　次節で述べるルール中のリンクの回数条件から，P が自由リンクをもつ場合は必ず P のルール外に存在する．

辺は書換え後のプロセスを指定する．

A をプロセス文脈の**剰余引数**と呼ぶ．直感的には，リンク束 $*X$ は不特定多数本のリンクを表す記法である．

プロセスおよびプロセステンプレート内のルールは，以下の構文条件を満たさなければならない．

構文条件：

1. ルールは，ルールの左辺に出現してはならない．
2. アトム集団は，ルールの左辺に出現してはならない．
3. ルールの左辺に出現するルール文脈およびプロセス文脈は，セルの中に出現しなければならない．

1 番目と 2 番目の条件はそれぞれ，ルールおよびアトム集団（が表すアトム多重集合，5.2 節参照）の動的生成は許すが，既存のルールおよびアトム集団（が表すアトム多重集合）の動的解析は許さないことを意味している．

ルール文脈やプロセス文脈は，ルール左辺に指定されセルの中の，明示的に指定されたアトムやセル以外の要素とマッチするためのものであり，3 番目の構文条件はそのことに由来する．ルール文脈はセルの中のすべてのルールの多重集合とマッチし，プロセス文脈はルール以外のプロセスのうち，明示的に指定された部分を除く全体とマッチする．プロセス文脈の引数は，自由リンクの出現に関する制約条件を指定するものであり，直感的な意味と簡単な例は本節で，正確な意味は 5 節で説明する．プロセス文脈 `$`p`[...]` の `$`p の部分を**プロセス文脈名**と呼ぶ．

ルールは，リンクその他の構文要素に関する以下の回数条件を満たさなければならない．

回数条件：

1. ルールに出現するリンクおよびリンク束は，そのルールにちょうど 2 回出現しなければならない．
2. ルール左辺の 1 つのプロセス文脈の引数に出現するリンクは互いに異なっていなければならない．
3. ルール左辺のリンク束は互いに異なっていなければならない．
4. ルールに出現するルール文脈名およびプロセス文脈名は，そのルールの左辺にちょうど 1 回出現しなければならず，そのルール内のほかのルールに出現してはならない．
5. ルール左辺の各セルの最外部には，ルール文脈とプロセス文脈はそれぞれ 1 回を越えて出現してはならない．

回数条件 1. は，ルールが自由リンクを持たないことを意味している．

回数条件 2. の「プロセス文脈の引数に出現するリンク」は，プロセス文脈にマッチするプロセスがもたなければならない自由リンクを指定するものであり，そのことからこの条件が課されている．

回数条件 3. の「ルール左辺のリンク束」は，構文条件 2. からプロセス文脈に出現するものに限られる．プロセス文脈 `$`p`[`$X_1, \ldots, X_m$`|*`X`]` の中の $*X$ は，マッチするプロセスの $X_1, \ldots, X_m$ 以外の自由リンクを受け取るためのものであり，「互いに異なっていなければならない」は，リンク束が，二つの自由リンク集合の同一性を比較する機能をもたないことを意味する．

回数条件 4. の「左辺での出現」は，ルール文脈やプロセス文脈がルール適用時に実際のルール多重集合やプロセスを受け取る必要があることを意味し，「ちょうど 1 回」は，複数の文脈の同一性を検査する機能をもたないことを意味する．なお，ルール文脈とプロセス文脈のルール**右辺**での出現については，回数条件がないことに注意してほしい．

回数条件 5. は，ルール文脈やプロセス文脈が受け取る値が一意的に決まるようにするための条件である．

次にいくつかの整合性条件を示す．

整合性条件：

1. ルール内では，同じ名前のプロセス文脈の剰余引数に空（`[]`）とリンク束の両方が存在してはならない．
2. ルール内では，同じ名前のプロセス文脈の引数の個数 m はすべて一致していなければなら

ない.

3. ルールにおいて，同じリンク束が 2 つのプロセス文脈の剰余引数として出現するならば，それらのプロセス文脈は同じ名前を持っていなければならない.
4. ルール内のアトム集団 $p(*X_1, \ldots, *X_m)$ $(m > 0)$ に対してプロセス文脈名 $\$q$ が 1 つ存在して，どの $*X_i$ も，名前 $\$q$ をもつそのルール内のプロセス文脈のどれかに，剰余引数として出現しなければならない.

たとえばルール

```
{exch,$a[X,Y|[]]} :- {$a[Y,X|[]]}
```

は，整合性条件 1. と 2. を満たしており（整合性条件 3. は自明）,「アトム exch を最外部にもち，自由リンクをちょうど 2 本もつセルがあったら，2 本の自由リンクをクロスさせ，アトム exch を消去する」ことを表している．なおこのルールは，後述の省略規則を用いると

```
{exch,$a[X,Y]} :- {$a[Y,X]}
```

と略記できる.

また,

```
{kill,$a[|*X]} :- killed(*X)
```

は整合性条件 3. を満たしており（整合性条件 1. と 2. は自明）,「アトム kill を最外部にもつセルがあったら，そのセルを消滅させて，セルの膜を貫通していた個々のリンクを 1 価のアトム killed で終端する」ことを表している.

ルール L :- R に出てくるリンクのうち，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を表す.

上記の各種条件から，LMNtal では，形が動的に決まるルールの動的生成はできないかわりに，形が静的に決まるルールを動的に追加することや，膜内のすべてのルールの集合をルール文脈を使ってコピーしたりよそへ移動したりすることが可能である．すなわち LMNtal は，ルールの膜単位でのコンパイルを可能にしつつ，一定の高階機能を提供している.

なお，LMNtal をグラフ書換えモデル [29] と比較する場合に注意を要するのは，LMNtal におけるアトムのようにきまった本数の腕をもつものは，グラフ書換えの文献ではノードではなくて hyperedge と呼ばれる点である．逆に LMNtal のリンクは，hyperedge 書換えモデルにおける（2 価の）ノードに対応する．Hyperedge 書換えに関しても形式言語の立場から多くの研究があるが [16]，ほとんどは単一の hyperedge を hypergraph （LMNtal のプロセスに対応）に書き換える体系に関するものである．これに対して LMNtal では，アトムでないプロセスを書き換えることが本質的に重要である.

4.3 略記法

以下の略記法を使用することができる.

1. ()（0 価アトムの括弧）および `|[]` は省略してよい.
2. `[|`$*X$`]` は $(*X)$ と略記してよい.
3. ルールに同一の `$`p`[|`$*X$`]` が 2 回出現するならば，両方を同時に `$`p と略記してよい.
4. $p(s_1, \ldots, s_m), q(t_1, \ldots, t_n)$ は，s_m と t_i が同じリンクならば，$q(t_1, \ldots, t_{i-1}, p(s_1, \ldots, s_{m-1}), t_{i+1}, \ldots, t_n)$ と略記してよい.

たとえばリスト

$$\texttt{c(A}_1\texttt{,X}_1\texttt{,X}_0\texttt{)}, \ldots, \texttt{c(A}_n\texttt{,X}_n\texttt{,X}_{n-1}\texttt{),n(X}_n\texttt{)}$$

は，4. の略記法を利用すると

$$\texttt{c(A}_1\texttt{,c(A}_2\texttt{,...,c(A}_n\texttt{,n)...),X}_0\texttt{)}$$

となる．さらにこれは 5.1 節に示す構造合同関係によって

$$\texttt{X}_0\texttt{=Y, c(A}_1\texttt{,c(A}_2\texttt{,...,c(A}_n\texttt{,n)...),Y)}$$

と等価であり，この Y に再び略記法を適用すると

$$\texttt{X}_0\texttt{=c(A}_1\texttt{,c(A}_2\texttt{,...,c(A}_n\texttt{,n)...))}$$

となる.

5 操作的意味論

本節では，プロセスの構造合同 (structural congruence) 関係 $\equiv$ を定めて論じたあと，遷移関係 $\longrightarrow$ を与える.

5.1 構造合同

プロセス間の関係 ≡ を，**図 4** の規則を満たす最小の同値関係として定義する．つまり，≡ の関係にあるプロセスは本質的に**等価**なプロセスと見なし，0 ステップで互いに変換可能であると見なす．

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**を表す．

(E1)–(E3) は，分子が多重集合であることの特徴づけである．

(E4) は局所リンク名のつけかえ，すなわち α 変換を許すものである．Y は任意のリンクであるが，$P[Y/X]$ に対するリンク条件から，P に出現しないリンクを選ぶ必要がある．

(E5)–(E6) は ≡ を合同関係 (congruence) にするための構造規則である．

(E7)–(E9) は = に関するもので，(E7) は自己完結的ループと **0** との等価性を，(E8) は = の対称性を表す．(E9) は = の吸収規則で，= アトムが他のアトム (= であってもよい）に吸収されうることを示す．≡ は対称なので，(E9) は，アトムが = を逆に放出できることも表している．

= は論理型言語の単一化 (unification) を源としている．単一化をチャネルの相互接続に用いるアイデアは，並行論理プログラミングで長らく実用に供されてきた．名前に基づく通信（3.3.1 節）においては当初は同様の概念がなかったが，名前どうしの融合 (fusion) を明示的に扱う試みも始まっている [14]．

5.2 遷移規則

LMNtal の計算とは，プロセスを，そのプロセスと（入れ子になった膜構造の）同じ「場所」に置かれたルールによって書き換えてゆくことである．

プロセス間の遷移関係 $\longrightarrow$ を，**図 5** の規則を満たす最小の関係として定義する．ただし各規則は略記法 4. の適用前の形で適用するものとする．いずれの規則においても，関係 $\longrightarrow$ の右辺はプロセスのリンク条件を満たしていなければならないことに注意する．

6 つの規則のうち，(R1)–(R3) は構造規則である．

(R1) は，計算を局所的な書換え条件に基づいて並行に進めることができることを表す．LMNtal のもつ細粒度の並行性はこの規則に由来する．

(R2) はセル内の計算を，セルの外側と独立に進めることができることを表す．ただしセル内で独立して計算を進めるためには，セルが内部に固有のルールを持っていなければならない．ルールを持たないセルの計算は，外側のルールが制御する．内部にルールを持つセルは一種の自律機械であるが，大多数の応用では，自律機械といえども外部との通信を必要とする．この通信のプロトコルを定め，セルに対して実際にアトムの出し入れを行うのは，セル外部のルールである．つまり各セルは，外界の定めるプロトコルの下で自らの計算を進めることになる．

(R3) は構造合同を遷移関係に取り込むための規則である．

(R4) と (R5) は = アトムと膜の相互作用であり，通信および中継の局所化に寄与する．

(R4) はあるセルの中の = がセル外の 2 地点を中継しているとき，その中継をセルの外に追い出せることを表す．(R4) の適用によって，このセルの自由リンクは 2 本減る．

(R5) は，= の一端がセル内に通じているとき，= をセル内に移動できることを表す．(R5) を適用すると，= の他端が同じセル内に通じているときはセルの自由リンクが 2 本減る．通じていなければ自由リンクの本数は変化しない．

(R6) が LMNtal で中心となる規則である．θ は，

- ルール左辺にプロセス文脈やルール文脈が存在する場合に，それらがどのようなプロセスやルール多重集合にマッチしたか
- アトム集団がどのようなアトムの多重集合を表すか

を表現する手段であるが，Flat LMNtal では（プロセス文脈，ルール文脈，アトム集団が出現しないので）θ は不要となって，規則は

$$(\mathrm{R6}')\quad T, (T :\!- U) \longrightarrow U, (T :\!- U)$$

となる．(R6′) は，膜に隔てられずに共存するプロセスとルールとの反応を記述したものであり，プロセスはルールにしたがって書き換わるがルール自身は触媒のように保存されることを表す．

(E1)	$\mathbf{0}, P$	$\equiv$	P	
(E2)	P, Q	$\equiv$	Q, P	
(E3)	$P, (Q, R)$	$\equiv$	$(P, Q), R$	
(E4)	P	$\equiv$	$P[Y/X]$	ただし X は P の局所リンク
(E5)	$P \equiv P'$	$\Rightarrow$	$P, Q \equiv P', Q$	
(E6)	$P \equiv P'$	$\Rightarrow$	$\{P\} \equiv \{P'\}$	
(E7)	$X = X$	$\equiv$	$\mathbf{0}$	
(E8)	$X = Y$	$\equiv$	$Y = X$	
(E9)	$X = Y,\ P$	$\equiv$	$P[Y/X]$	ただし P はアトム，X は P の自由リンク

図 4　LMNtal プロセスの構造合同

(R1) $\dfrac{P \longrightarrow P'}{P, Q \longrightarrow P', Q}$　　(R2) $\dfrac{P \longrightarrow P'}{\{P\} \longrightarrow \{P'\}}$　　(R3) $\dfrac{Q \equiv P \quad P \longrightarrow P' \quad P' \equiv Q'}{Q \longrightarrow Q'}$

(R4)　$\{X = Y, P\} \longrightarrow X = Y, \{P\}$　（ただし X および Y は P に現れず，X と Y は異なる）

(R5)　$X = Y, \{P\} \longrightarrow \{X = Y, P\}$　（ただし X は P のルール外に現れる）

(R6)　$T\theta, (T$ `:-` $U) \longrightarrow U\theta, (T$ `:-` $U)$　（ただし $T\theta$ はルール外に `=` を含まない）

図 5　LMNtal プロセスの遷移関係

(R6′) でのプロセスとルール左辺とのマッチングは，(R3) を援用してルールに α 変換を施すことによってとられる．では，アトム `p(A,A)` をルール

```
p(X,Y) :- q(X,Y)
```

を用いて書き換えることができるであろうか？ ルールは

```
p(A,A) :- ...
```

という形に α 変換することはできないので一見不可能に見えるが，`p(A,A)` は実は `p(A,B),A=B` （`B` は新しいリンク）と等価である．したがって `p(A,A)` は

```
p(A,A)  ≡  p(A,B), A=B
        ⟶  q(A,B), A=B
        ≡  q(A,A)
```

と書き換えることができる．このように，`=` に関する構造合同規則によって，リンクの非単射的なマッチングがとれるわけである．

書換え後の $U, (T$ `:-` $U)$ はリンク条件を満たさなければならないが，書換えによって新たな局所リンクが生成される場合，あらかじめ T `:-` U を適切に α 変換しておくことで，既存のリンクとの衝突を避けることができる．

規則 (R6) の θ は，以下の条件 1.〜3. を満たす置換である．ただし置換は，β_i/α_i の形の置換要素（α_i を β_i で置き換えることを意味する）の有限集合として表現するものとする．また，条件 3. の中の v は，リンク名と自然数の組からリンク名への一対一写像とする．

1. θ の定義域は，左辺 T の中，または右辺 U のルール外に出現するすべてのルール文脈およびプロセス文脈およびアトム集団の集合に一致する．
2. 左辺 T 内の `@`p に対して，あるルールの列 P が存在して $P/$`@`$p \in \theta$ を満たす．
3. 左辺 T 内の `$`p`[`$X_1, \ldots, X_m$`|`A`]` に対して，以下の (i)–(iii) が成り立つ．ここで P は，$\{X_1, \ldots, X_{m+n}\}$ ($n \geq 0$，ただし $A =$ `[]` ならば $n = 0$) を自由リンクの全体集合，$\{Z_1, \ldots, Z_\ell\}$ を局所リンクの全体集合とするプロセスで，P 内

のルールはすべてセルの中に存在するものとする．また右辺 U 内のプロセス文脈名 $\$p$ の各出現に対して，互いに異なる自然数を割り当てておく．

(i) T の自由リンクは P に出現しない．

(ii) $A = []$ のとき：

(a) $P/\$p[X_1, \ldots, X_m] \in \theta$

(b) U 内の自然数 h が割り当てられた $\$p[Y_1, \ldots, Y_m]$ に対して

$$P[v(Z_1,h)/Z_1, \ldots, v(Z_\ell,h)/Z_\ell, Y_1/X_1, \ldots, Y_m/X_m] \,/\, \$p[Y_1, \ldots, Y_m] \in \theta$$

(iii) $A = *V$ のとき：

(a) $P/\$p[X_1, \ldots, X_m | *V] \in \theta$

(b) すべての $i \in \{1, \ldots, n\}$ に対して $v(V, i) = X_{m+i}$

(c) U 内の自然数 h が割り当てられた $\$p[Y_1, \ldots, Y_m | *W]$ に対して

$$P[v(Z_1,h)/Z_1, \ldots, v(Z_\ell,h)/Z_\ell, Y_1/X_1, \ldots, Y_m/X_m, v(W,1)/X_{m+1}, \ldots, v(W,n)/X_{m+n}] \,/\, \$p[Y_1, \ldots, Y_m | *W] \in \theta$$

(d) U 内かつ子ルール外にあって，ある V_i が V であるような $q(*V_1, \ldots, *V_k)$ に対して

$$(\, q(v(V_1,1), \ldots, v(V_k,1)),\ \ldots,\ q(v(V_1,n), \ldots, v(V_k,n))\,) \,/\, q(*V_1, \ldots, *V_k) \in \theta$$

直感的に述べると，ルール左辺のプロセス文脈 $\$p[X_1, \ldots, X_m | A]$ の引数 $X_1, \ldots, X_m$ は，その文脈が持っていなければならない自由リンクを指定している．ただし X_i が左辺に他に出現しない場合は，任意の自由リンクを表す．剰余引数 A が $*V$ の場合，それは $X_1, \ldots, X_m$ 以外の 0 本以上の自由リンクの束を表し，$[]$ の場合は $X_1, \ldots, X_m$ 以外に自由リンクがないことを表す．

ルール右辺に同名のプロセス文脈 $\$p[X'_1, \ldots, X'_m | *V']$ があるとき，左辺のプロセス文脈がマッチしたプロセスと同型のプロセスが作られ，その自由リンクには $X'_1, \ldots, X'_m$ および $*V'$ の各リンクが接続される．

アトム集団 $p(*X_1, \ldots, *X_m)$ は，リンク束 $*X_i$ が表すリンクの本数と同じ個数の m 価アトムを表す．各 $*X_i$ はプロセス文脈名に関して同じ出自をもたなければならない．つまり各 $*X_i$ のもう一方の出現は，すべて同じ名前のプロセス文脈の中になければならない．回数条件 4. より，$*X_1, \ldots, *X_m$ のうちちょうど 1 個がルールの左辺に出現することに注意する．

例を示そう．プロセス

```
kill(S), {i(S),a(X),b(Y,Z),c(Z,U)}
```

をルール

```
kill(S), {i(S),$p[|*P]} :- killed(*P)
```

と反応させると，プロセス文脈 `$p[|*P]` は `a(X),b(Y,Z),c(Z)` に，アトム集団 `killed(*P)` は `killed(X),killed(Y)` に対応づけられ，反応後のプロセスは

```
killed(X), killed(Y)
```

となる．この例で，セルは 0 本以上の自由リンクをもつプロセス構造を管理するためのもので，セル内の `i( )` はこのグラフに対する外部からの操作を伝えるためのメッセージチャネルのタグと考えることができる．上のルールは，チャネルに `kill` メッセージが来たら管理範囲のプロセスを消去し，プロセスの持っていた自由リンクを `killed` で終端することを表している．

また

```
cp(S,S1,S2), {i(S),a(X),b(Y,Z),c(Z)}
```

をルール

```
cp(S,S1,S2), {i(S),$p[|*P]} :-
        {i(S1),$p[|*P1]},
        {i(S2),$p[|*P2]},
        cp(*P,*P1,*P2)
```

と反応させると，セルの複製が起きて

```
{i(S1),a(X1),b(Y1,Z1),c(Z1)},
{i(S2),a(X2),b(Y2,Z2),c(Z2)},
cp(X,X1,X2), cp(Y,Y1,Y2)
```

となる．これは，`cp` 指令が来たら，対象のセルを複製することと，複製されたセル側の自由リンクとオリジナルの自由リンクとをそれぞれ 3 価の `cp` ノードで

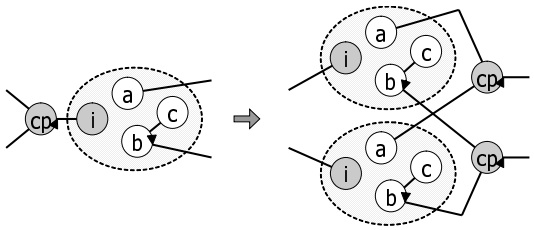


図 6　プロセス文脈とアトム集団を利用したセル複写

結ぶことを表している（**図 6**）.

(R6) は，左辺のルール外に = を含まないという付帯条件をもつ．= の機能はリンクを中継もしくは相互接続することであるが，= は (E6) および (R4)(R5) によって移動と吸収ができるようになっている．したがって，= がないプロセスを正規形とみなして (R6) のマッチング対象とすることにより，ルール左辺に = を書くことを排除できる．つまり，付帯条件の帰結として，= の意味を再定義することができなくなっている．

6　プログラム例

冒頭の二つの例に続き，いくつかの例を紹介しよう．

6.1　プロセス移送

一本の通信リンクを共有する二つのセルを考える．両者はほとんどの時間，個々のセル内のルールを使って独立に計算を進めるが，ときどき，リンクづたいにプロセスやメッセージを移送するものとする．移送のルールやプロトコルを定めるのは一段階上位の層である．ここでは**図 7** のように，2 価のアトム g（第 1 引数が出発地側，第 2 引数が目的地側へのリンク）を内部にもつセルを用意すると上位層がそれを移送するものとする．

```
{$s[S0|*S], @s, {g(S0,D0),$m[|*M],@m}},
{$d[D0|*D], @d} :-
    {$s[S|*S], @s},
    {{s(S,D),$m[|*M],@m}, $d[D|*D], @d}
```

{g(S0,D0),$m[|*M],@m} を囲む膜は，移送すべき資源を明示するためのものである．@m 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，移送後は新たな別のリンクになってい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．到着後のメッセージを g (go) から s (stop) に変えるのは逆送防止のためである（図 7 の対称性からわかるように，メッセージタグを変更しないと逆送が起きる）．なお，受信後の膜は，受信側で必要に応じて消去する．

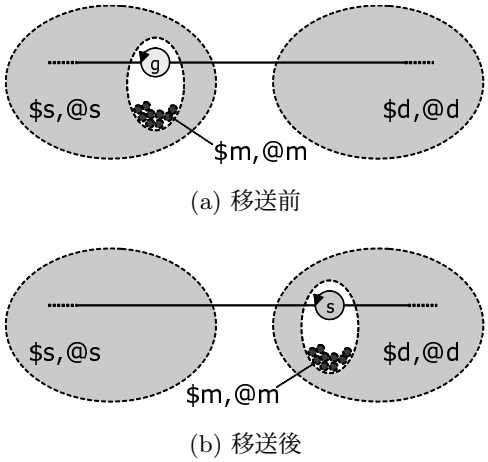


図 7　プロセス移送

6.2　循環データ構造

LMNtal は循環データ構造の扱いを得意とする．たとえば，双方向環状バッファは

b(S,X_n,X_0),n(A_1,X_0,X_1),...,n(A_n,X_{n-1},X_n)

というリンク構造をもつプロセスで表現できる．ここで b はヘッダアトム，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，**図 8** のように 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(S,S0), n(A,L,C0), b(S0,C0,R) :-
        b(S,L,C), n(A,C,R)
right(S,S0), b(S0,L,C0), n(A,C0,R) :-
        n(A,L,C), b(S,C,R)
put(A,S,S0), b(S0,L,R) :-
        n(A,L,C), b(S,C,R)
...
```

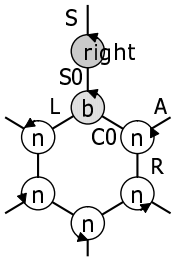


図 8　循環データ構造

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 Types [12] がある．Shape Types は，Gamma モデルの多重集合に要素間の関係を表現するための述語記号を導入した Structured Gamma [11] をベースとして，サイクルを含む動的データ構造の型記述の枠組みを提案したものである．述語記号を要素間の有向リンクの名前と解釈すると，Shape Types はポインタが形作るグラフ構造のノードを変数で，リンクを名前で表現していることになり，LMNtal とは双対な方法となっている．

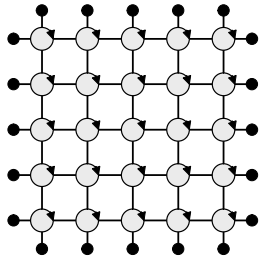
6.3　構造形成

図 9 (a) のような 2 次元格子構造を作りたいとする．このようなリンク構造を手続き型プログラムで実現するのはかなり面倒である．LMNtal では，初期状態として図 9 (b) のように縦横の一辺ずつが与えられれば，あとは図 9 (c) のルール一本で欠損部分が順次補われてゆき，長方形格子が完成する．

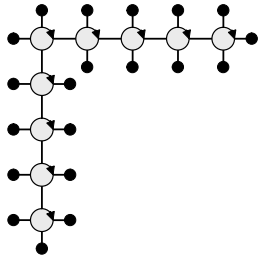
ルールのテキスト表現は，白丸を c，黒丸を n とすると以下のようになる．

```
c(X01o,Y01o,X00,Y00), c(X02,n,X01o,Y10),
c(n,   Y02, X10,Y01o) :-
   c(X01,Y01,X00,Y00), c(X02,Y11,X01,Y10),
   c(X11,Y02,X10,Y01), c(n,  n,  X11,Y11)
```

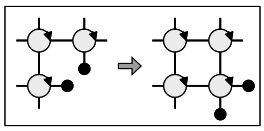
これは自己組織化に基づく構造形成の単純な例であるが，同様の原理に基づいて，ASCII アートとして書いたグラフ構造のボトムアップパーザも容易に記述できた．これは，テキストファイルから ASCII 文字を要素とする 2 次元格子を形成し，次に要素の消去や融合を局所的に行いつつ目的のグラフ構造を得るというものである．



(a) 目標状態



(b) 初期状態



(c) 書換え規則

図 9　2 次元格子の生成

6.4　マップ関数

リストの各要素に至るリンクに，2 価のアトムをはさみ込むことを考える．たとえば

```
X = c(A1,c(A2,c(A3,n)))
```

というリスト X （通常記法では c(A1,X1,X),c(A2,X2,X1),c(A3,X3,X2),n(X3)）を

```
X = c(s(A1),c(s(A2),c(s(A3),n)))
```

に変換したい．あるいは一般に，自由リンクを 2 本持つプロセス，たとえば $\mathtt{s(I}_i\mathtt{,M}_i\mathtt{)}$,$\mathtt{s(M}_i\mathtt{,O}_i\mathtt{)}$ の形の分子をはさみ込んで

```
X = c(s(s(A1)),c(s(s(A2)),c(s(s(A3)),n)))
```

などと変換したい．これらの例では，ノード s または s からなるプロセスを必要な数だけ複製する機能，およびオリジナルコピーを外界と反応できない形で持ち運ぶ機能が必要となる．LMNtal では，前者にはプロセス文脈を，後者には膜のもつ保護機能を活用する（**図 10**）．

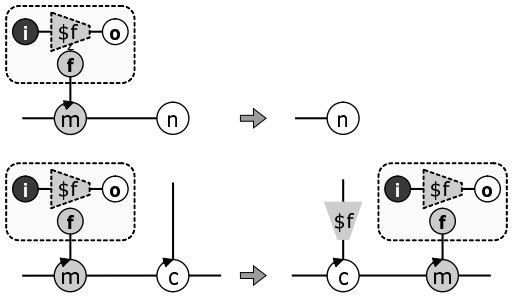


図 10　マップ関数

```
map(F,C,L), n(C),
{f(F),i(I),o(O),{$f[I,O]}} :-
        n(L)
map(F0,C0,L), c(A0,R,C0),
{f(F0),i(I0),o(O0),{$f[I0,O0]}} :-
        $f[A0,B0],
        c(B0,C,L), map(F,R,C),
        {f(F),i(I),o(O),{$f[I,O]}}
```

マップするプロセスは

```
{f(F),i(I),o(O),{s(I,O)}}
```

または

```
{f(F),i(I),o(O),{s(I,M),s(M,O)}}
```

のように指定する．この形式は λ 抽象のエンコーティングと見ることもできる．一方の λ 適用は，上記のように，プロセス文脈の新たなコピーへのリンキングと膜の除去とによって表現している．

6.5　非同期 π 計算

非同期 π 計算における通信を LMNtal で表現することを考えよう．話を簡単にするために，複数チャネルからの選択的受信は考えないことにする．

非同期 π 計算のチャネルは，一般に不特定多数の送信者と受信者をもつ．これは，LMNtal では同一セルに入射するリンクによって表現できる．このセルがチャネルのアイデンティティを表現していることになる．またチャネル y からメッセージ z を受け取る受信プロセス $y(z).P$ は，受信が成立するまで P の実行を保護しなければならない．反応の保護は，LMNtal では膜によって実現できる．これらの考察から，通信

$$\overline{y}(x) \,|\, y(z).P \longrightarrow P[x/z]$$

は下記のようにエンコードできる．

```
send(X,Ybar), {c(Ybar),c(Y),$y[|*Y]},
receive(Z,Y), {$p[Z|*P]} :-
        {$y[|*Y]}, $p[X|*P]
```

ここで `send(X,Ybar)` は `Ybar` に `X` を送信することを，`receive(Z,Y)` は `Y` から `Z` を受信することを表す．

上記のエンコーディングを BRS（3.3.2 節）による非同期通信のエンコーディング [20] と比較すると，`receive` の後の計算のガード方法などに興味深い共通点がある．

LMNtal と BRS は，非常に異なる背景から独立に生まれたモデルであるが，リンクと階層化の両方を備えている点で共通している．しかし BRS のリンクが 3 点以上を結ぶことのできるハイパーリンクであるのに対し，LMNtal ではハイパーリンクはセルの多重集合機能を使えば実現できることから，通常のリンクを基本としている．BRS では，ノードに入れ子関係を導入することによって階層化を実現しているが，この考え方のもとでは，LMNtal の膜は無引数の無名ノードと考えることもできる．LMNtal のプロセス文脈は BRS の inner interface にほぼ対応するが，プロセス文脈は，文脈の複写や自由リンクに関する仕様の指定などの強力な機能を持っている．BRS のリダクションは，複数のノードがリンクでつながっていれば，それらがノードの階層構造において任意に離れた 2 点にあっても同期的に書き換えることができる．これに対して LMNtal では，複数のアトムがリンクでつながっていても，そのリンクが通過する膜の枚数がわかっていないと同時に書き換えることはできず，LMNtal の書換えの方が局所性が高い．この点で，LMNtal の局所性の思想は Ambient 計算における局所化の思想と共通性が高い．

7　まとめ

Logical links，multisets，nested nodes，transformation を四大要素としてもつ単純な言語モデル LMNtal を紹介した．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．多重集合やグラフの書換え

を何らかの意味でサポートする計算モデルや言語は多いが，LMNtal は，階層的多重集合の下でのリンク管理機能の設計やルールセットの管理方式に特徴がある．

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．Flat LMNtal は CHR の部分集合とみなすこともできるが，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．CHR を LMNtal に埋め込むのは興味深い研究課題である．

LMNtal の実装として，Java で記述したプロトタイプ処理系[†3]が稼働を始めている [33]．LMNtal は多重集合書換え言語の一種であるが，細粒度の並行性を持っている点に加えて，多重集合が膜によって構造化される点，膜が局所的な書換えルールを持てる点，およびルールセットが移送可能である点から，正しくかつ効率的な実装方式は自明でない．特に，複数の計算主体による同時実行の制御方式を確立することは，広域計算への応用を目指すにあたって非常に重要である．この観点から我々は，プロトタイプ処理系の開発を通じて，言語設計自体を検証することと，LMNtal 実装の基本方式の確立とを目指してきた．この処理系は本解説で紹介した言語機能の大多数をカバーするとともに，

- セルの反応停止をセル外から検知する機能
- 組込みの数値型
- 数値やシンボルの検査と比較のための型制約概念およびガード機能
- 他言語 (Java) とのインタフェース機能

なども備える．詳細は別論文で紹介したい．

言語仕様に関しては，上記のほかに

- モジュール機能および名前の局所化機能
- 可算無限個の多重度をもつ分子
- 言語要素（アトムや膜など）と物理資源（空間および計算時間）との対応づけ機能

などの検討を進めている．

将来課題は多岐にわたる．言語設計上の最大の将来課題は，役に立つ型体系の構築である．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），セルの自由変数の管理，セル内のルールの不変性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であろう．実装に関しても，プロセスとリンクの表現の最適化，ルールの最適化コンパイル，並列分散実装など，多くの興味深い課題がある．LMNtal は既存のモデルの統合を目指しているので，関連言語や計算モデルの LMNtal への埋込みや LMNtal による実装も重要な課題である．

プログラムの検証においては，自動検証法の確立も重要ではあるが，LMNtal の特徴を活かした「プログラム理解の助けになる human-oritented な証明」の方法論を開拓したい．グラフ表現との対応や，本解説に示したプログラムの多くに見られる対称性をうまく活用して直感的な証明ができれば，プログラム検証に新たな分野が開ける可能性がある．

LMNtal は言語設計と処理系の両面において「使える」言語を目指している．そのことを実証するには広範な分野におよぶプログラム記述例の蓄積が大切である．並行分散計算以外の応用分野としては，グラフアルゴリズム，マルチエージェント，Web サービス，グラフィクス，自己組織化に基づくプログラミングなどがあり，記述実験と動作確認を進めつつ，今後も言語仕様の細部の洗練を図ってゆきたい．

†3 `http://www.ueda.info.waseda.ac.jp/lmntal/`

謝辞

LMNtal の設計は，早稲田大学上田研究室言語班のメンバ諸氏との活発な議論と彼らの実装努力に支えられてきた．ここに感謝したい．また，査読者からは多数の有益なコメントをいただいた．本研究の一部は，文部科学省科学研究費特定 (C)(2)13324050 および特定 (B)(2)14085205 の補助を得て行った．

参考文献

[1] Andreoli, J.-M. and Pareschi, R. : Linear Ojects: Logical Processes with Built-in Inheritance, *New Generation Comput.*, Vol. 9, Nos. 3-4 (1991), pp. 445–473.

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

[3] Banâtre, J.-P. and Le Métayer, D. : Program-

ming by Multiset Transformation, *Commun. ACM*, Vol. 35, No. 1 (1993), pp. 98–111.

[4] Berry, G. and Boudol, G. : The Chemical Abstract Machine, in *Proc. POPL'90*, ACM, pp. 81–94.

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

[6] Cervesato, I., Durgin, N. A., Lincoln, P. D., Mitchell, J. C. and Scedrov, A. : A Meta-Notation for Protocol Analysis, in *Proc. IEEE Computer Security Foundations Workshop (CSFW'99)*, 1999, pp. 55–69.

[7] De Nicola, R., Ferrari, G. and Pugliese, R. : Klaim: a Kernel Language for Agents Interaction and Mobility, *IEEE Trans. Softw. Eng.*, Vol. 24, No. 5 (1998), pp. 315–330.

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

[9] 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.

[10] Forgy, C. L. : A Fast Algorithm for the Many Patterns/Many Objects Match Problem, *Artif. Intell.*, Vol. 19, No. 1 (1982), pp. 17–37.

[11] Fradet, P. and Le Métayer, D. : Structured Gamma, *Sci. Comput. Program.*, Vol. 31, No. 2–3 (1998), pp. 263–289.

[12] Fradet, P. and Le Métayer, D. : Shape Types, in *Proc. POPL'97*, ACM, 1997, pp. 27–39.

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

[14] Gardner, P. and Wischik, L. : Explicit Fusions, in *Proc. 25th Int. Symp. on Mathematical Foundations of Computer Science (MFCS 2000)*, LNCS 1892, Springer-Verlag, pp. 373–382.

[15] Gelernter, D. : Generative Communication in Linda, *ACM Trans. Prog. Lang. Syst.*, Vol. 7, No. 1 (1985), pp. 80–112.

[16] Habel, A. : *Hyperedge Replacement: Grammars and Languages*, LNCS 643, Springer-Verlag, 1992.

[17] Hodas, J. S., Watkins, K. M., Tamura, N. and Kang, K.-S. : Efficient Implementation of a Linear Logic Programming Language, in *Proc. 1998 Joint Int. Conf. and Symp. on Logic Programming*, MIT Press, 1998, pp. 145–159.

[18] Holzbaur, C., Garcia de la Banda, M. J., Jeffery, D. and Stuckey, P. J. : Optimizing Compilation of Constraint Handling Rules, in *Proc. 17th Conf. on Logic Programming (ICLP 2001)*, LNCS 2237, Springer-Verlag, 2001, pp. 74–89.

[19] 石田亨：プロダクションシステムの発展，朝倉書店，1996.

[20] Jensen, O. H. and Milner, R. : Bigraphs and Transitions, in *Proc. POPL 2003*, ACM, 2003, pp. 38–49.

[21] Kanada, Y. and Hirokawa, M. : Stochastic Problem Solving by Local Computation based on Self-organization Paradigm, in *Proc. 27th Hawaii Int. Conf. on System Sciences (HICSS-27)*, 1994, pp. 82–91.

[22] Lafont, Y. : Interaction Nets, in *Proc. POPL'90*, ACM, 1990, pp. 95–108.

[23] Miller, D. : Overview of Linear Logic Programming, to appear in *Linear Logic in Computer Science*, Ehrhard, T., Girard, J.-Y., Ruet, P. and Scott, P. (eds.), Cambridge University Press.

[24] Milner, R. : Calculi for Interaction, *Acta Informatica*, Vol. 33, No. 8 (1996), pp. 707–737.

[25] Milner, R. : Bigraphical Reactive Systems, in *Proc. CONCUR 2001*, LNCS 2154, Springer-Verlag, 2001, pp. 16–35.

[26] 村田忠夫：ペトリネットの解析と応用，近代科学社，1992.

[27] Murphy, A. L., Picco, G. P. and Roman, G.-C. : LIME: A Middleware for Physical and Logical Mobility, in *Proc. 21st Int. Conf. on Distributed Computing Systems (ICDCS 2001)*, IEEE Computer Society, 2001, pp. 524–533.

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

[29] Rozenberg, G. (ed.) : *Handbook of Graph Grammars and Computing by Graph Transformation, Vol.1: Foundations*, World Scientific, 1997.

[30] 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.

[31] Ueda, K. : Concurrent Logic/Constraint Programming: The Next 10 Years, in *The Logic Programming Paradigm: A 25-Year Perspective*, Apt, K. R., Marek, V. W., Truszczynski M., and Warren D. S. (eds.), Springer-Verlag, 1999, pp. 53–71.

[32] Ueda, K. : Resource-Passing Concurrent Programming, in *Proc. TACS 2001*, LNCS 2215, Springer-Verlag, 2001, pp. 95–126.

[33] 矢島伸吾, 永田貴彦, 加藤紀夫, 上田和紀：LMNtal プロトタイプ処理系の設計と実装，日本ソフトウェア科学会第 20 回記念大会論文集, 2003, pp. 21–25.