GHC から *LMNtal* へ

上田 和紀，加藤 紀夫

早稲田大学理工学部情報学科

2002年9月

2002年夏のプログラミング・シンポジウム

概 要

- 多重集合（multisets）の書換えに基づく多数の計算モデルの統合を目指して設計した言語 *LMNtal* を紹介する．*LMNtal* は，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で，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．

LMNtal (pronounce: "*elemental*")

L inked

M ultisets of

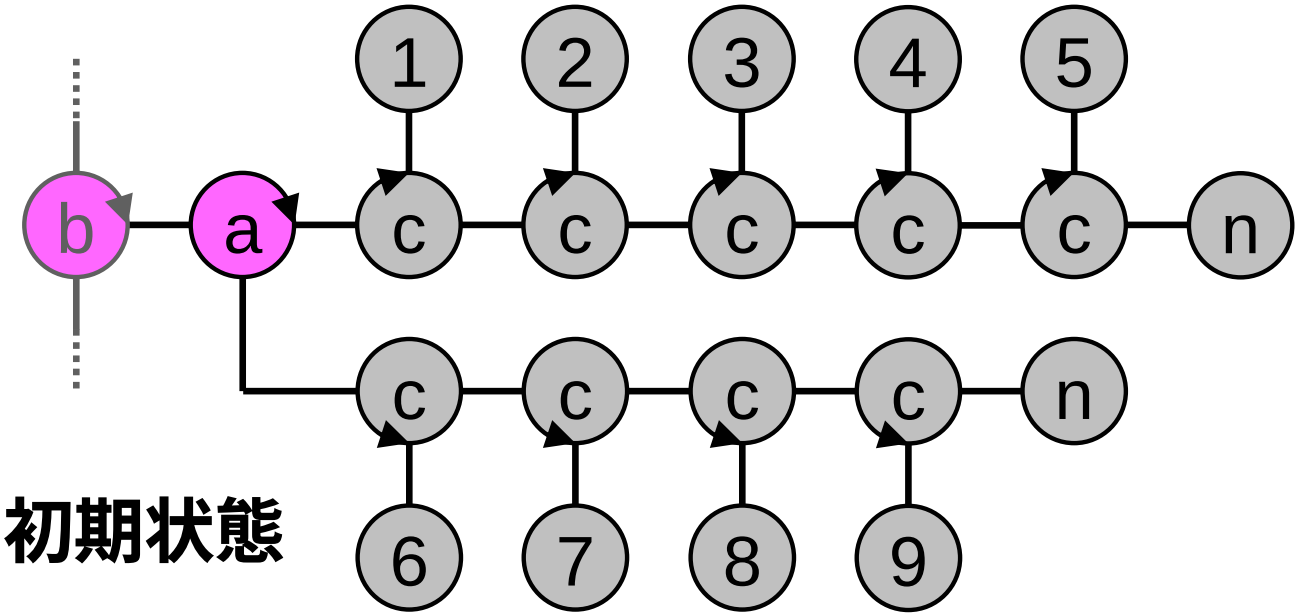
N odes

t ransform-

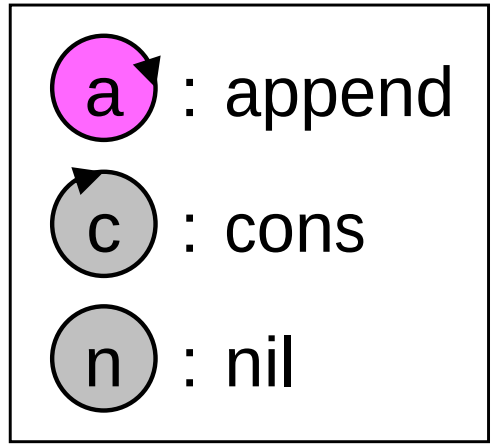
a tion

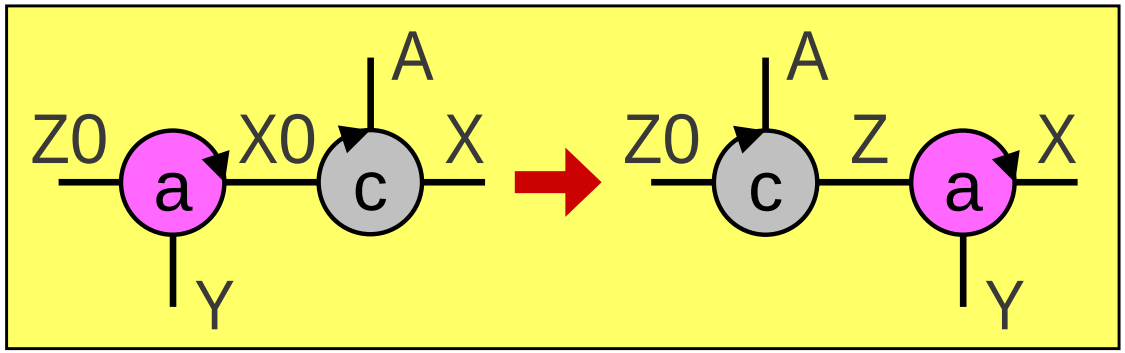
l anguage

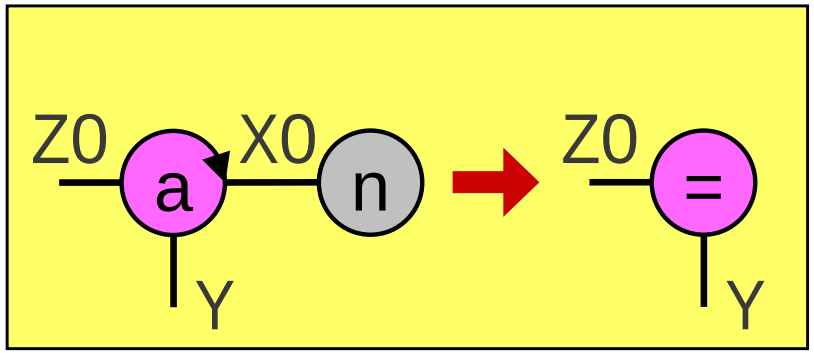
Example 1: append



初期状態

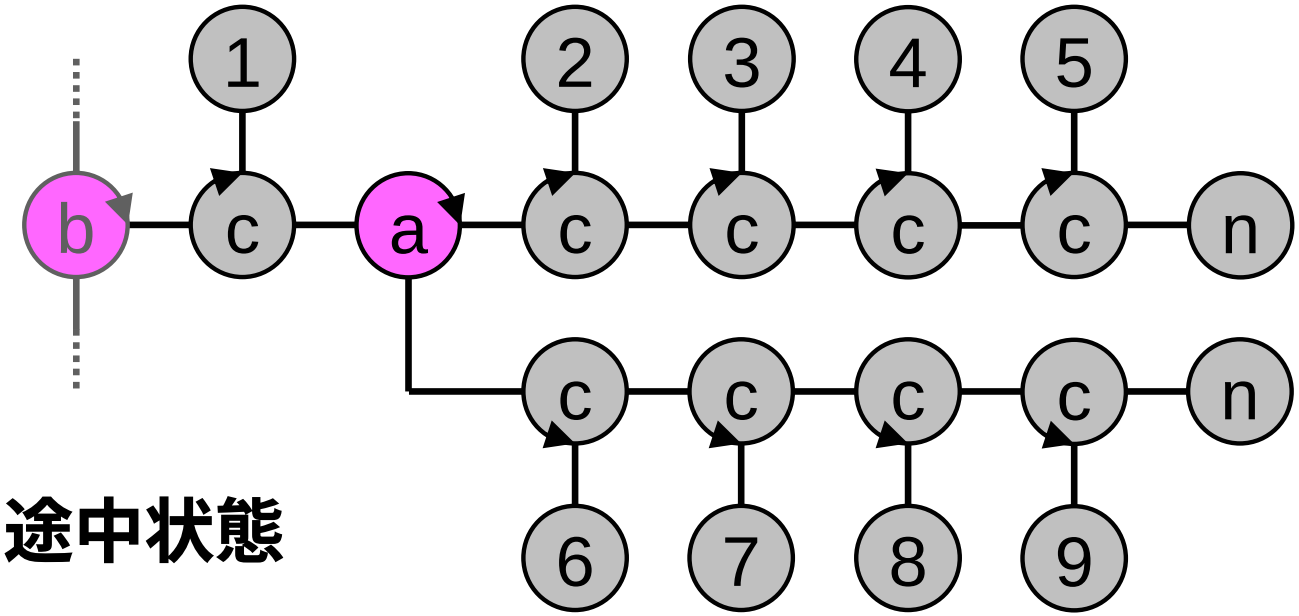




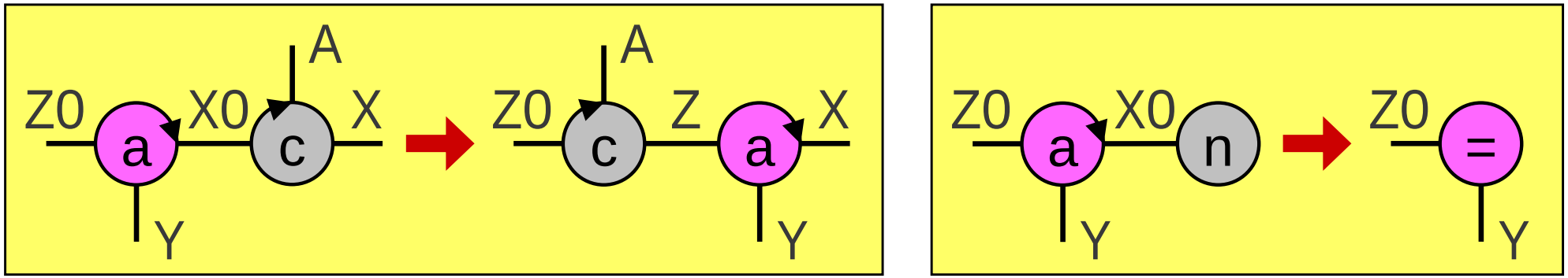


プログラム（書換規則）

Example 1: append

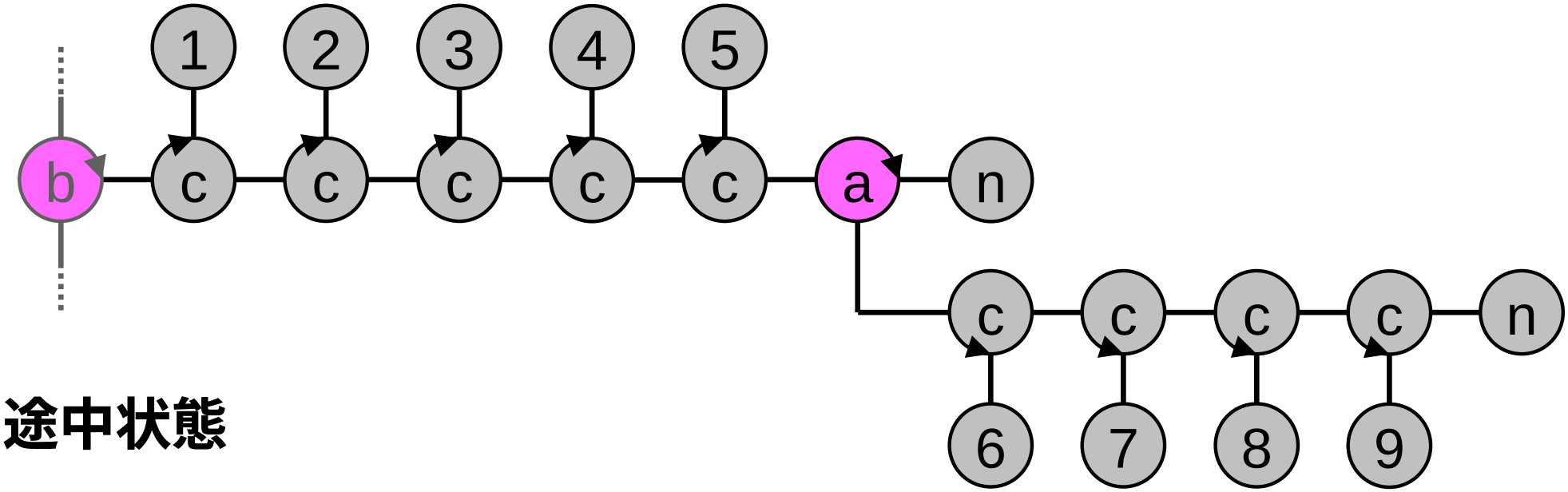


途中状態

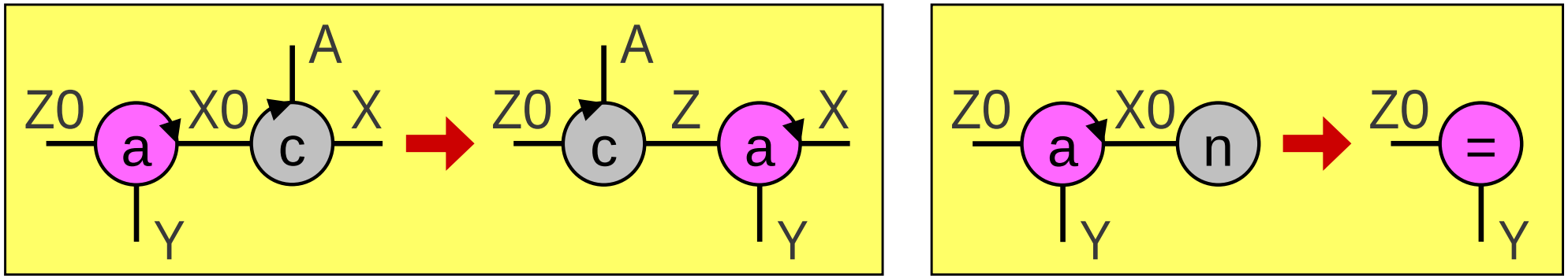


プログラム（書換規則）

Example 1: append

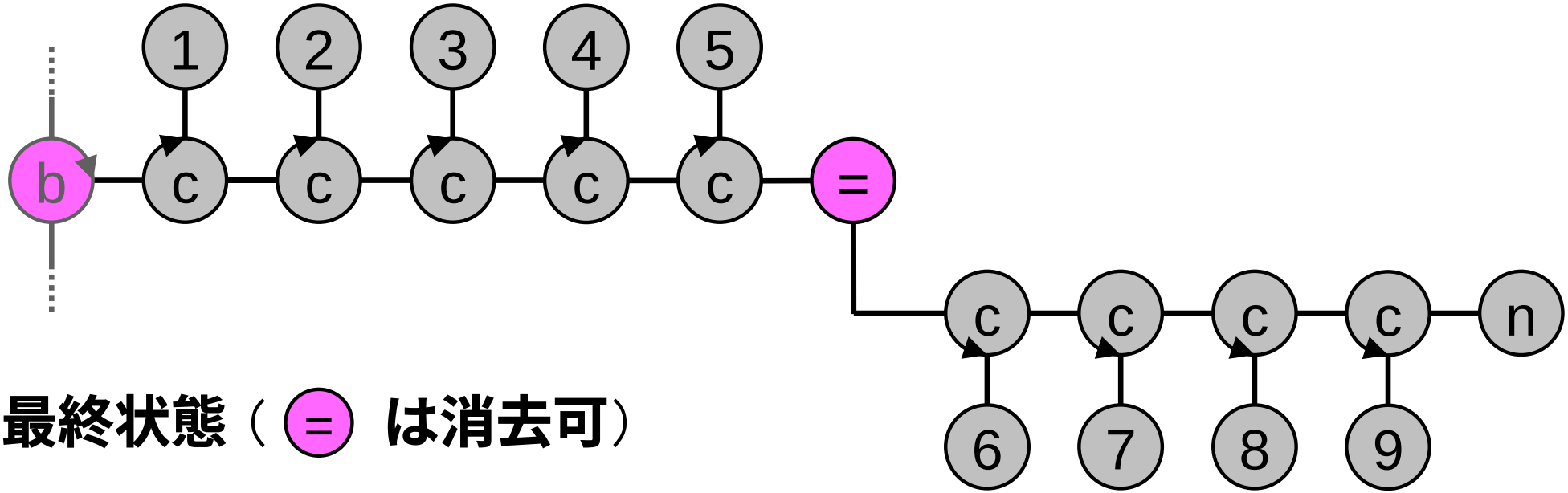


途中状態

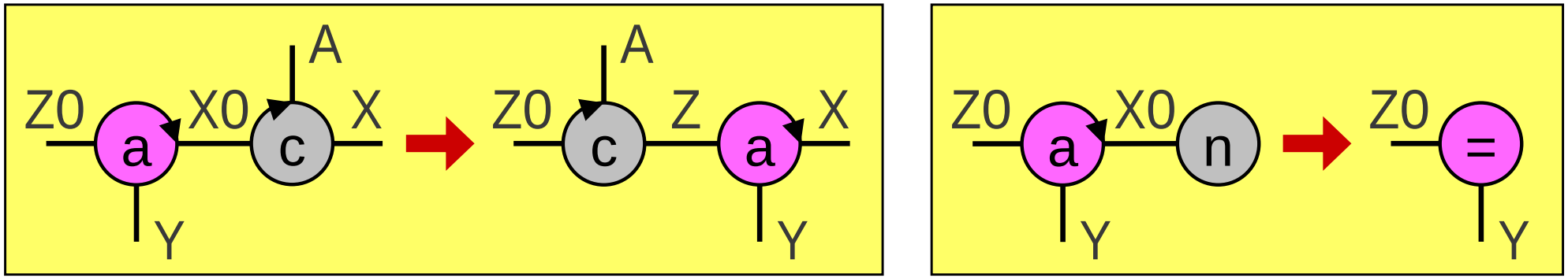


プログラム（書換規則）

Example 1: append

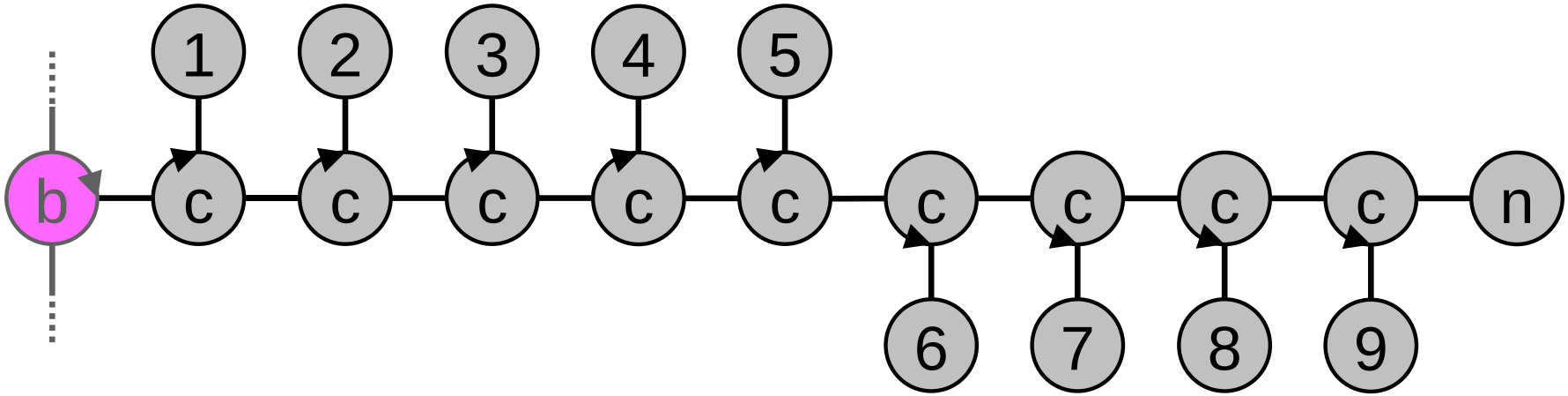


最終状態（ = は消去可）

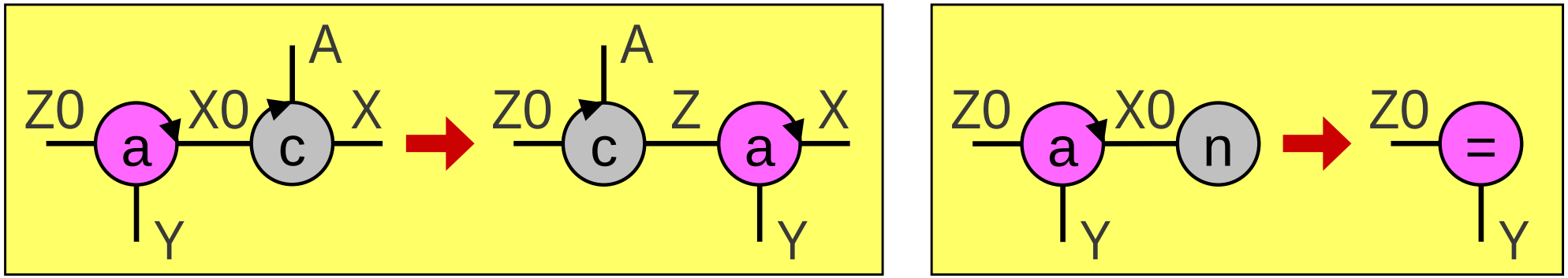


プログラム（書換規則）

Example 1: append



最終状態



プログラム（書換規則）

append in *LMNtal* , text version

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

★ cf. GHC version

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

- append **と** c(ons) **は基本的に同格．実装上の区別がありうるのみ．**

背景 (1)

- **並行論理プログラミング**
  - ★ **論理変数を用いた** channel mobility **の実現**
  - ★ **多様な型体系（⊇ 線形性），豊富な実装経験**
- **並行制約プログラミング**
  - ★ **データ領域の一般化（**cc(FD), cc(multiset), . . .**）**
- CHR (Constraint Handling Rules)
  - ★ **頭部にゴール（制約）の多重集合が書ける**
  - ★ **多重集合書換えに基づく言語の中では強力**
  - ★ **多くの応用プログラム（特に制約処理系）**
  - ★ **反応制御の機構（終了検出，階層化等）が欠如**

背景 (2) — 多重集合をもつ計算モデル

- Petri Nets
- Production Systems and RETE match
- CCS, CSP, concurrent logic programming
- Linda
- Linear Logic languages
- Gamma model
- Constraint Handling Rules
- Graph transformation formalisms
- Chemical Abstract Machine (CHAM), reflexive CHAM, and the Join Calculus
- Mobile ambients
- P system & membrane computing
- Amorphous computing

赤字：対称な会合を持つ計算モデル

LMNtal の設計目標 (1)

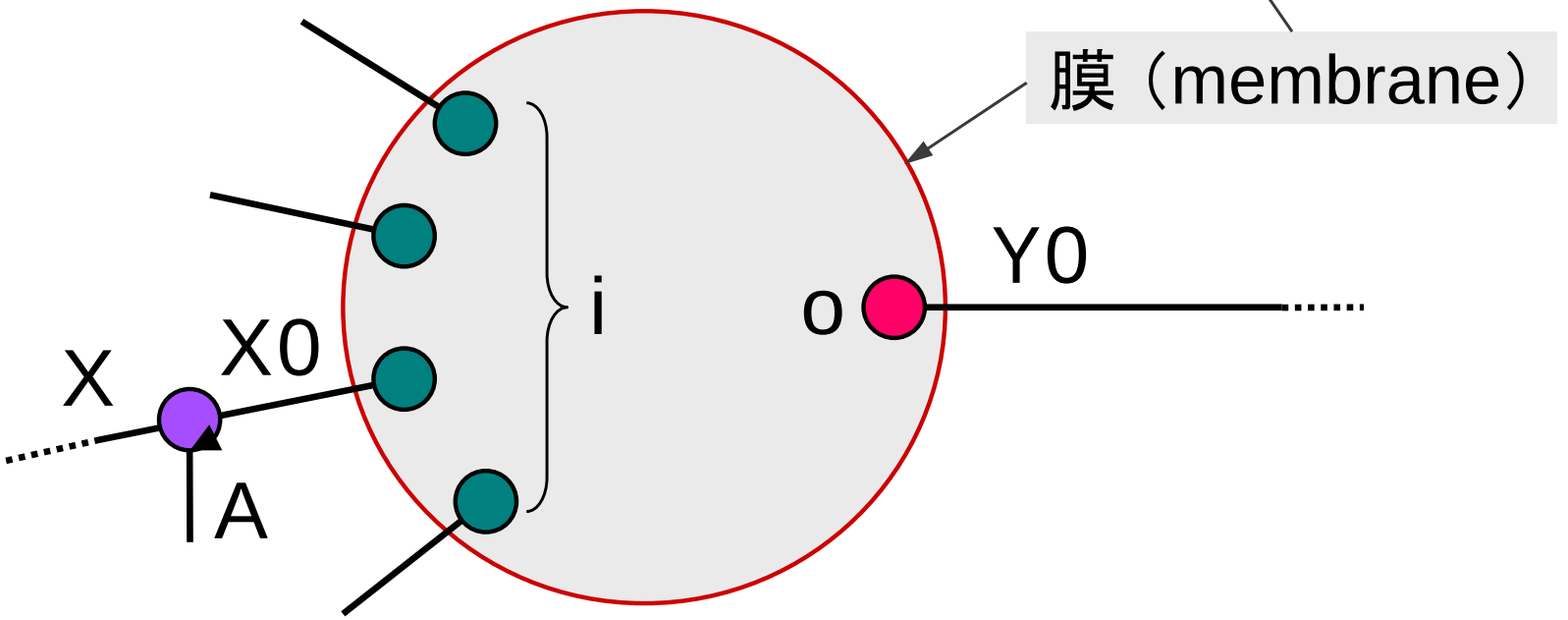
- Simple
  - ★ 理論計算モデルとしての簡潔さとプログラム言語としての実際性を兼備
- Unifying and scalable
  - ★ 多様な計算モデルや概念の統合を図る（後述）
  - ★ 多様な計算環境をカバーする
- Easy to program
  - ★ 計算は図形変換とみなせる ➔ 書き易く直観的
  - ★ ビジュアル版とテキスト版が提供可能
- Fast
  - ★ 効率的な実装の見通しがある

LMNtal の設計目標 (2)

- 概念統合の例：
  - ★ 自律プロセスとデータを統一的に扱いたい
    - プロセス vs. メッセージ
    - 動的プロセス構造 vs. 動的データ構造
      - 多くの宣言型言語は木構造以外を無視！
      - 手続き型言語のポインタ操作は error-prone
      - ➔ 動的データ構造（一般にはグラフ）の操作を自然に記述できる枠組みとしても使える
  - ★ 同期通信と非同期通信を統一的に扱いたい

Example 2: N-to-1 stream communication

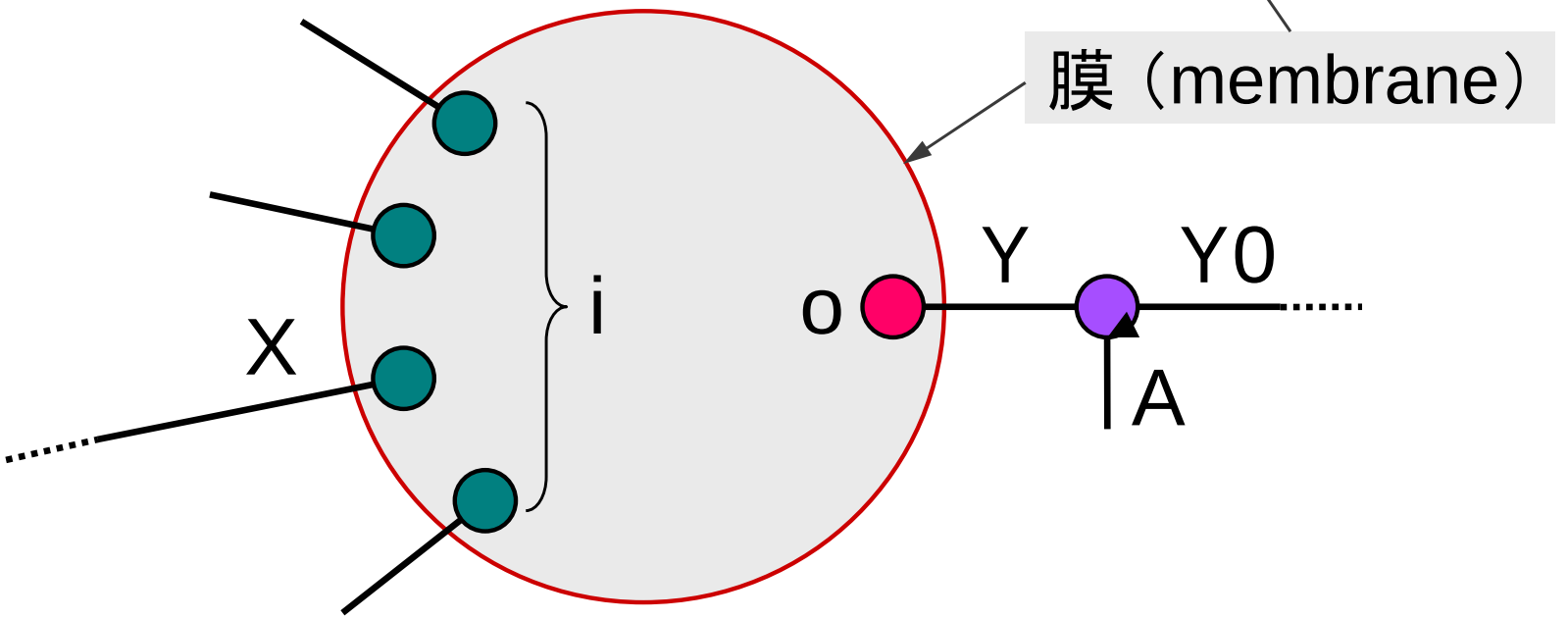
```
{i(X0), o(Y0), $p(*)}, c(A,X,X0) :-
        c(A,Y,Y0), {i(X), o(Y), $p(*)}
```



★ The number of free variables in { } remain unchanged

Example 2: N-to-1 stream communication

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



★ The number of free variables in { } remain unchanged

LMNtal の大枠 (1)

１．リンク（腕）をもつ原子（ノード）

- ★ リンクは１対１接続 (cf. hyperlinks (n-to-1))
- ★ リンクの実体は論理変数
  - メッセージ送信の前後でアイデンティティが変わる (cf. π 計算)
  - リンクは常にプライベート
  - しかし具体化（instantiate）という概念はない (cf. 論理型言語，CHR)
- ★ 基本的に無方向リンク
- ★ 各原子のもつリンクは順序づけられている

LMNtal の大枠 (1) — リンクの有用性

- (a) チャネル（プロセス間通信路）の表現
  - チャネルはプライベート
    - 他のプロセスはアクセス不可能
- (b) データ構造（グラフ）の表現
- (c) 多重集合書換えにおけるパートナーの探索
  - リンクがあれば O(1) 時間
  - リンクがないと（下手をすると）O(*n*) 時間

注：リンクにはとりあえず向きの概念はない

- 向きは型体系を使って表現・発見

LMNtal の大枠 (2)

２．多重集合とその階層化（膜の概念）

- ★ 一級市民（first-class citizen）として多重集合をもつ言語は多くない
  - ◆ cf. Janus　/　Gamma, Linda, CHR
- ★ 階層化の目的（cf. CHAM, Oz）
  - ■ （論理的な）計算管理
    - ◆ cf. dklic, ambients, join calculus, OS *vs*. user programs
  - ■ （物理的な）資源管理
    - ◆ cf. regions
  - ■ 計算の局所化

LMNtal の大枠 (3)

３．書換え規則に基づく構文

- ★ リンクに関する機能設計がポイント
  - ◆ cf. グラフ文法，グラフ書換え，論理プログラミング
- ★ Channel mobility と process mobility の両方を備える
  - ■ 書換え規則（の多重集合）が一級市民

構文 — リンクと名前

- 最初に，次の2種類の構文カテゴリを仮定．
  - ★ *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．
  - ★ *p* は名前を表わす．数も名前の一種．
    具体構文ではリンクと区別できる識別子で表記．
- 名前 = は，*LMNtal* 唯一の予約名．

LMNtal process の構文

Flat LMNtal
にはない機能

- P ::= **0** (null)
  - | $p(\vec{X})$ (simple process)
  - | P, P (composition)
  - | $\{ P \}$ (compound process)(*)
  - | $(T :- T)$ (reaction rule)
- **リンク条件：** *P* **中の各リンク（**reaction rule **内は除く）は**たかだか 2 回**しか出現しない**
  - ★ *P* **の自由リンク** = 1 **回しか出現しないリンク**
  - ★ *P* **が閉じている** = **自由リンクを持たない**

LMNtal process template の構文

- T ::= **0** (null)
  - | $p(\vec{X})$ (simple process)
  - | $\$p(FVspec)$ (process variable)(*)
  - | $@p$ (rule variable)(*)
  - | T, T (composition)
  - | $\{ T \}$ (compound process)(*)

Flat LMNtal
にはない機能

- **プロセス変数は書換え規則以外のプロセスとマッチ．**
  - ★ *FVspec* **で自由リンクの出現条件を指定**
- **ルール変数は書換え規則の多重集合とマッチ**

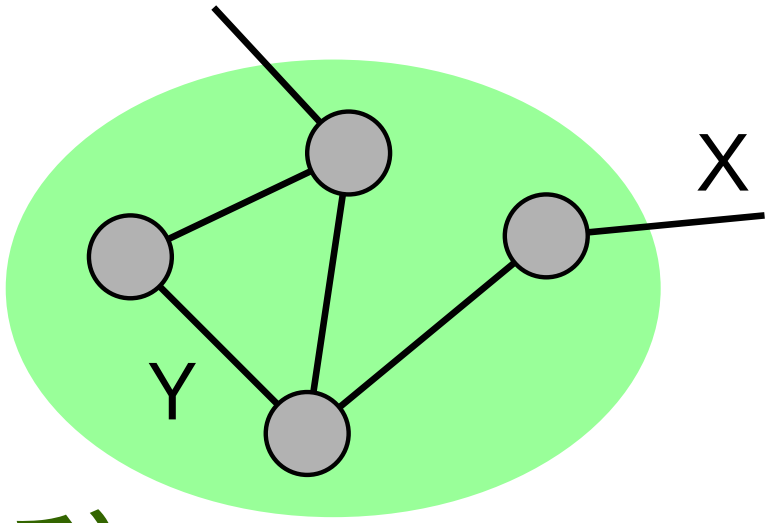
プロセス変数の自由リンク制約

- [+] *FVspec* ::= [+ X]* [+ * [– X]*]
  - ★ ＋：自由リンクとして出現しなければならない
  - ★ –：自由リンクとして出現してはならない
  - ★ ＋*：言及されていないリンクが自由に出現してもよい
- **例：** $p(X+Y)

  $p(X+*–Y) ----------→

  $p

  （閉じたプロセスとマッチ）



プロセス変数・ルール変数の出現規則

1．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

➔系： Flat LMNtal にはプロセス変数とルール変数は存在しない

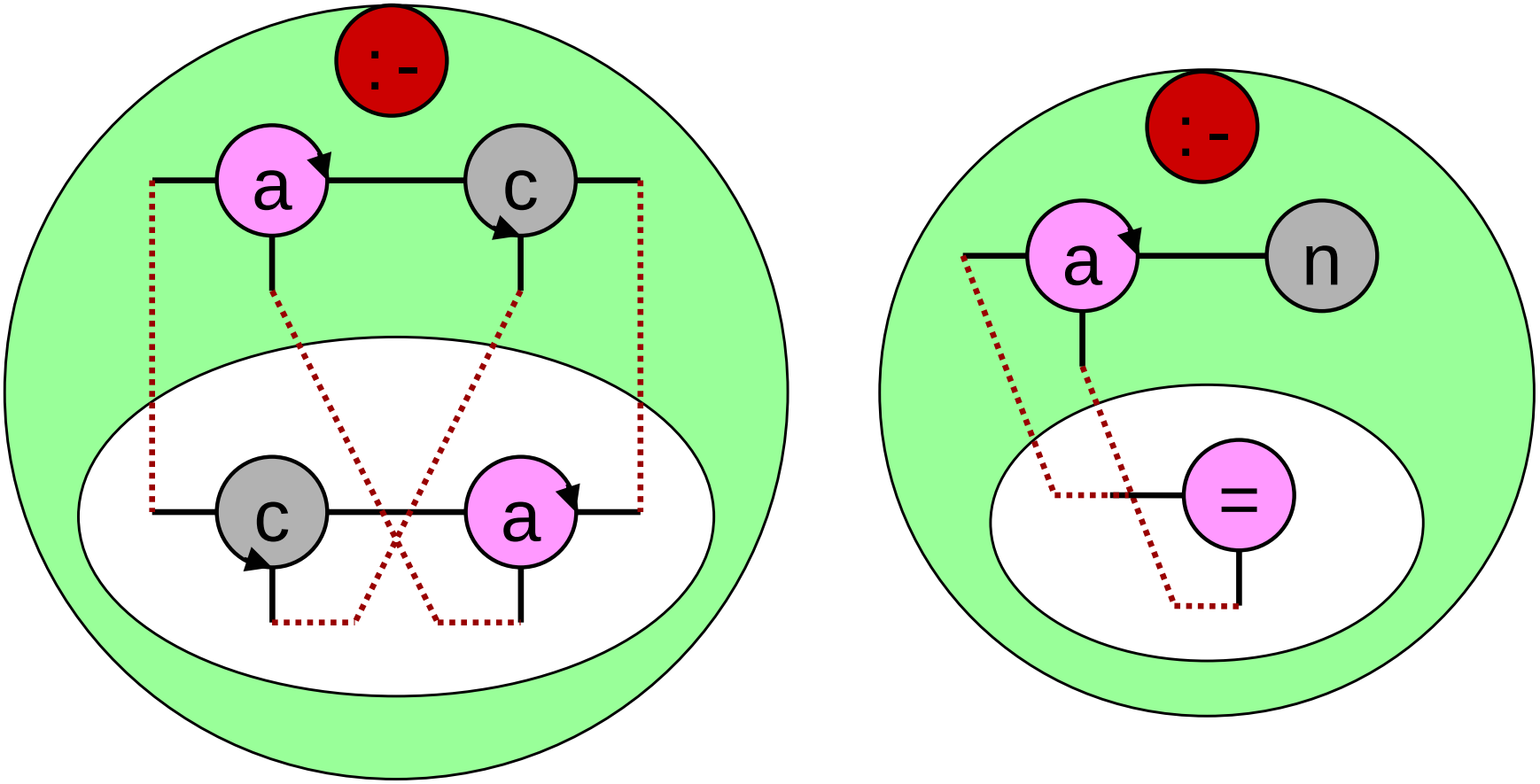
2．ルール左辺の compound process は，そのトップレベルに，プロセス変数およびルール変数をたかだか 1 個ずつ持つことができる

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（cf. 閉じたプロセスは複写廃棄可）

LMNtal process template の構文

- Reaction rule (*L* :- *R*) 内のリンクは次のいずれか.
  - ★ ちょうど 2 回現れる
  - ★ *L* と *R* 中に 2 回ずつ現れる
    （ただしプロセス変数中の負の出現は数えない）
- リンクの分類
  - ★ *L* 中のみ：消費されるリンク
  - ★ *R* 中のみ：生成されるリンク
  - ★ 1 回ずつ：継承されるリンク
  - ★ 2 回ずつ：リサイクルされるリンク

わき道：append のエンコーディング例



：越境リンク

意味：構造合同 (structural congruence)

(1) $\mathbf{0}, P \equiv P$

(2) $P, Q \equiv Q, P$

(3) $P, (Q, R) \equiv (P, Q), R$

(4) $P \equiv P'$ if P' and P are α-convertible

(5) $P \equiv P' \Rightarrow P, Q \equiv P', Q$

★ Link conditions should hold on both sides of $\equiv$

(6) $P \equiv P' \Rightarrow \{P\} \equiv \{P'\}$

(7) $\{P\} \equiv \{P\}/$ if P is irreducible

意味：構造合同 (structural congruence)

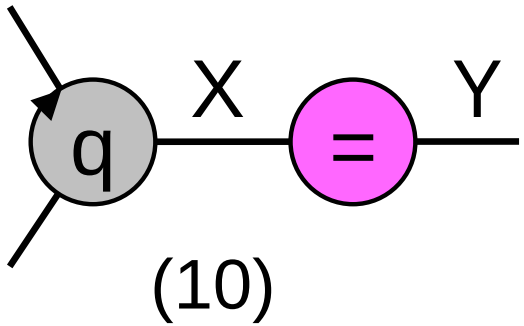
The rest of the rules are about “=” :

(8) $X = X \equiv \mathbf{0}$

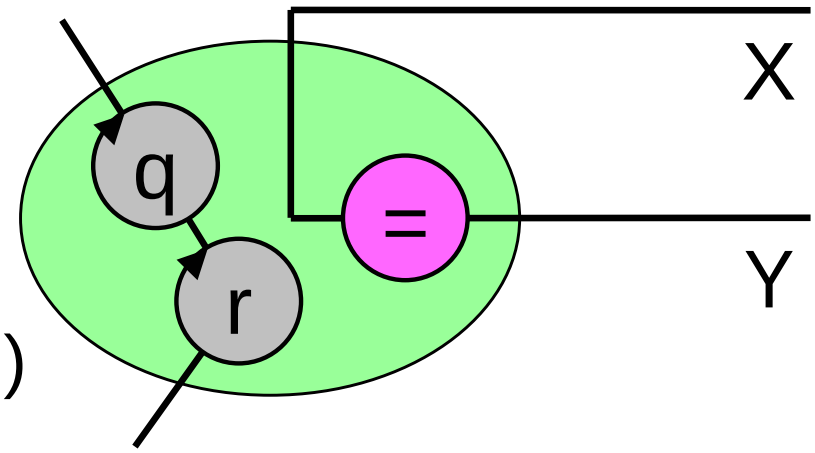
(9) $X = Y \equiv Y = X$

(10) $X = Y, \$p(X + *) \equiv \$p(X + *)[Y/X]$

(11) $\{X = Y, P\} \equiv \{P\}, X = Y$



(10)



(11)

意味：構造合同 (structural congruence)

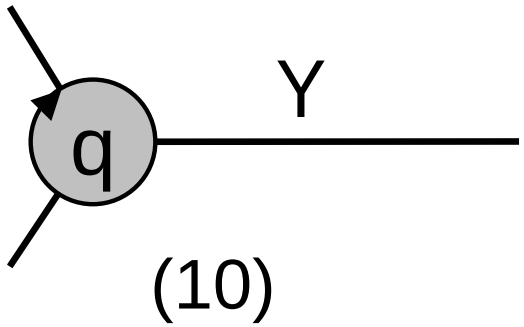
The rest of the rules are about “=” :

(8) $X = X \equiv \mathbf{0}$

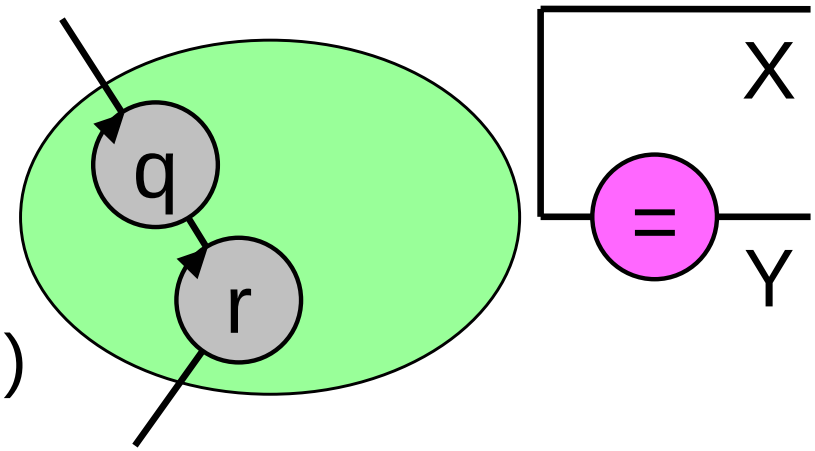
(9) $X = Y \equiv Y = X$

(10) $X = Y, \$p(X + *) \equiv \$p(X + *)[Y/X]$

(11) $\{X = Y, P\} \equiv \{P\}, X = Y$



(10)



(11)

意味：簡約規則 (reduction rules)

$$\frac{P \to P'}{P, Q \to P', Q} \quad \text{(i)}$$

$$\frac{P \to P'}{\{P\} \to \{P'\}} \quad \text{(ii)}$$

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

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

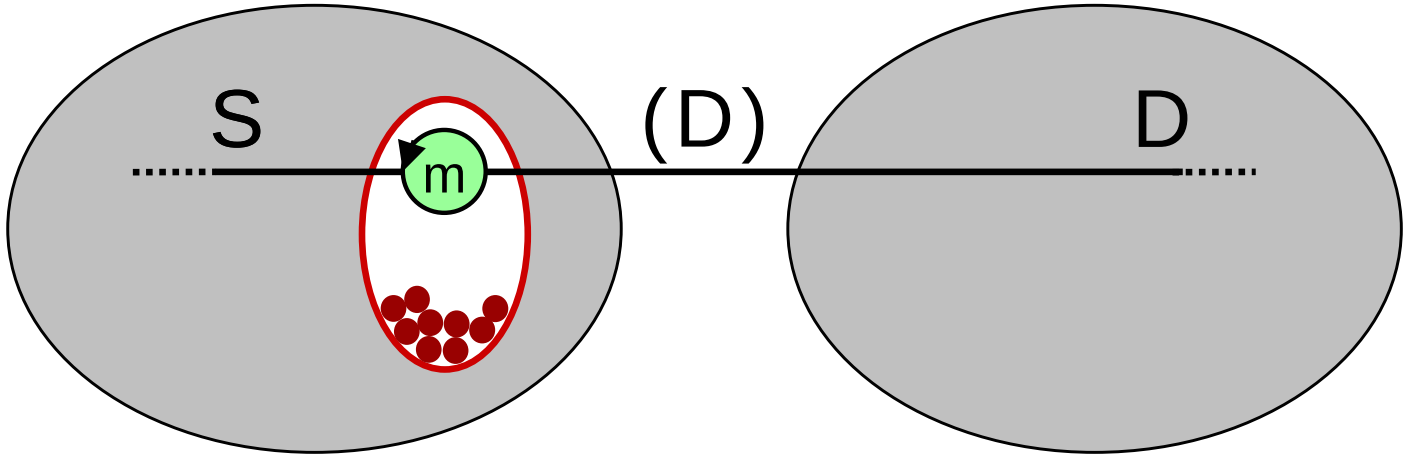
θはプロセス変数とルール変数の具体化用.
リンクの対応はα変換で自動的にとられる.

（Flat LMNtal では
$P, (P :- Q) \to Q, (P :- Q)$ となる）

Example 3: process migration

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

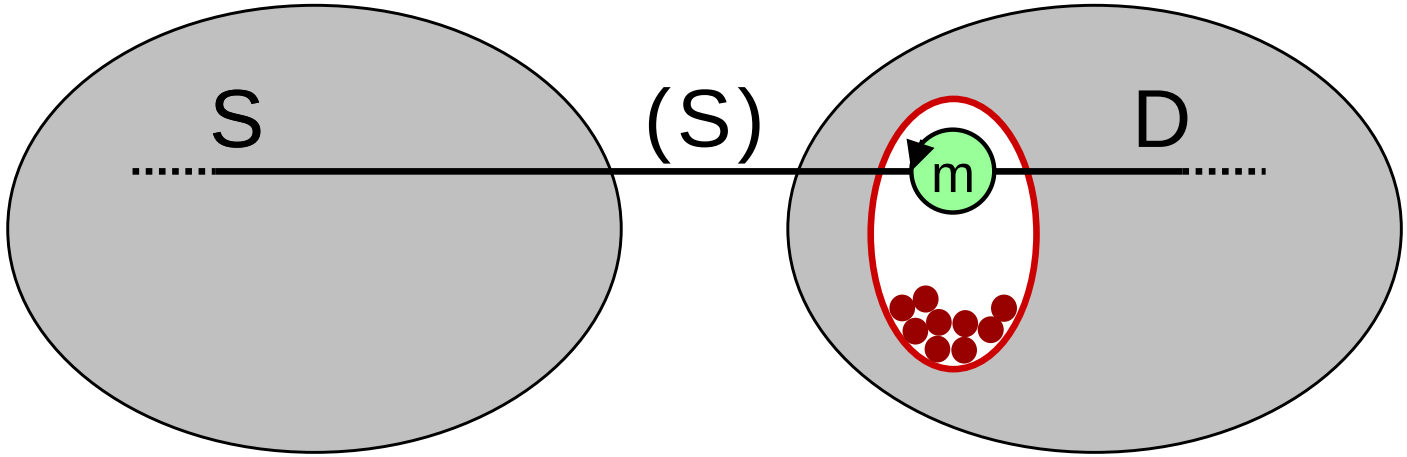
- ★ Placed outside the membranes
- ★ Innermost { } is to specify what should migrate



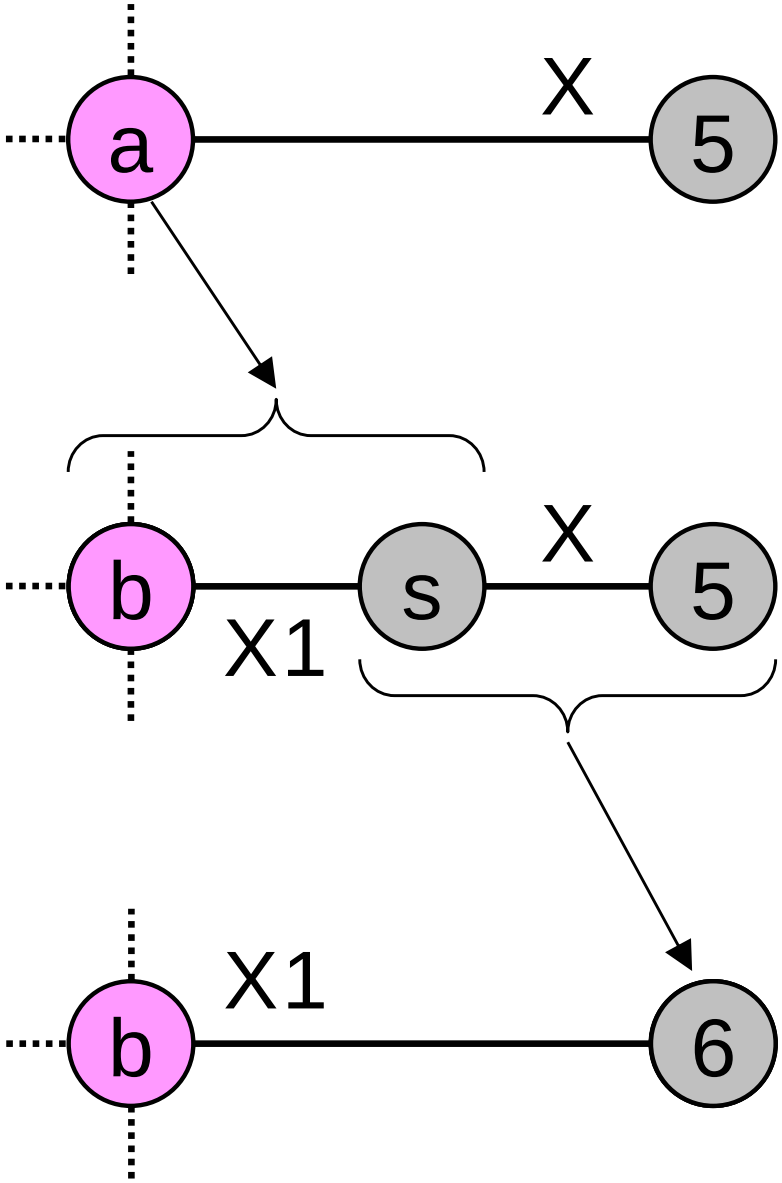
Example 3: process migration

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

- ★ Placed outside the membranes
- ★ Innermost { } is to specify what should migrate



Example 4: integer operations



a(X), 5(X)
alternative notations:
a(5) or 5(a)

b(X1), s(X,X1), 5(X)
or b(s(X)), 5(X)
or b(s(5)) or s(b,5)
or …

b(X1), 6(X1)
or b(6)
or 6(b)

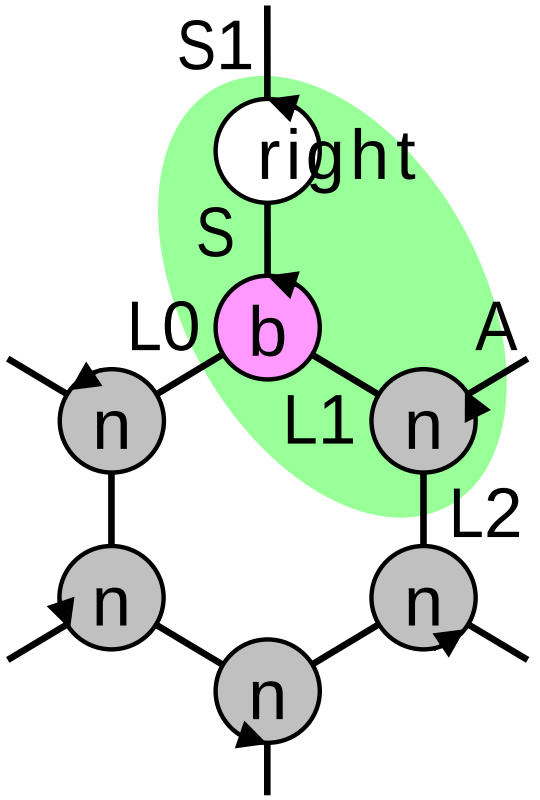
Example 5: circular data structures

- Bidirectional circular buffer:

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

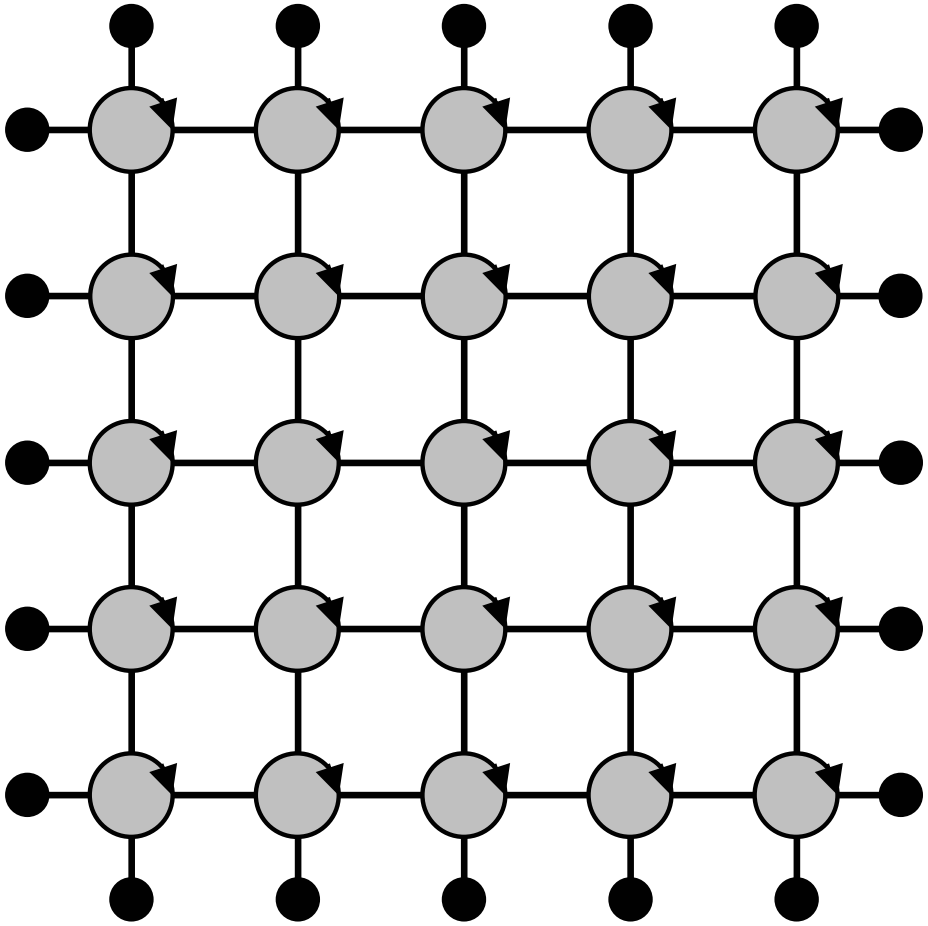
★ *S* acts as an interface link

```
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)
```



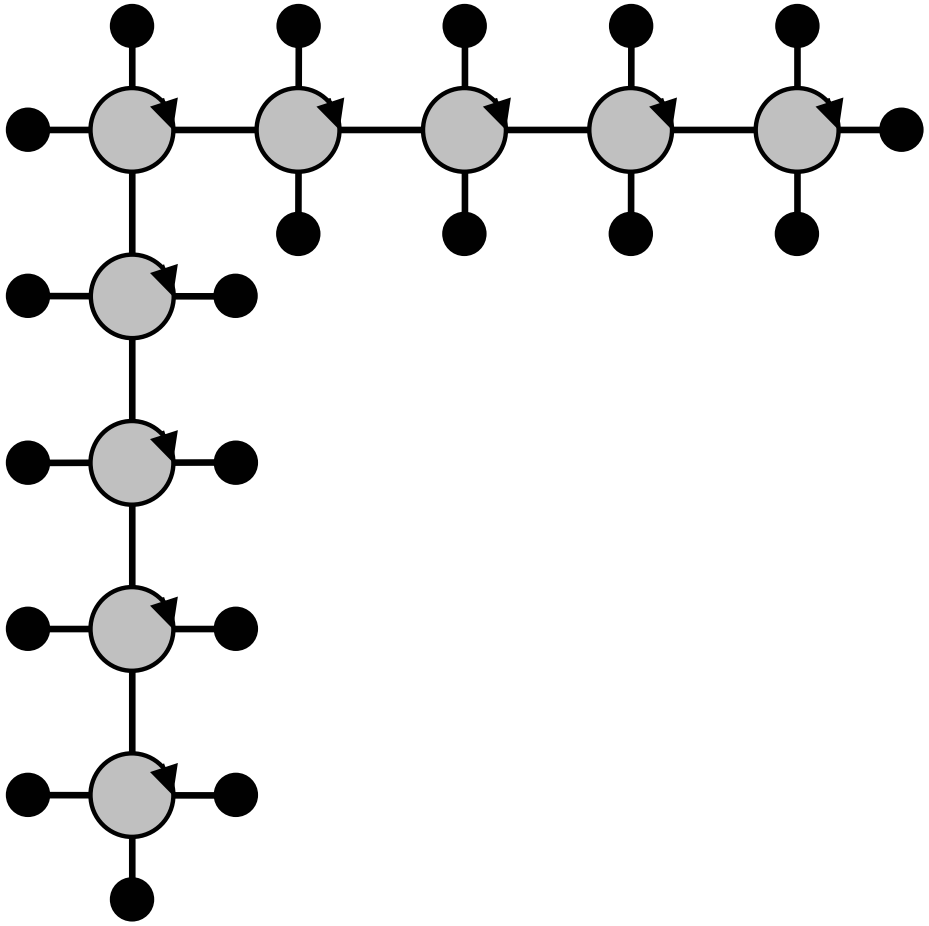
★ cf. Shape Types

Example 6: self-organizing 2-D grid

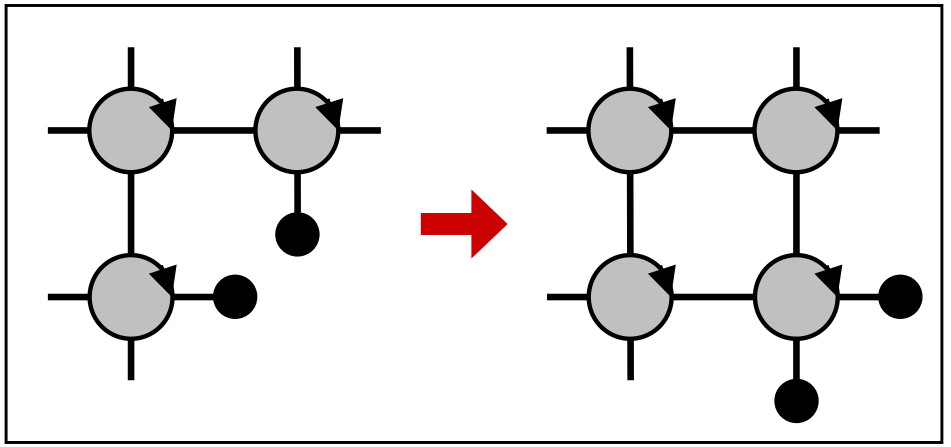


目的状態

Example 6: self-organizing 2-D grid

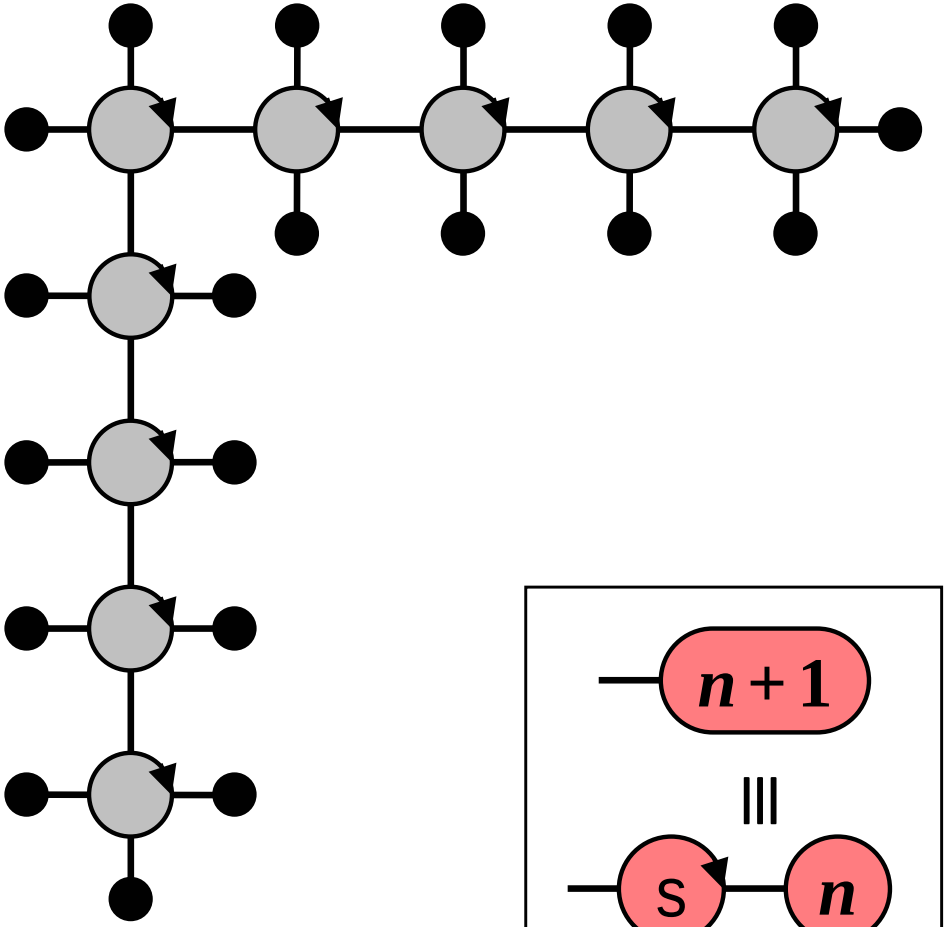


初期状態

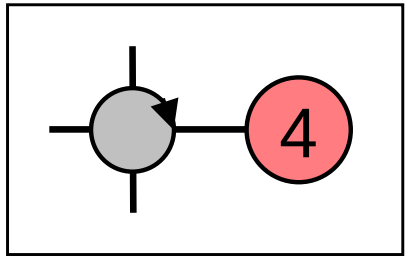


必要なルールは１本

Example 6: self-organizing 2-D grid

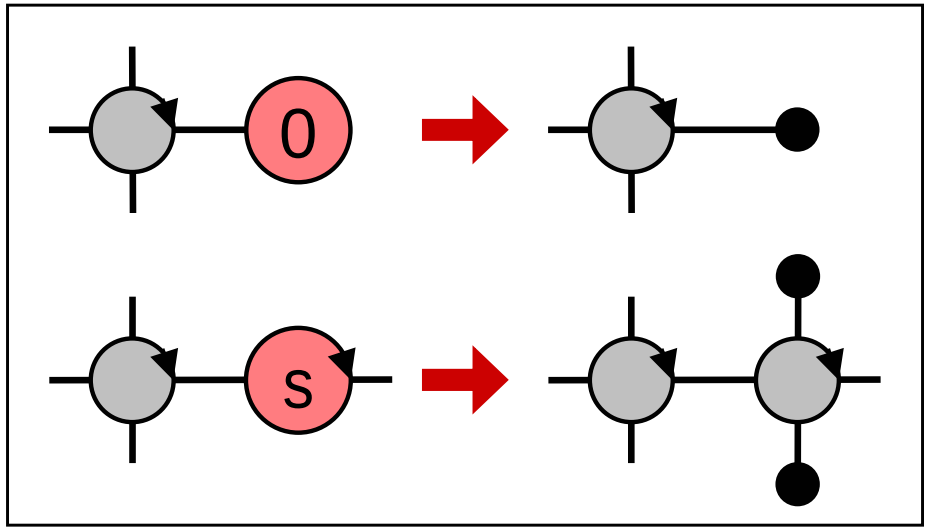


上端の横１列の生成法



初期状態

合同関係



変換規則

研究課題 (1)

- **多重集合をサポートする型体系の理論と実装**
  - ★ **目的：**
    - **安全性** (security) **のため**
    - **プログラムの性質の把握のため**
    - **処理系最適化のため**
  - ★ **扱う性質：**
    - **プロセス構造（リスト / 木 / 環 / …）**
    - **リンクの方向性（単方向実装が可能か）**
    - **マッチング方法（**active *vs*. passive, **パートナー探し）**
    - **膜から出る自由リンク**

研究課題 (2)

- *LMNtal* の実装
  - ★ プロセスの表現
  - ★ ルールのコンパイル技法
  - ★ 並列分散処理

研究課題 (3)

- **多重集合と会合に基づく他の計算モデルの埋込み，実装，関連付け**
  - ★ **特に π 計算，Ambient 計算，CHR**
  - ★ **既存CHRプログラムの解析**

- **言語設計上の課題**
  - ★ **例外処理機能の設計**
  - ★ **メタレベルアーキテクチャの設計**

研究課題 (4)

- **応用プログラムの記述**
  - ★ グラフアルゴリズム
  - ★ マルチエージェント（MOO）
  - ★ Webサービス
  - ★ 広域分散計算
  - ★ 分子計算シミュレータ
  - ★ グラフィクス
  - ★ 自己組織化に基づくプログラム
  - ★ amorphous computing

まとめ

- **言語*LMNtal* の四大要素は**
  - ★ links,
  - ★ (nested) multisets
  - ★ nodes,
  - ★ transformation.
- **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，データの統一を図った．**

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

- Andries, M. *et al.,* Graph Transformation for Specification and Programming. *Sci. Comput. Program*., Vol.34, No.1 (1999), pp.1-54.
- Banâtre, J.-P. and Le Métayer, D., Programming by Multiset Transformation. *Commun. ACM*, Vol.35, No.1 (1993), pp.98-111.
- Drewes, F., Hoffmann, B. and Plump, D., Hierarchical Graph Transformation. *J. Comput. Syst. Sci.*, Vol.64, No.2 (2002), pp.249-283.
- 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.

参考文献

- Fradet, P. and Le Métayer, D., Shape Types. In *Proc. POPL'97*, ACM Press, 1997, pp.27-39.
- Frühwirth, T., Theory and Practice of Constraint Handling Rules. *J. Logic Programming*, Vol.37, No.1-3 (1998), pp.95-138.
- Paun, Gh., Computing with Membranes. *J. Comput. Syst. Sci.,* Vol.61, No.1 (2000), pp.108-143.
- 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.
- Ueda, K., Resource-Passing Concurrent Programming. In *Proc. TACS 2001*, LNCS 2215, Springer, 2001, pp.95-126.