LMNtal

— From Concurrent Constraint Programming to Concurrent Hierarchical Graph Rewriting

Kazunori Ueda, Waseda Univ.

(Joint work with Norio Kato)

October 2004

LMNtal (pronounce: "*elemental*")

$\mathcal{L}$ = logical links

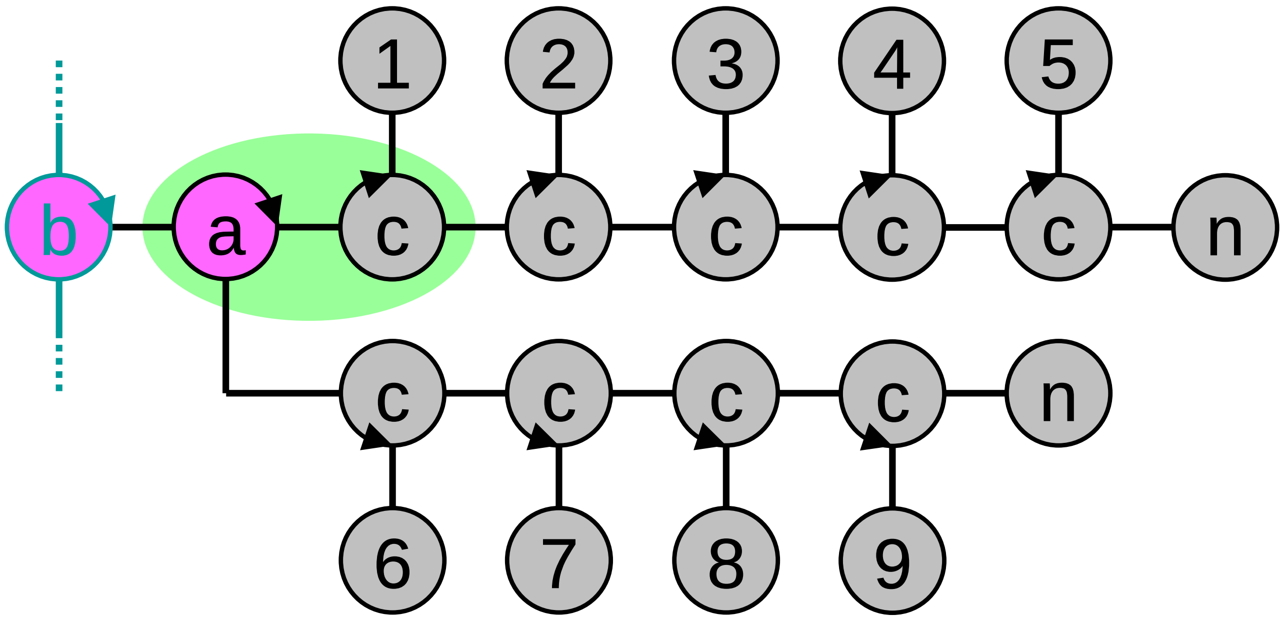
$\mathcal{M}$ = multisets/membranes

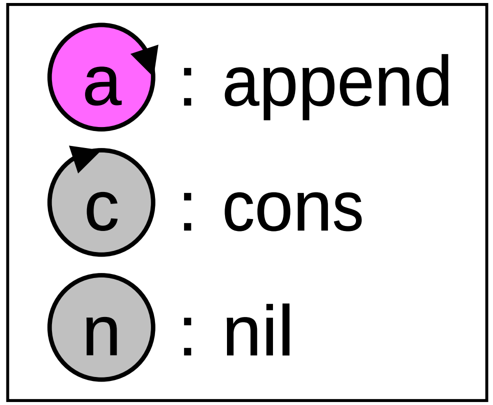
$\mathcal{N}$ = nested nodes

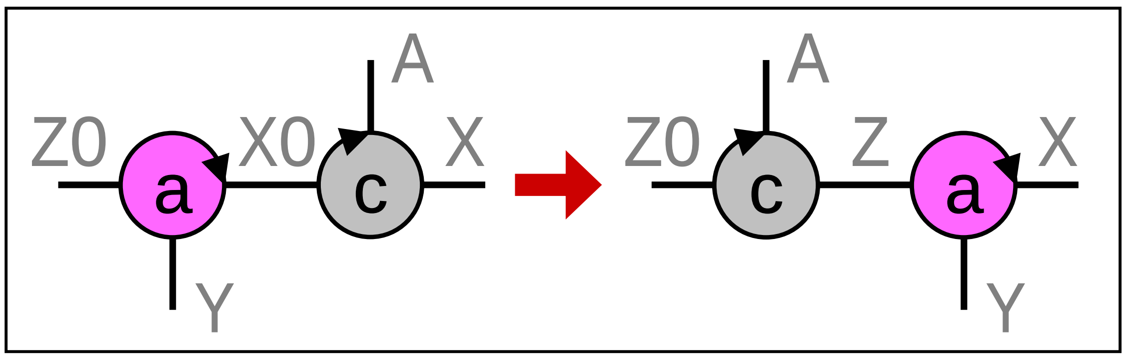
ta = transformation

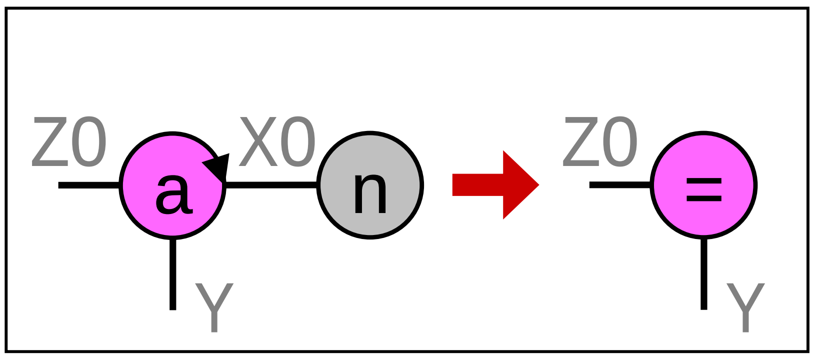
l = language

Example 1: append

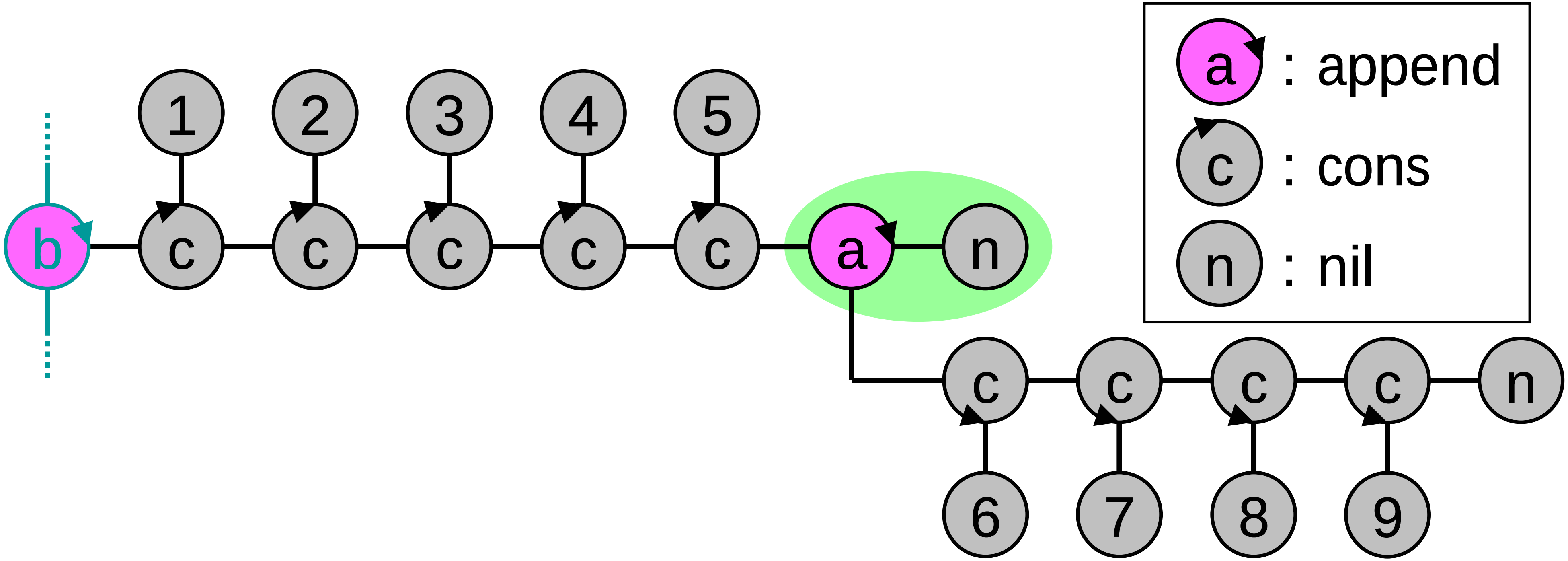


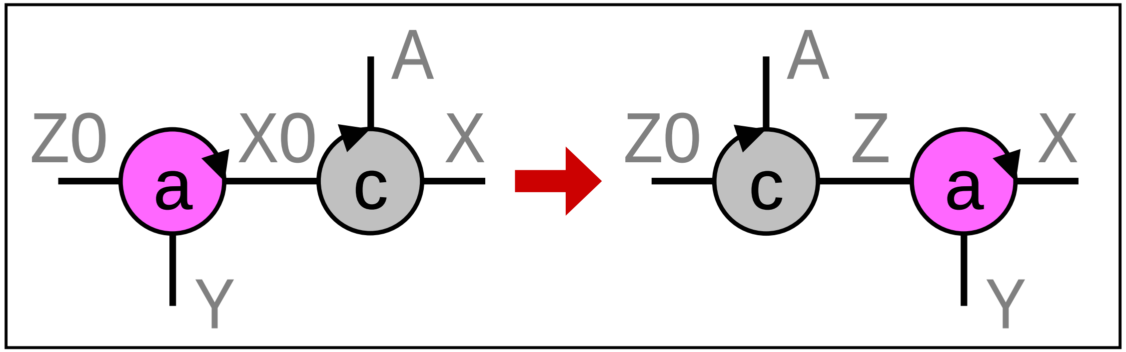


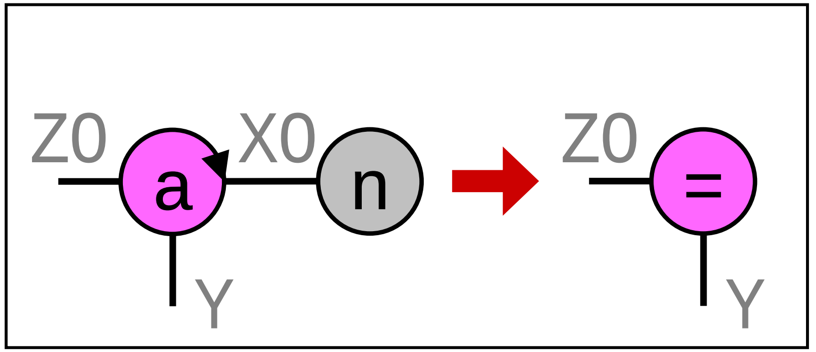




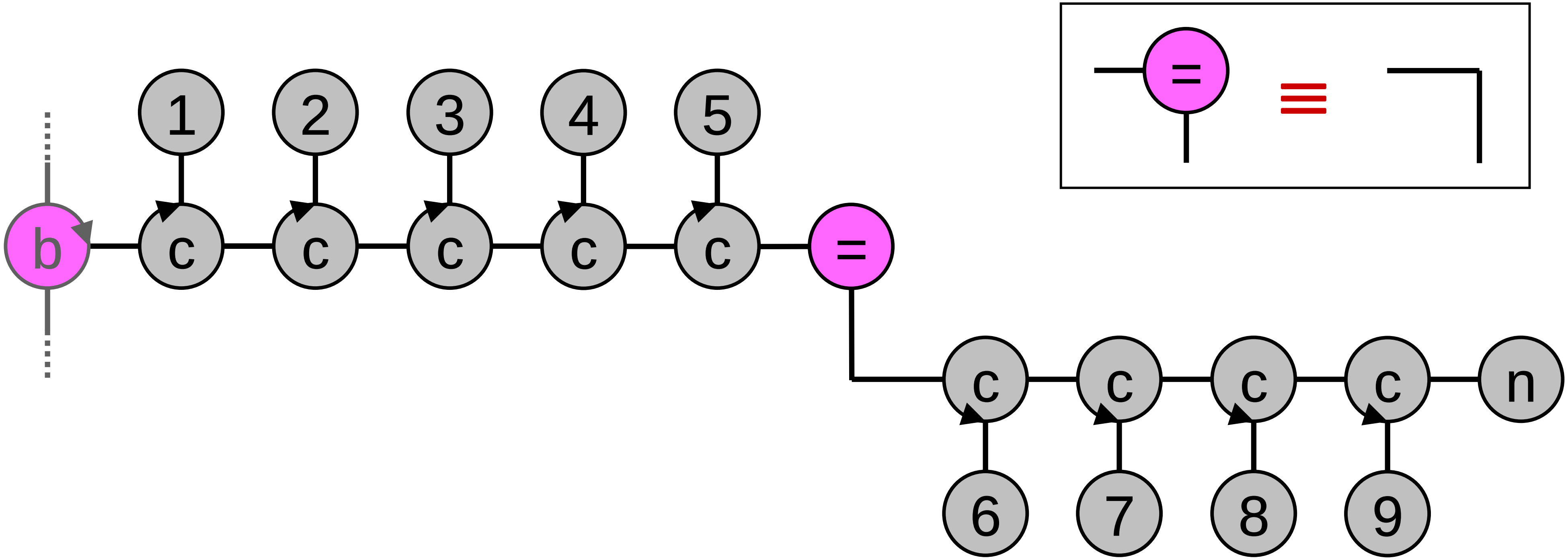
Example 1: append

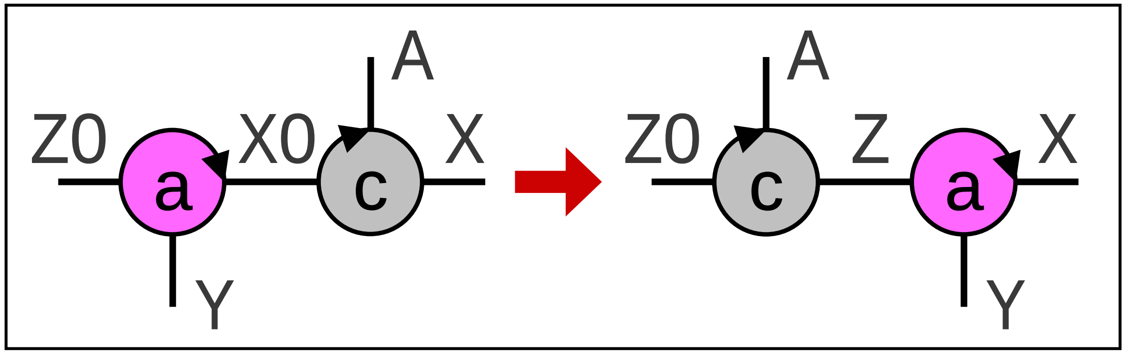


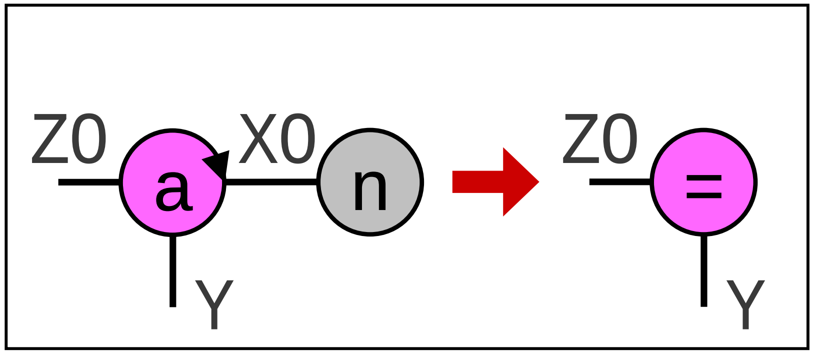




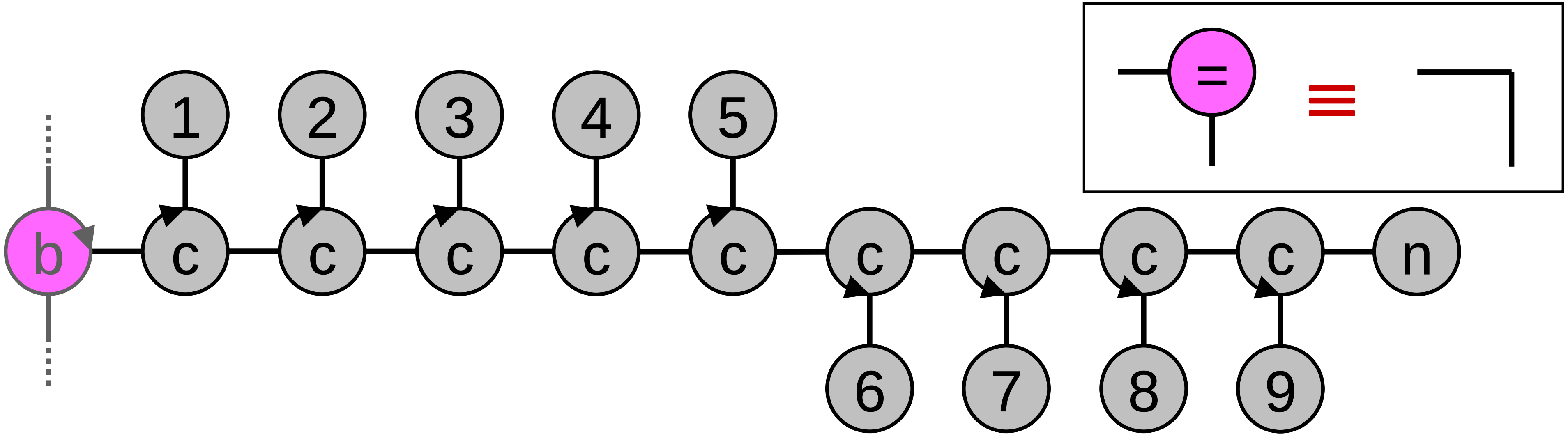
Example 1: append

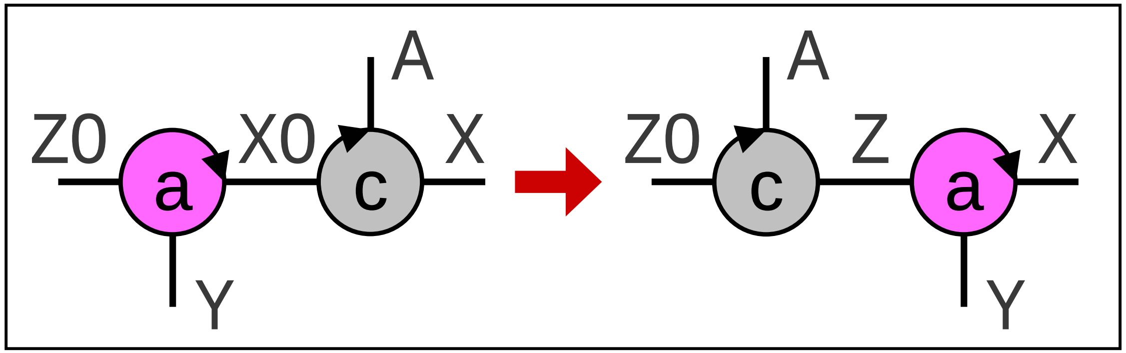


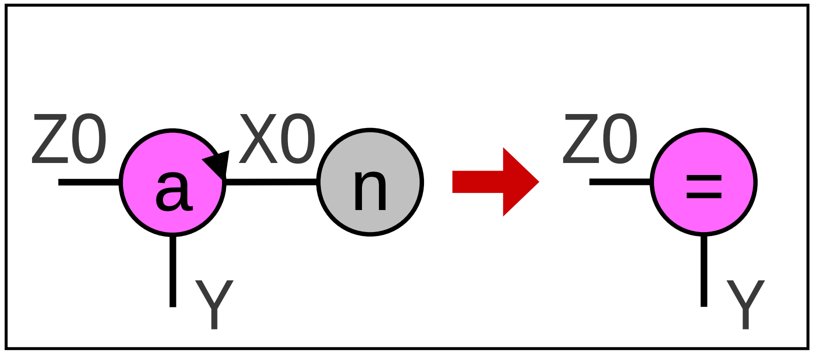




Example 1: append







append in *LMNtal*

```
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. Guarded Horn Clauses (GHC) version

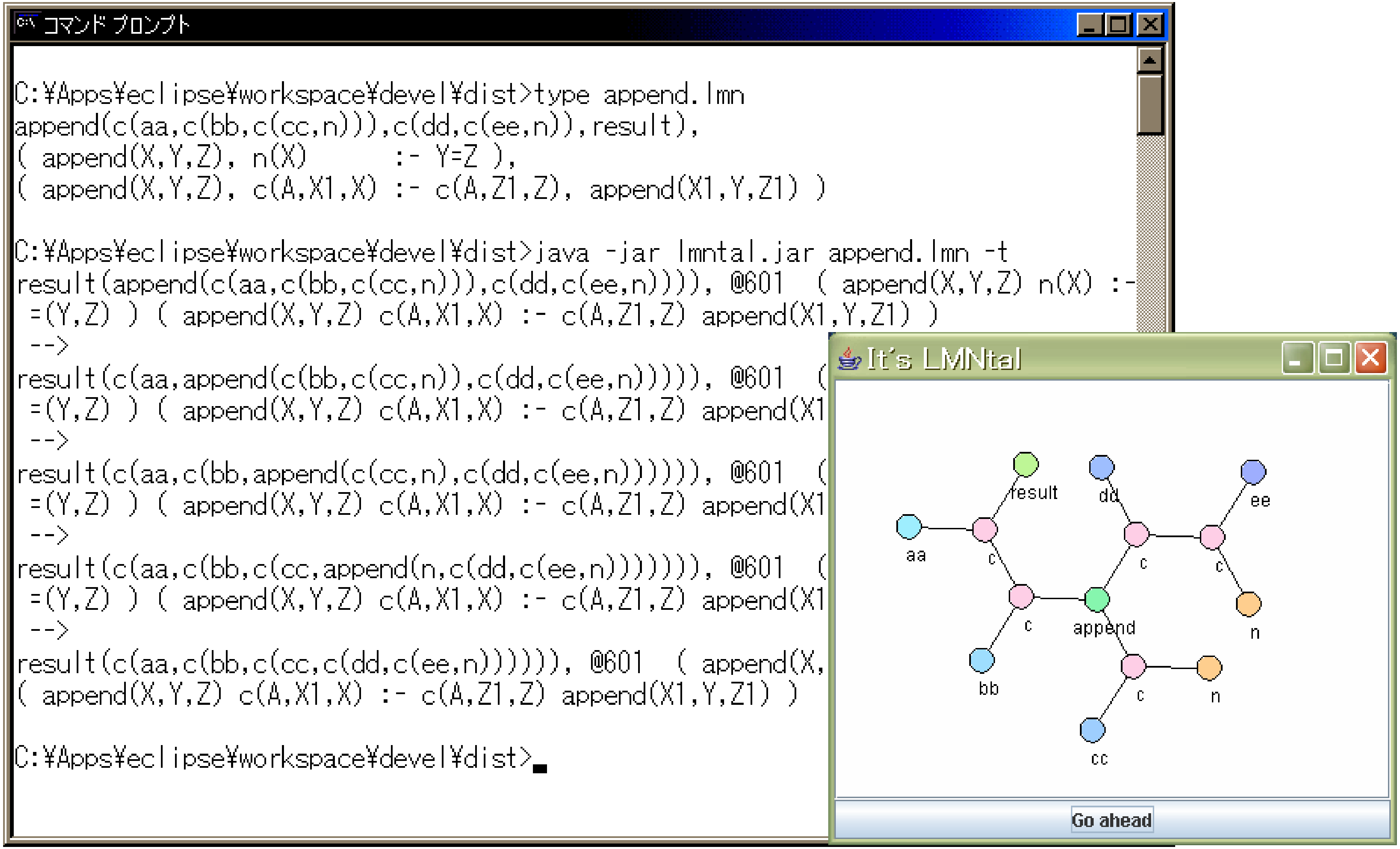
```
append(X0,Y,Z0) :- X0=[A|X]  |
        Z0=[A|Z], append(X,Y,Z).
append(X0,Y,Z0) :- X0=[ ]  |  Y=Z0.
```

- No distinction between append and c(ons)
- cf. (small fragment of) CHR, Interaction Nets

Design goals of *LMNtal*

- *A "Turing Machine" for universal computing environments (from wide-area to nanoscale)*
- Unifying model of concurrency
  - ★ e.g., processes = messages = data
- Simple and versatile
  - ★ Computation is direct manipulation of graphs
  - ★ Membranes express multisets and locality
  - ★ Allows programming by self-organization
- Implementation (with interface to Java) available
  - ★ **http://www.ueda.info.waseda.ac.jp/lmntal/**

Running, tracing and visualizing append



Background

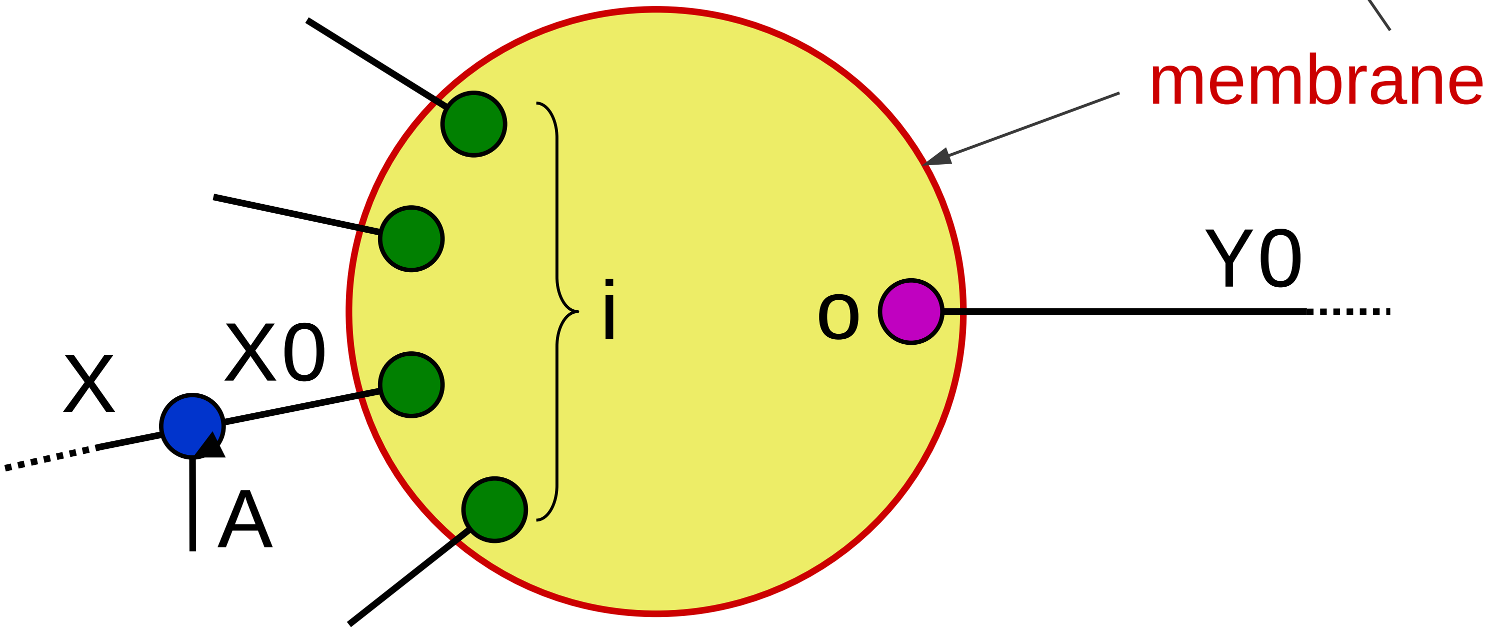
- Concurrent Logic Programming (early 1980’s)
  - ★ Channel mobility using logical variables
  - ★ Various type systems (including linearlity) and implementation experiences
- Concurrent Constraint Programming (late 1980’s)
  - ★ Generalization of data domains (FD, multisets, . . .)
- CHR (Constraint Handling Rules) (early 1990’s)
  - ★ Allows multisets of goals in rule heads
  - ★ An expressive multiset rewriting language
  - ★ Many applications (esp. constraint solvers)
  - ★ Lacks reaction control mechanisms such as termination detection and hierarchies

Models and languages with multisets and symmetric join

- Petri Nets (1962)
- Production Systems and RETE match
- Graph transformation formalisms
- CCS, CSP
- Concurrent logic/constraint programming
- Linda
- Linear Logic languages
- Interaction Net
- Chemical Abstract Machine, reflexive CHAM, Join Calculus
- Gamma model
- Constraint Handling Rules
- Mobile ambients
- P-system, membrane computing
- Amorphous computing
- Bigraphical reactive system (2001)

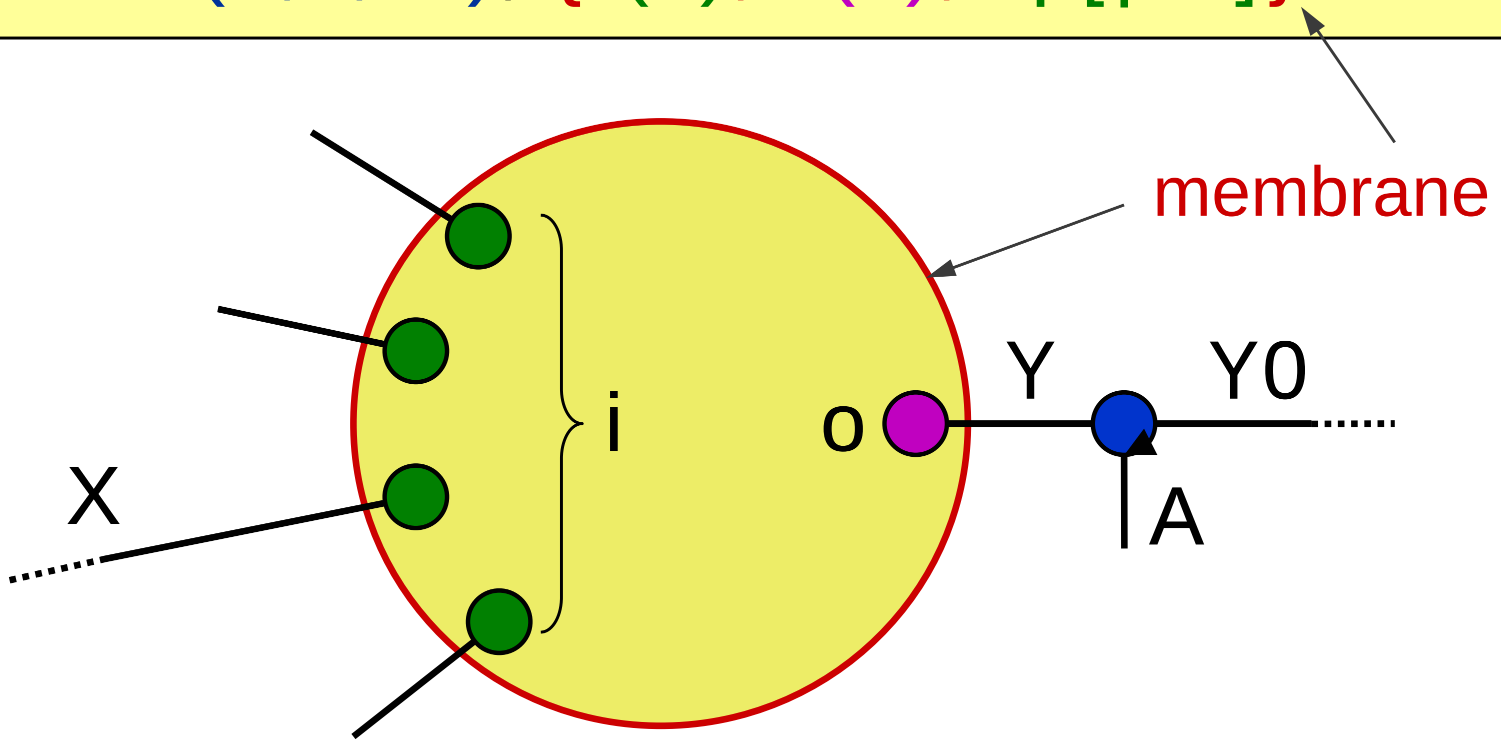
Example 2: N-to-1 stream communication

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



Example 2: N-to-1 stream communication

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



★ The number of free links in { } remain unchanged

Elements of *LMNtal* (1)

1. Nodes (= atoms) with links

- ★ Links are linear, zero-assignment logical variables
  - ■ linear = occurring twice (1-to-1 comm.)
  - ■ logical = link identity changes after message sending (↔ π-calculus)
  - ■ zero-assignment = not instantiated (↔ logic programming)
  - ■ private
  - ■ not directed (cf. chemical bonds)
- ★ Links of a node are ordered

Elements of *LMNtal* (1)

- Links are used
  - (a) to represent (private) communication channels
  - (b) to represent data structures (= graphs)
  - (c) to find partners in multiset rewriting
    - O(1) if linked
    - can be O(*n*) if not linked
  - (d) to represent hyperlinks (using membranes; see next slide)

Elements of *LMNtal* (2)

2. First-class multisets (using membranes)

- ★ Not many languages feature multisets as *first-class* citizens
- ★ Used for :
  - representing records (feature structures)
  - localization and logical management of computation
    - cf. ambients, join calculus, Unix processes

Elements of *LMNtal* (3)

3. Rewrite rules

- Can be put in a membrane to realize
  - local reaction
  - process mobility
- Design issue: proper handling of free links
  - cf. graph grammars and transformation, logic programming

Syntax: preliminaries

- Two presupposed syntactic categories:
  - ★ *X* : links (or link variables)
    - In concrete syntax, start with capital letters
  - ★ *p* : names (including numbers)
    - In concrete syntax, use identifiers different from links
- The name "**=**" (called a connector) is the only reserved symbol in *LMNtal*

Syntax of *LMNtal* processes

Not in Flat LMNtal

- P ::= **0** (null)
  | $p(X_1,\ldots,X_m)$ $(m \geq 0)$ (atom)
  | P, P (molecule)
  | { P } (cell)
  | T :- T (rule)
- **Link condition:** Each link in P (except those in reaction rules) occurs at most twice.

★ Free link of P = link occurring only once

★ P is closed = has no free links

Syntax of *LMNtal* process templates

Not in Flat LMNtal

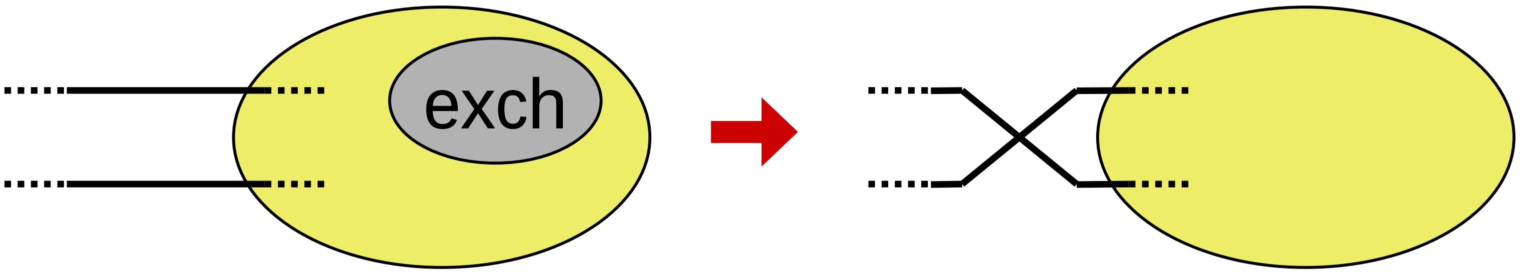
■ T ::=	$\mathbf{0}$		(null)
|	$p(X_1,\ldots,X_m)$	$(m \geq 0)$	(atom)
|	T, T		(molecule)
|	{ T }		(cell)
|	T :- T		(rule)
|	@p		(rule context)
|	\$$p[X_1,\ldots,X_m | A]$	$(m \geq 0)$	(process context)
|	$p(*X_1,\ldots,*X_m)$	$(m > 0)$	(aggregate)

■ (residual args) A ::= [] (empty)

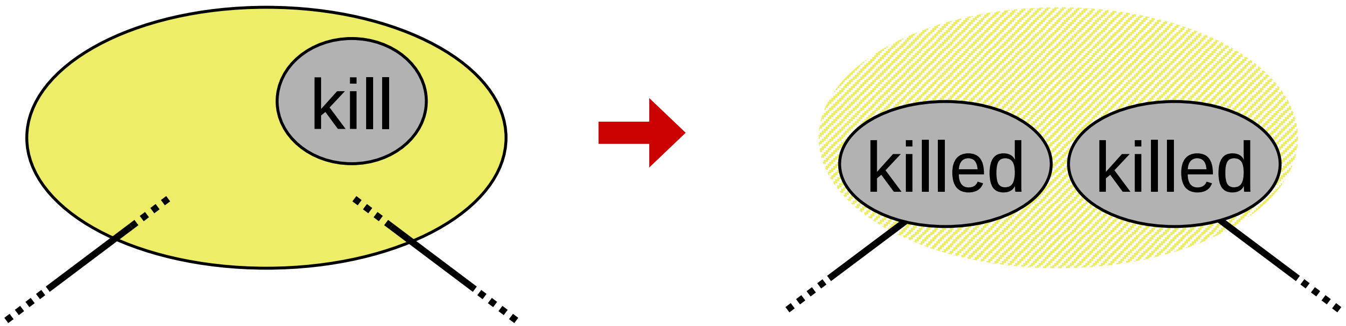
| *X (bundle)

Process contexts, examples

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

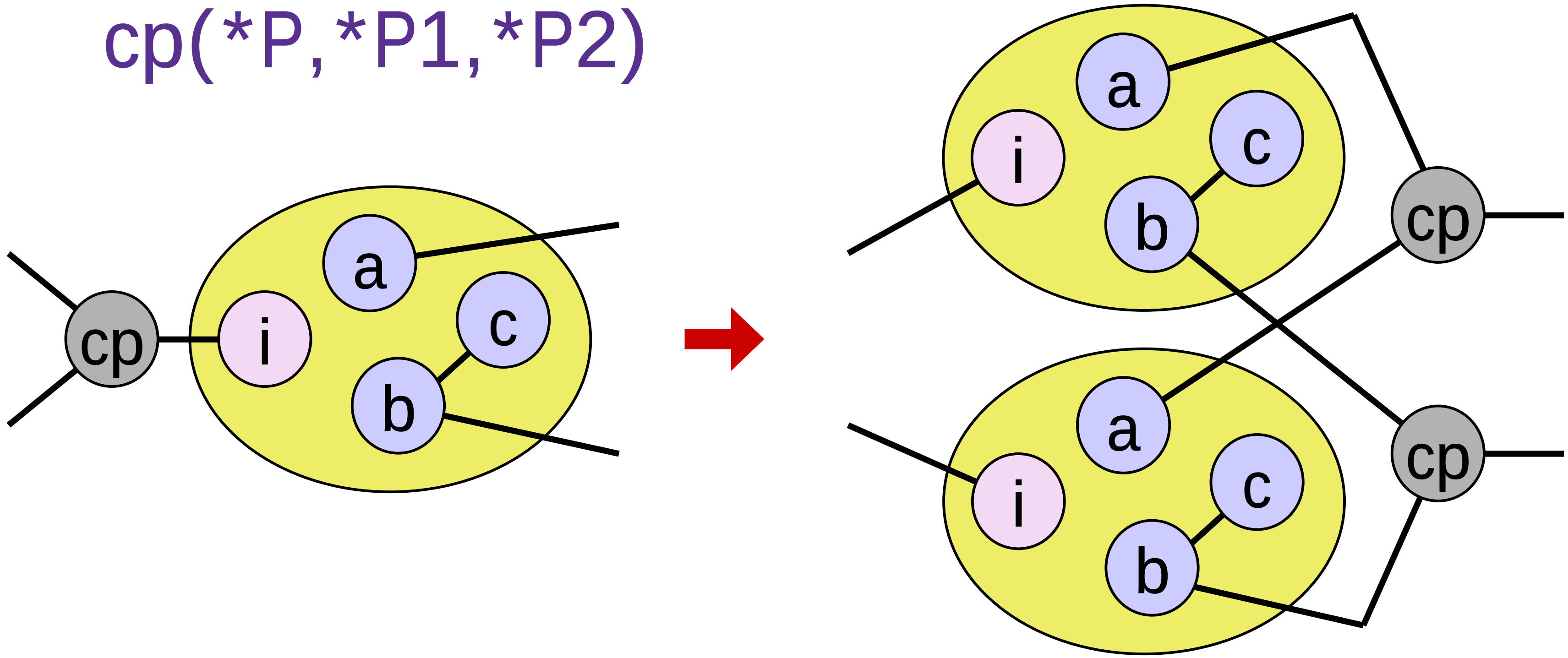


- {kill, $a[|*X]} :- killed(*X)



Process contexts, examples

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



Syntactic sugar

c(A1,X1,X0), c(A2,X2,X1), c(A3,X3,X2), n(X3)

≡ c(A1,c(A2,c(A3,n)),X0)

≡ X0=Y, c(A1,c(A2,c(A3,n)),Y)

≡ X0=c(A1,c(A2,c(A3,n)))

Structural congruence ($\equiv$)

(E1) $\mathbf{0}, P \equiv P$

(E2) $P, Q \equiv Q, P$

(E3) $P, (Q, R) \equiv (P, Q), R$

(E4) $P \equiv P[Y/X]$ if X is a local link of P

(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) $X = Y, P \equiv P[Y/X]$

if P is an atom and X is a free link of P

(E10) $\{X = Y, P\} \equiv \{P\}, X = Y$

if X is a free link of P and Y is not a free link of P

Reduction semantics

(R1) $$\frac{P \to P'}{P, Q \to P', Q}$$

(R2) $$\frac{P \to P'}{\{P\} \to \{P'\}}$$

(R3) $$\frac{Q \equiv P \quad P \to P' \quad P' \equiv Q'}{Q \to Q'}$$

(R4) $\{X{=}Y, P\} \to X{=}Y, \{P\}$

X and *Y* are free links of (*X*=*Y*, *P*)

(R5) $X{=}Y, \{P\} \to \{X{=}Y, P\}$

X and *Y* are free links of *P*

(R6) $T\theta, (T :\!- U) \to U\theta, (T :\!- U)$

θ is to instantiate process & rule variables. Links are matched using α -conversion.

Reduction semantics

- Can p(A,A) be reduced using p(X,Y) :- q(Y,X) ?
  - ★ The rule can't be α-converted to the form p(A,A) :- ...
  - ★ However, because p(A,A) is equivalent to p(A,B), A=B (B a fresh link), it can be reduced as:

$$
\begin{aligned}
& p(A,A) \\
\equiv\ & p(A,B),\ A=B \\
\rightarrow\ & q(B,A),\ A=B \\
\equiv\ & q(A,A)
\end{aligned}
$$

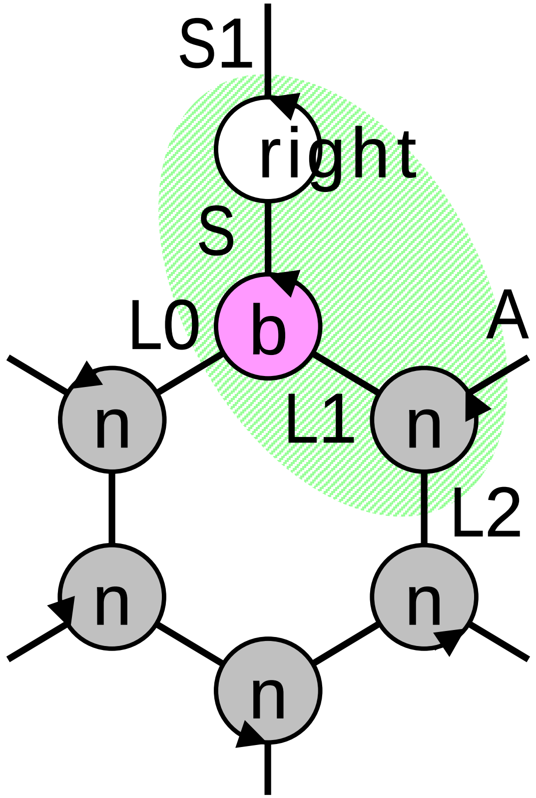
Example 3: circular data structures

- Bidirectional circular buffer:

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

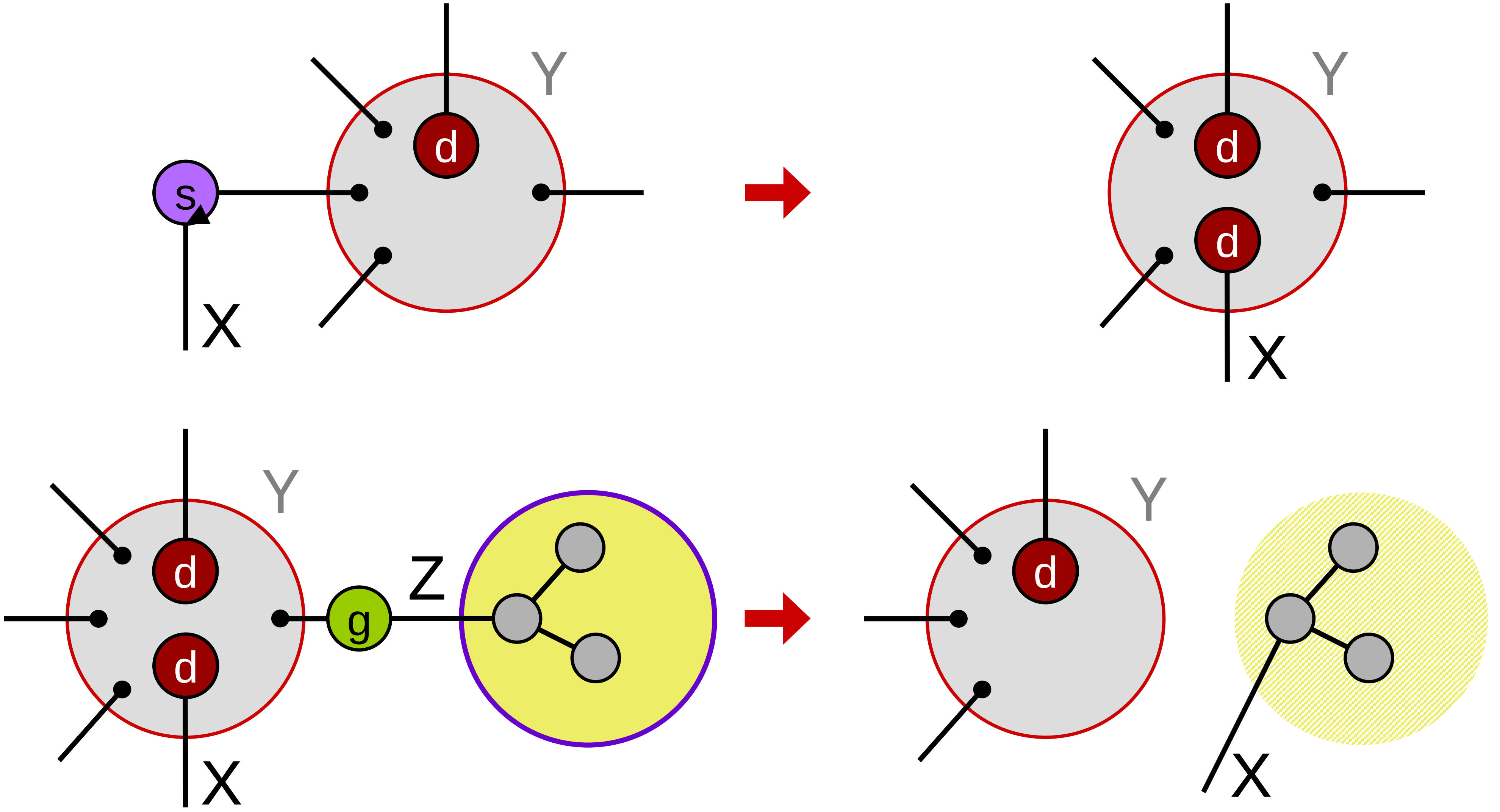
★ *S* acts as an interface link

```
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)
```

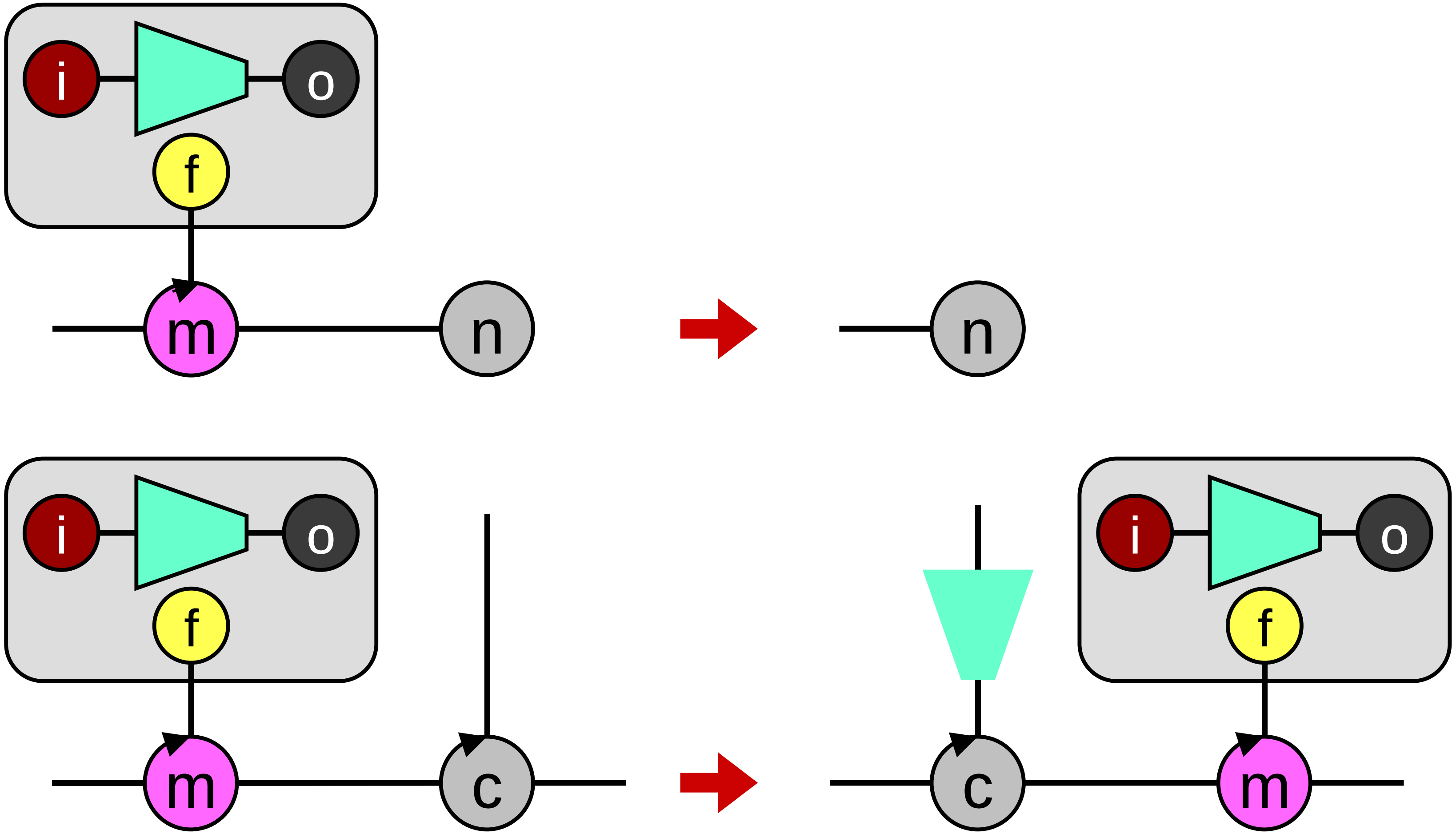


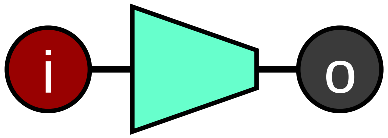
★ cf. Shape Types

Example 4: asynchronous π-calculus



Example 5: map function



★ 

is lambda abstraction

Extension: termination detection

- Syntax, expanded

$$P \ ::= \ \dots \mid \{P\}/ \qquad \text{(quiet cell)}$$

$$T \ ::= \ \dots \mid \{T\}/ \qquad \text{(quiet cell)}$$

- Structural congruence, expanded

$$\{P\} \equiv \{P\}/ \quad \text{if } P \not\rightarrow$$

  - ★ An irreducible cell can show the flag "/"
  - ★ Irreducibility of a cell can only be checked from outside the cell

Future Work

- Language
  - ★ Constructs for distributed and real-time computing
- Foundations
  - ★ Type systems
  - ★ Theory of computational resources
  - ★ *Human-oriented* program verification
- Implementation
  - ★ Optimizing complation of sequential core
  - ★ Parallel and distributed implementation
  - ★ Interoperability
  - ★ Integration with static analysis

➔ **"Theory meets practice, logic meets physics."**

Conclusions

- The "four elements" of *LMNtal* are:
  - ★ (logical) links,
  - ★ multisets/membranes,
  - ★ (nested) nodes, and
  - ★ transformation.
- Inspired by communication using logical variables, we have designed a simple language model for the unified treatment of processes, messages and data.