Hierarchical graph rewriting as a unifying model and language of concurrency

Kazunori Ueda, Waseda Univ.

November 2006

LIX Colloquium on Emerging Trends in Concurrency Theory
École Polytechnique, Paris, November 13-15, 2006

LMNtal (pronounce: "elemental")

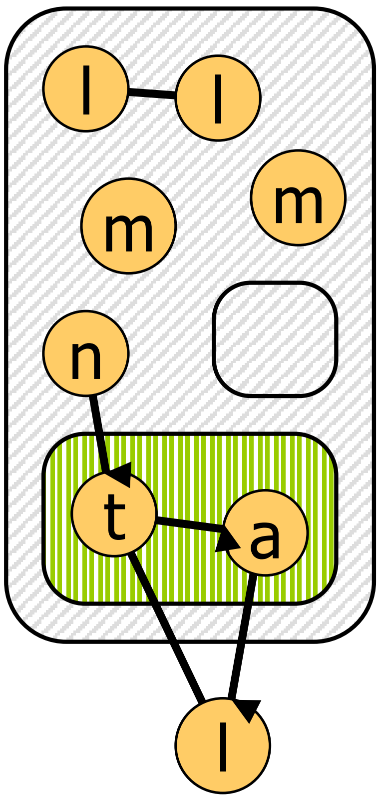
$\mathcal{L}$ = "logical" links

$\mathcal{M}$ = multisets/membranes

$\mathcal{N}$ = nested nodes

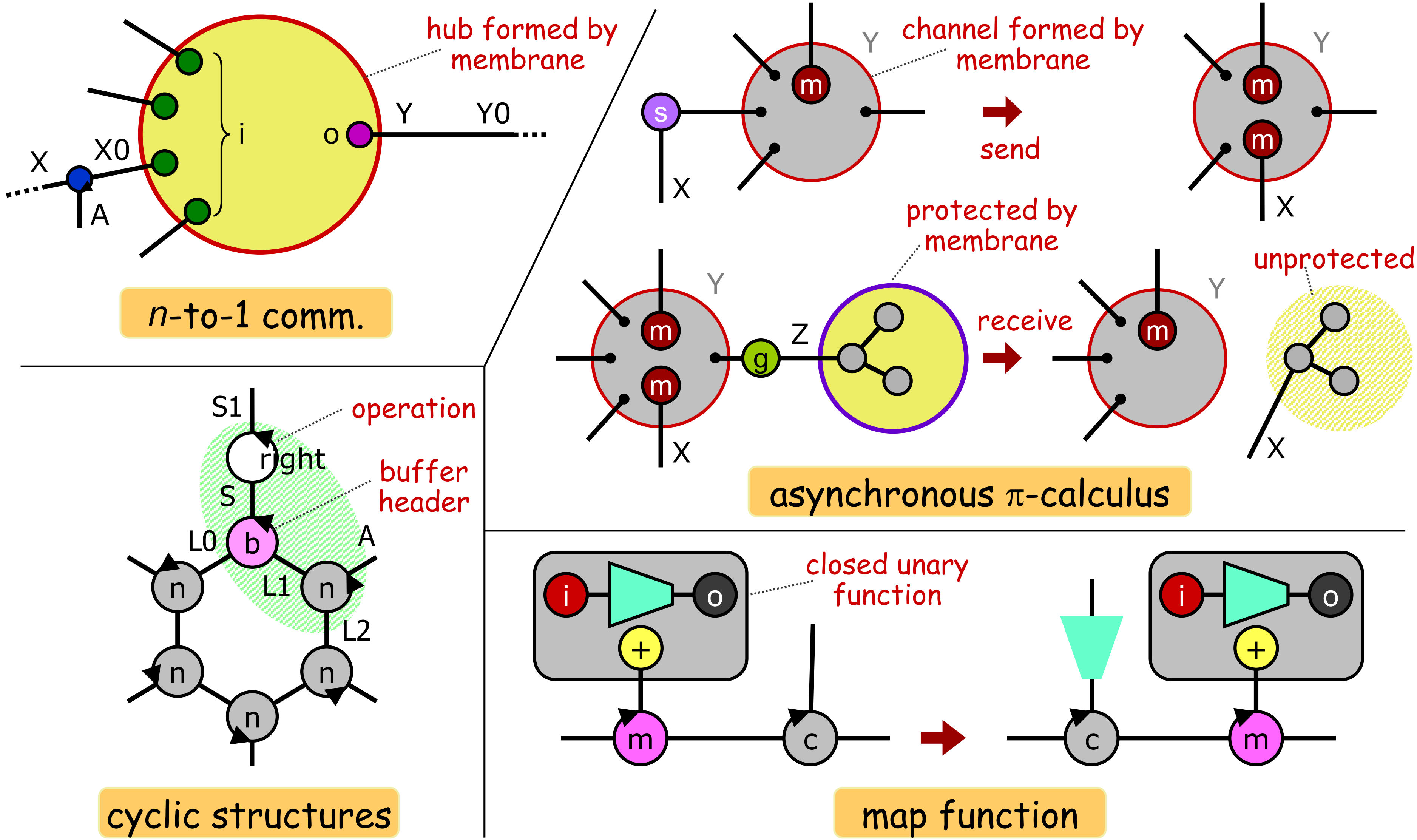
ta = transformation

$\mathcal{l}$ = language



hierarchical graph

Diagrammatic representation of computation



Models and languages with multisets and symmetric join

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

Models and languages with membranes + hierarchies

- Petri Nets
- Production Systems and RETE match
- Graph transformation formalisms *
- CCS, CSP
- Concurrent logic/constraint programming
- Linda *
- Linear Logic languages
- Interaction Net
- Chemical Abst. Machine, reflexive CHAM, Join Calculus
- Gamma model
- Constraint Handling Rules
- Mobile ambients
- P-system, membrane computing
- Amorphous computing
- Bigraphical reactive system
- Seal calculus
- Kell calculus
- Brane calculi

* : some versions feature hierarchies

LMNtal in a nutshell

- A rule-based concurrent language for expressing and rewriting connectivity and hierarchy
- Unifying model of X-calculi (X = lambda, pi, ambient, ...) and multiset rewriting
- Computation is manipulation of diagrams
  - ★ Links express 1-to-1 connectivity
  - ★ Membranes express hierarchy and locality
  - ★ Allows programming by self-organization
  - ★ Good also for data/knowledge representation
  - ★ Low entry barrier

LMNtal: Language and implementation

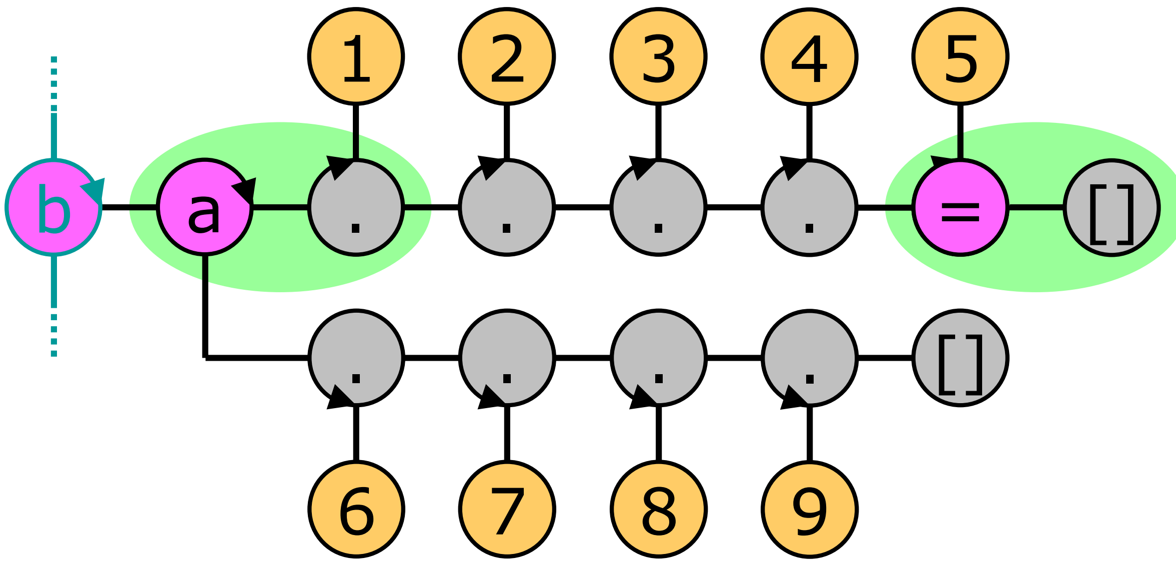
- Language
  - ★Developed since 2002, tested from many angles
    - K. Ueda and N. Kato, LNCS 3365
- Implementation
  - ★Translator to Java running on JDK 1.5
    - **http://www.ueda.info.waseda.ac.jp/lmntal/**
  - ★>40,000 LOC, ready to use
  - ★Dedicated intermediate code
  - ★Features: Module systems / Foreign-language interface to Java / Visualizer / interactive mode / optimizer / library APIs etc.

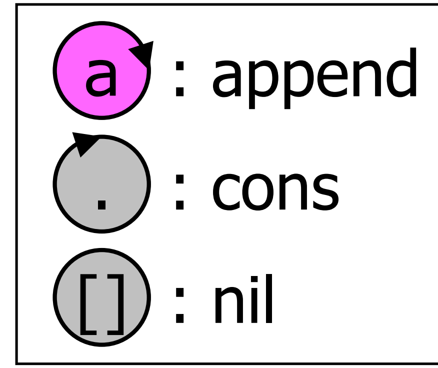
Hierarchical multiset rewriting

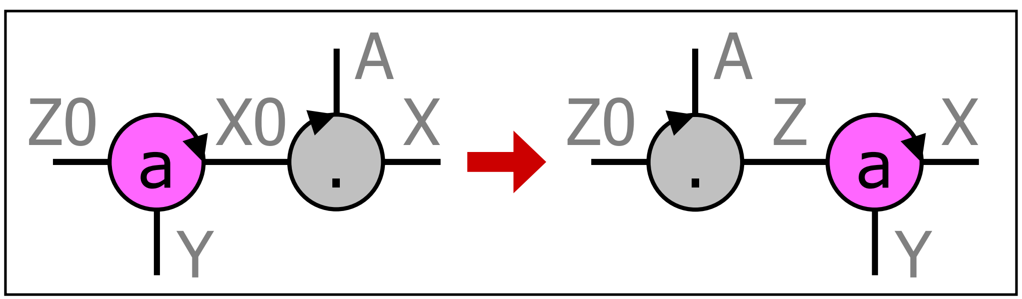
```
$ lmntal --immediate
        LMNtal version 0.81.20060613
Type :h to see help.
Type :q to quit.

# 1,1,1, {1,1,1,1,1, {1,1,1}, (1,1:-2)}
1,1,1, {2,2,1,{1,1,1}, @601}

# {out, a,b,c}, d, {e,f}, ({out, $p[]}:-$p[]).
d,a,c,b, {f,e}, @603
```

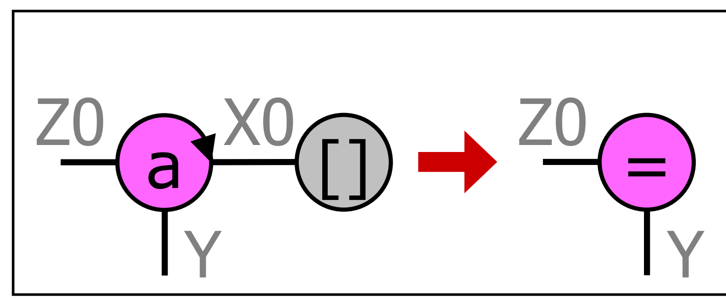
Process Context
(local context in a membrane)

List concatenation





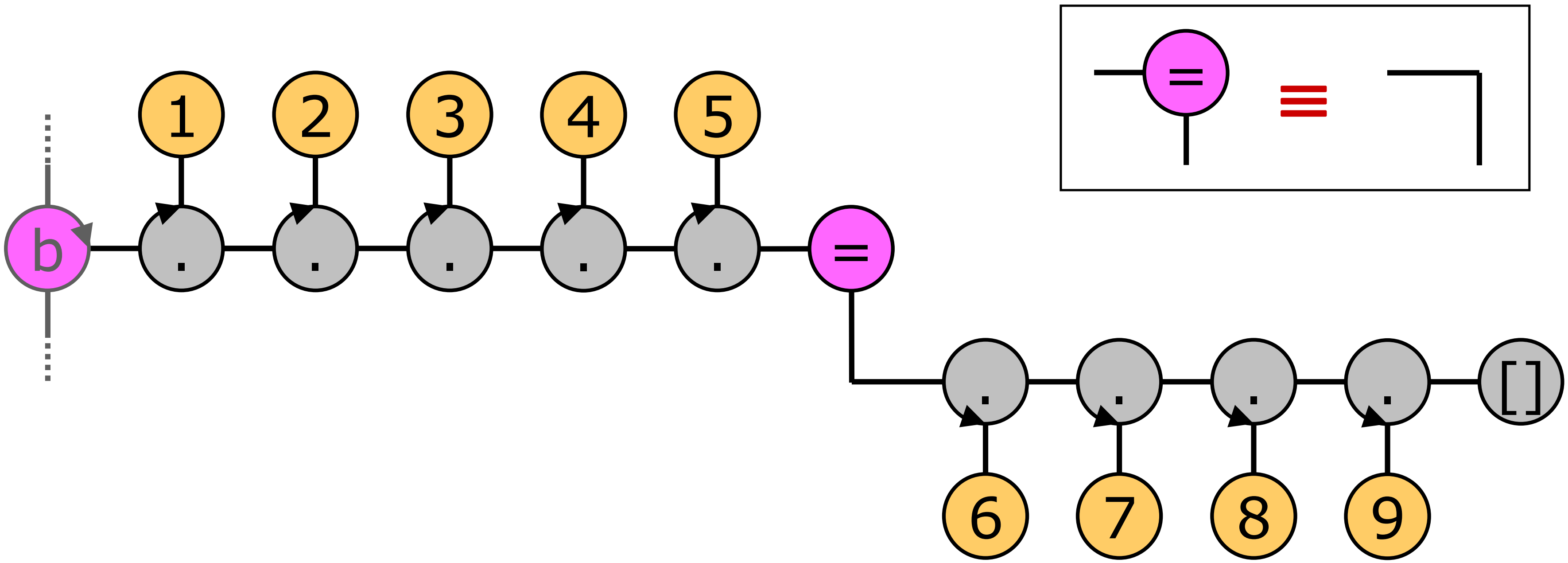


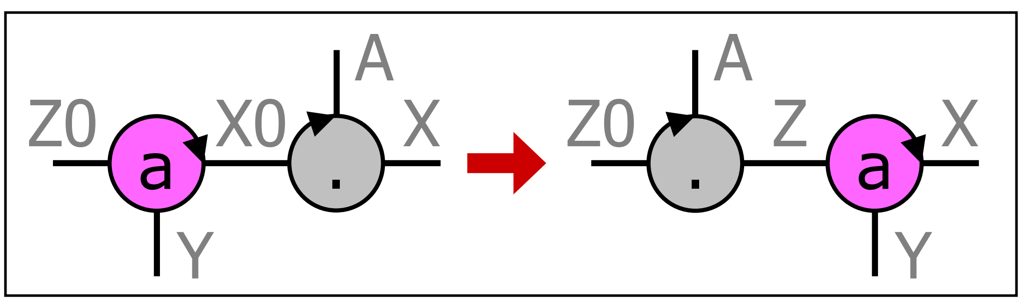


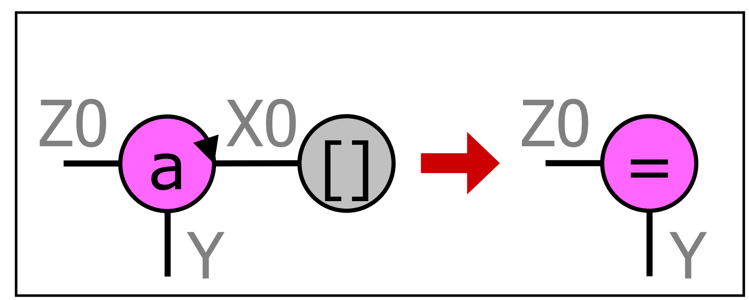
a(X0,Y,Z0), '.'(A,X,X0) :- '.'(A,Z,Z0), a(X,Y,Z)

a(X0,Y,Z0), '.'(X0) :- Y=Z0

List concatenation





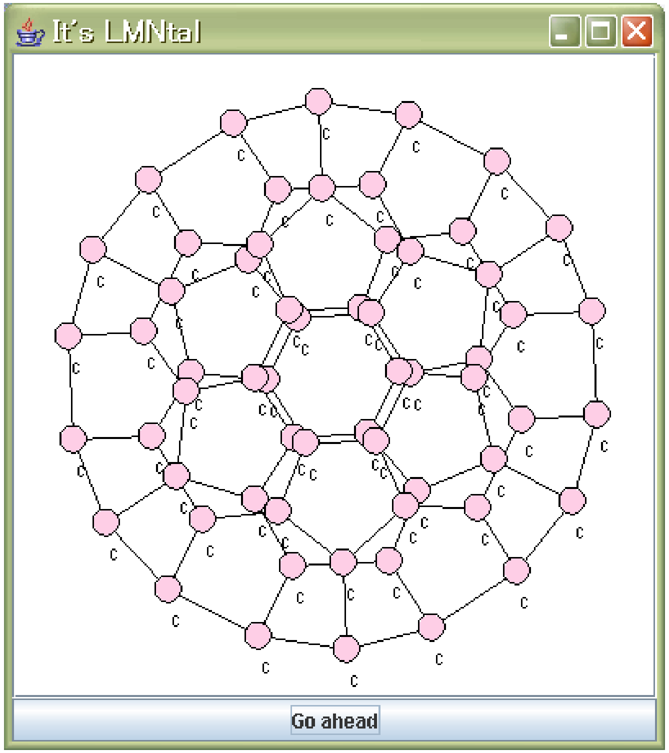


a(X0,Y,Z0), '.'(A,X,X0) :- '.'(A,Z,Z0), a(X,Y,Z)

a(X0,Y,Z0), '.'(X0) :- Y=Z0

Fullerene (C_{60})

```
 /* icosahedron */
dome(L0,L1,L2,L3,L4,L5,L6,L7,L8,L9) :-
    p(T0,T1,T2,T3,T4),  p(L0,L1,H0,T0,H4),
    p(L2,L3,H1,T1,H0), p(L4,L5,H2,T2,H1),
    p(L6,L7,H3,T3,H2), p(L8,L9,H4,T4,H3).

dome(E0,E1,E2,E3,E4,E5,E6,E7,E8,E9),
dome(E0,E9,E8,E7,E6,E5,E4,E3,E2,E1).

/* icosahedron -> fullerene */
p(L0,L1,L2,L3,L4) :-
  c(L0,X0,X4), c(L1,X1,X0), c(L2,X2,X1), c(L3,X3,X2), c(L4,X4,X3).
```



Syntax: preliminaries

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

Syntax of *LMNtal* processes

Not in Flat LMNtal

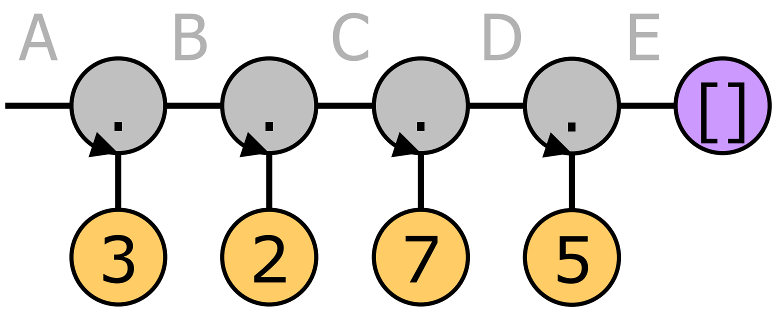
- P ::= $\mathbf{0}$ (null)
  - | $p(X_1, \ldots, X_m)$ $(m \geq 0)$ (atom)
  - | P, P (molecule)
  - | $\{ P \}$ (cell)
  - | T :- T (rule)
- **Link condition:** Each link name in P occurs at most twice and each link name in a rule occurs exactly twice.
  - ★ Free link of P = link occurring only once
  - ★ P is closed = has no free links

Syntax of *LMNtal* process templates

Not in Flat LMNtal

- T ::= **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)

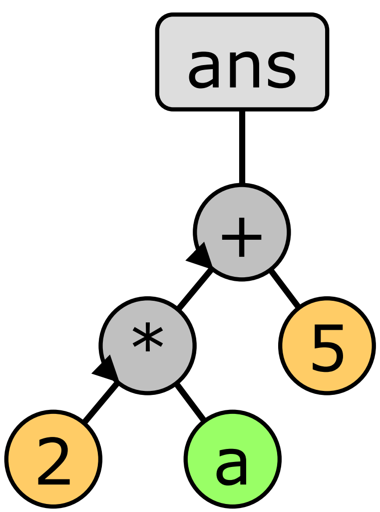
Lists, trees, cells (bags)



'.'(3,B,A), '.'(2,C,B),
'.'(7,D,C), '.'(5,E,D), '[]'(E)

or

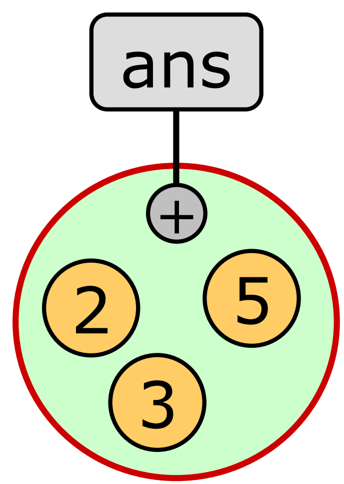
A= '.'(3,'.'(2,'.'(7,'.'(5,'[]'))))
A=[3 | [2 | [7 | [5 | []]]]]
A=[3,2,7,5]



ans(A), '+'(B,C,A),
'*'(D,E,B),
2(D), a(E), 5(C)

or

ans('+'('*'(2,a),5))
ans(2*a+5)



ans(A),
{+A, 2,3,5}

or

ans({2,3,5})

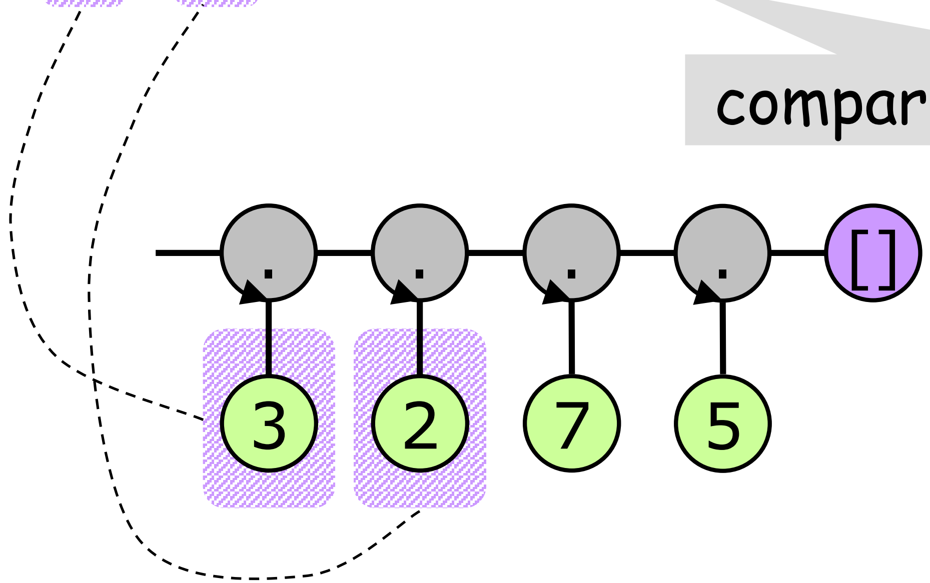
Bubblesort

guard

typed process context

```
L=[$x , $y|L2] :- $x > $y | L=[$y,$x|L2].
```

compare and swap if **$x > $y**



Scheduling achieves $O(N^2)$ sequential complexity

Structural congruence (≡)

multisets

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

structural

connectors

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

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

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

if X and Y are free links of P

(R6) $T\theta, (T \text{:-} U) \to U\theta, (T \text{:-} U)$

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

The other extreme: Axiomatic set theory natural numbers

```
$ lmntal --immediate --remain --hideruleset
      LMNtal version 0.81.20060613

Type :h to see help.
Type :q to quit.

# inc, {$p[]} :- {$p[], {$p[]}}.

# {}
{}
# inc
{{}}
# inc
{{{}}, {}}
# inc
{{}, {{}}, {{{}}, {}}}
```

Foreign-language interface

inline atom

```
H=io.print(Object, String) :-
  class(Object, "java.io.PrintWriter"), unary(String) |
  H=[:/*inline*/
   try {
     java.io.PrintWriter pw = (java.io.PrintWriter)
       ((ObjectFunctor)me.nthAtom(0).getFunctor()).getObject();
     Atom done = mem.newAtom(new Functor("done", 1));
     if(pw!=null) {
       pw.print(me.nth(1));
       pw.flush();
     }
     mem.relink(done, 0, me, 2);
     me.nthAtom(1).remove();
     me.remove();
   } catch(Exception e) {e.printStackTrace();}
  :] (Object, String).
```

args of the inline atom

Pure lambda calculus (1)

graph copying

β-reduction

```
H=apply(lambda(A,B), C) :- H=B, A=C.

lambda(A,B)=cp(C,D,L), {+L,$q} :-
    C=lambda(E,F), D=lambda(G,H), A=cp(E,G,L1), B=cp(F,H,L2),
    {{+L1},+L2,sub(S)}, {super(S),$q}.
apply(A,B)=cp(C,D,L), {+L,$q} :-
    C= apply(E,F), D= apply(G,H), A=cp(E,G,L1), B=cp(F,H,L2),
    {+L1,+L2,$q}.
cp(A,B,L1)=cp(C,D,L2), {{+L1,$p},+L2,$q} :-
    A=C, B=D, {{$p},$q}.
cp(A,B,L1)=cp(C,D,L2), {{+L1,$p},$q}, {+L2,top,$r} :-
    C=cp(E,F,L3), D=cp(G,H,L4), {{+L3,+L4,$p},$q},
    A=cp(E,G,L5), B=cp(F,H,L6), {+L5,+L6,top,$r}.
$u=cp(A,B,L), {+L,$q} :- unary($u) | A=$u, B=$u, {$q}.
```

(inspired by F.-R. Sinot, TLCA 2005)

Pure lambda calculus (2)

graph descruction

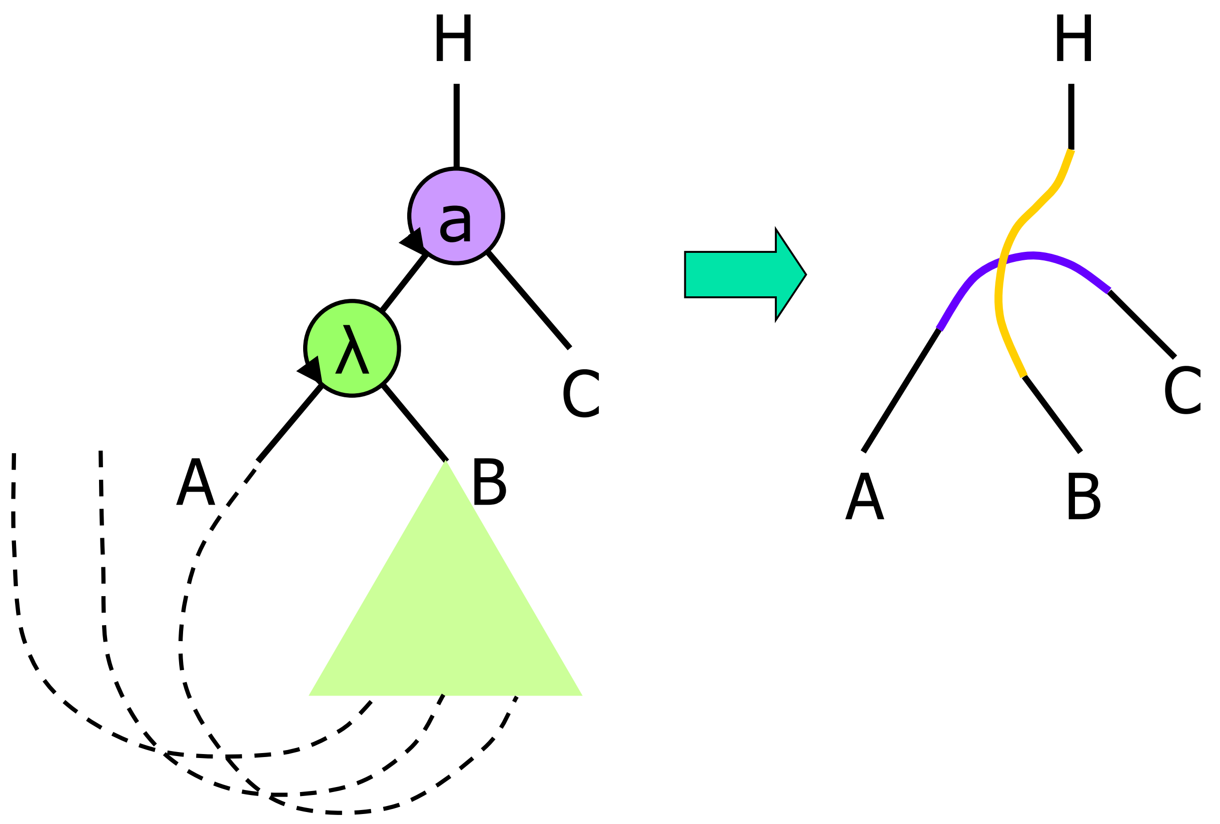
```
lambda(A,B)=rm :- A=rm, B=rm.
apply(A,B)=rm :- A=rm, B=rm.
cp(A,B,L)=rm, {+L,$q} :- A=rm, B=rm, {$q}.
cp(A,B,L)=rm, {{+L,$p},$q} :- A=rm, B=rm, {{$p},$q}.
rm=rm :- .
$u=rm :- unary($u) | .
```

```
{{},$p,sub(S)}, {$q,super(S)} :- {$p,$q}.
A=cp(B,C) :- A=cp(B,C,L), {+L,top}.
{top} :- .
```

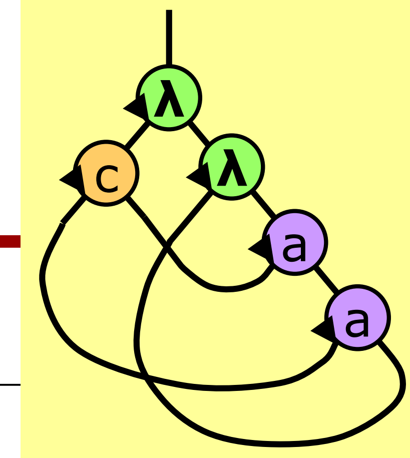
color management

Pure lambda calculus (3)

H=apply(lambda(A,B), C) :- H=B, A=C.



Pure lambda calculus (4)



- Church numeral 2: $\lambda f.\, \lambda x.\, f(f\, x)$

```
lambda(cp(F0,F1),
   lambda(X,apply(F0,apply(F1,X))), Result).
```

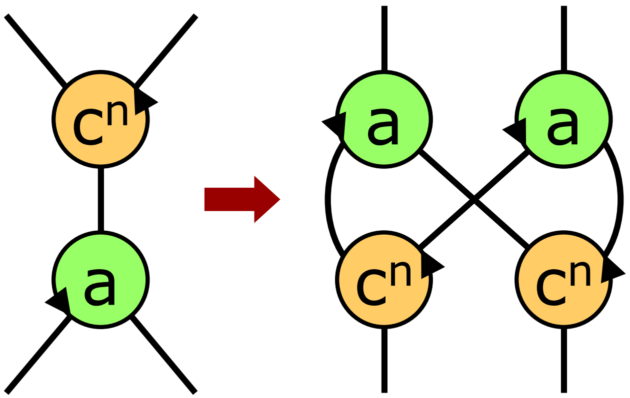
- 3^2 : $(((\lambda m.\lambda n.\, n\, m)\ 3)\ 2)$

```
N=n(2) :- N=lambda(cp(F0,F1),
  lambda(X, apply(F0,apply(F1,X)))).
N=n(3) :- N=lambda(cp(F0,cp(F1,F2)),
  lambda(X, apply(F0,apply(F1,apply(F2,X))))).
res=apply(apply(apply(n(2), n(3)), succ), 0).
H=apply(succ, I) :- int(I) | H=I+1.
```

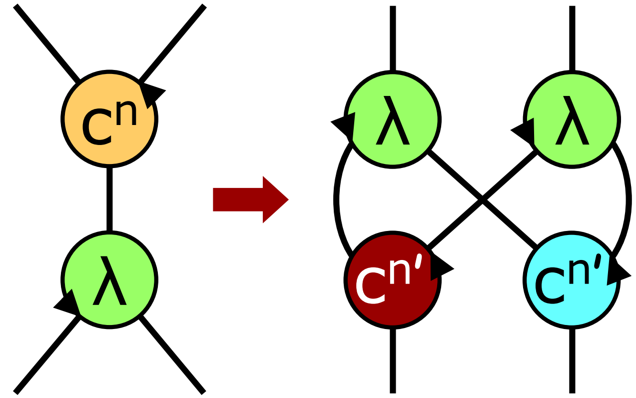
converting to numbers

applying succ and 0 to the Church numeral 3^2

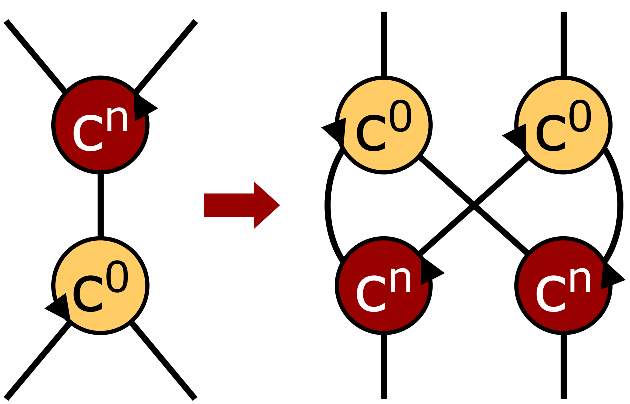
Pure lambda calculus (5)



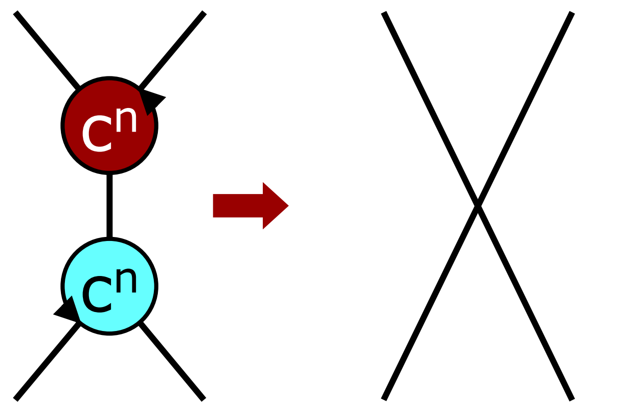
(apply-cp)



(lambda-cp)



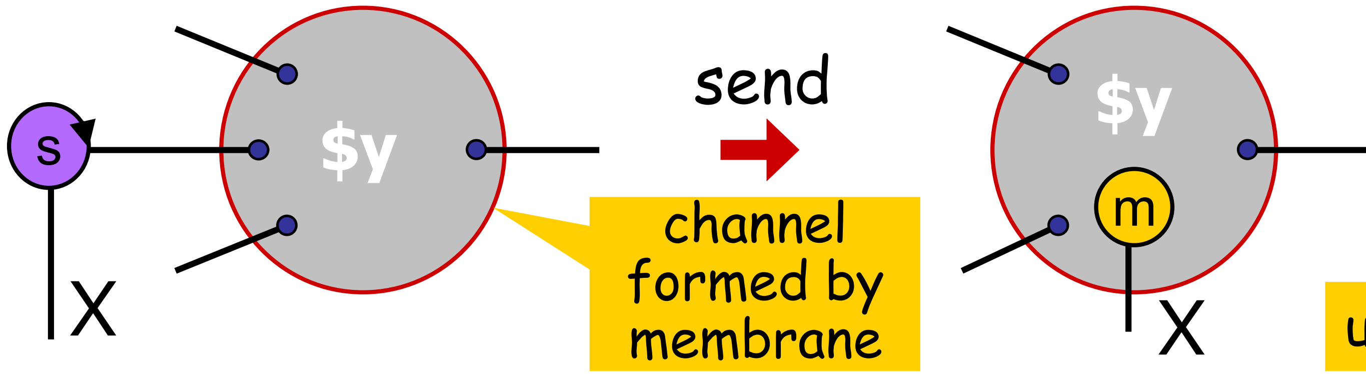
(cp-cp1)



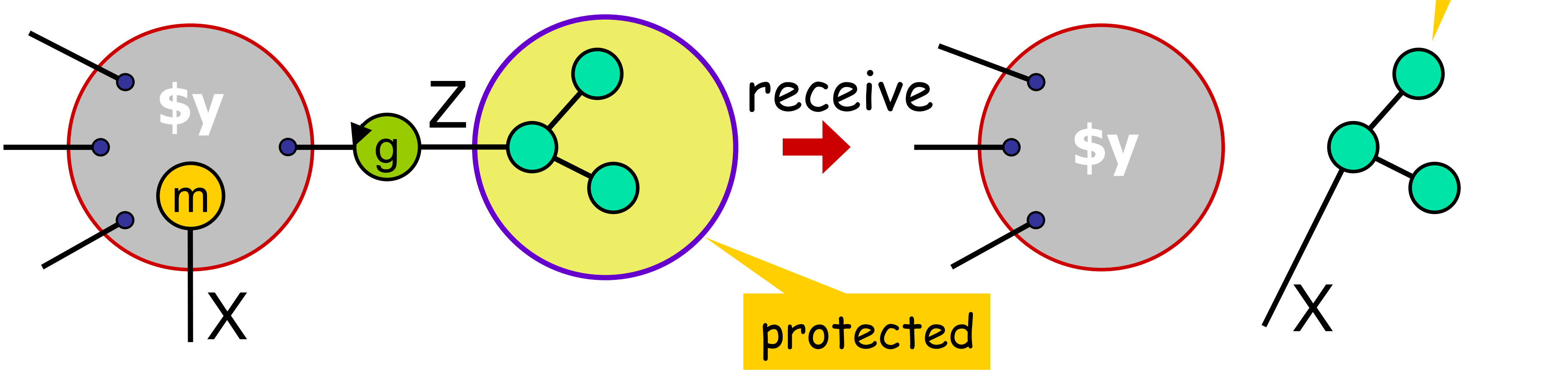
(cp-cp2)

Asynchronous pi-calculus (1)

snd({$y[|*V]},X) :- {$y[|*V], msg(X)}.



get({msg(X),$y[|*V]},Z), {$body[Z|*W]} :-
{$y[|*V]}, $body[X|*W].



Asynchronous pi-calculus (2)

$$(a(z).\,b(y).\,\bar{z}\langle y\rangle) \mid \bar{a}\langle c\rangle \mid \bar{b}\langle d\rangle$$

$$\Rightarrow (b(y).\,\bar{c}\langle y\rangle) \mid \bar{b}\langle d\rangle$$

$$\Rightarrow \bar{c}\langle d\rangle$$

d has been sent to c

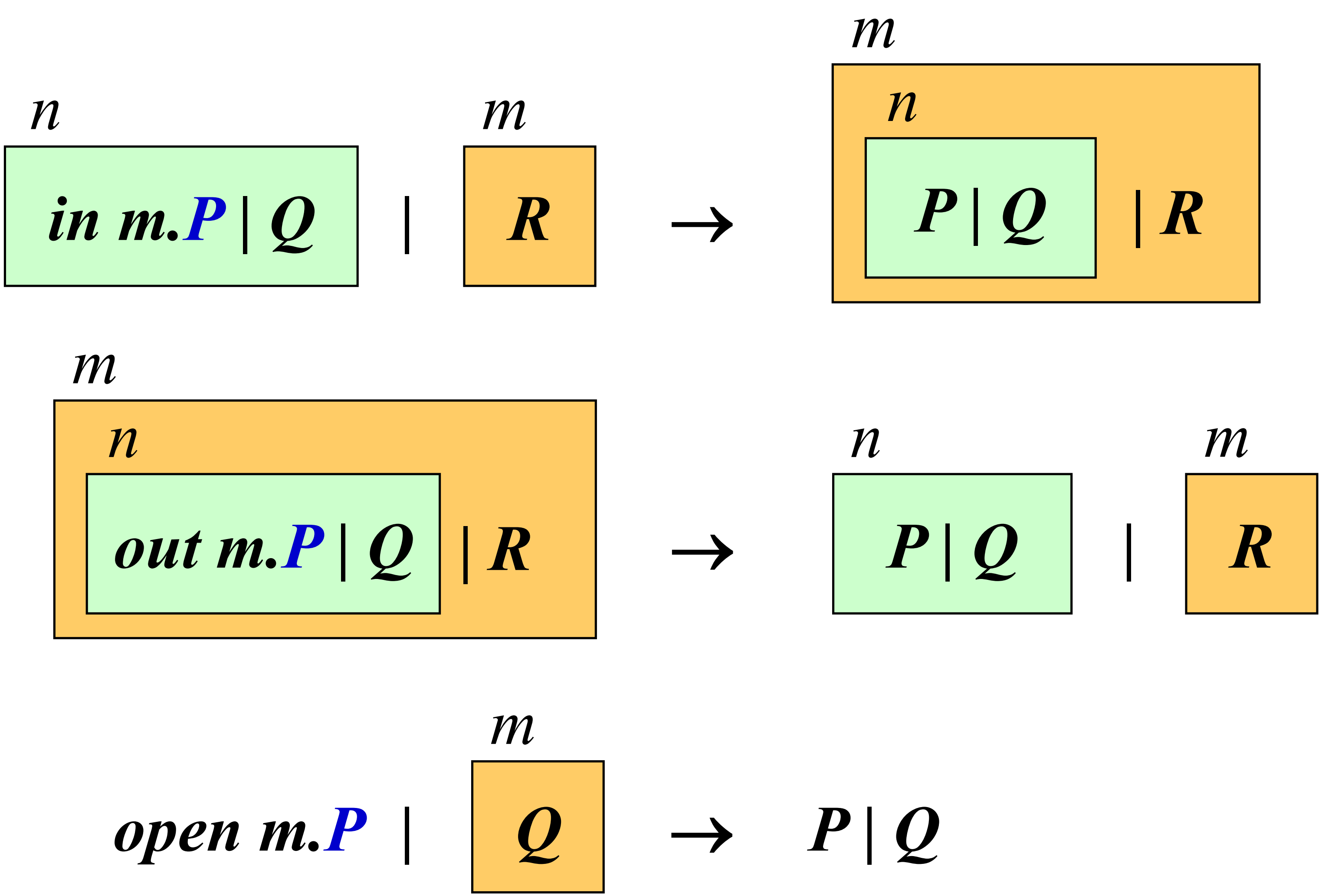
sequencing by membranes

```
get(A0,Z), {get(B0,Y), {snd(Z,Y)}}.
snd(A1,C).  snd(B1,D).
{name(a),+A0,+A1}, {name(b),+B0,+B1},
{name(c),+C}, {name(d),+D}.
```

outstanding message

```
{name(a)}, {name(b)},
{name(c),msg(_78)}, {name(d),'+'(_78)}
```

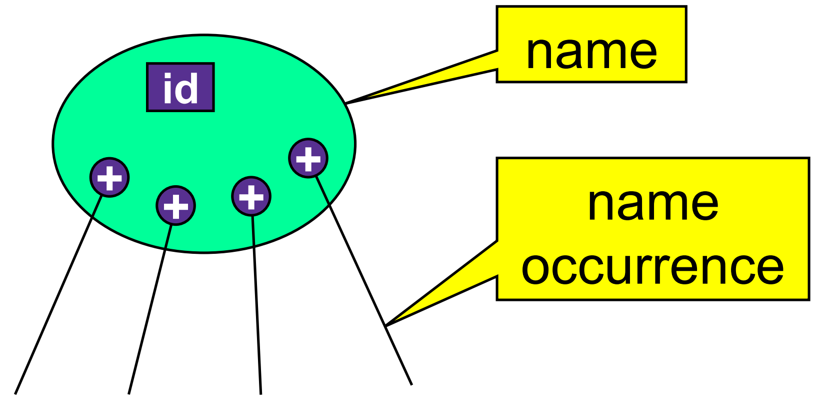
Ambient calculus



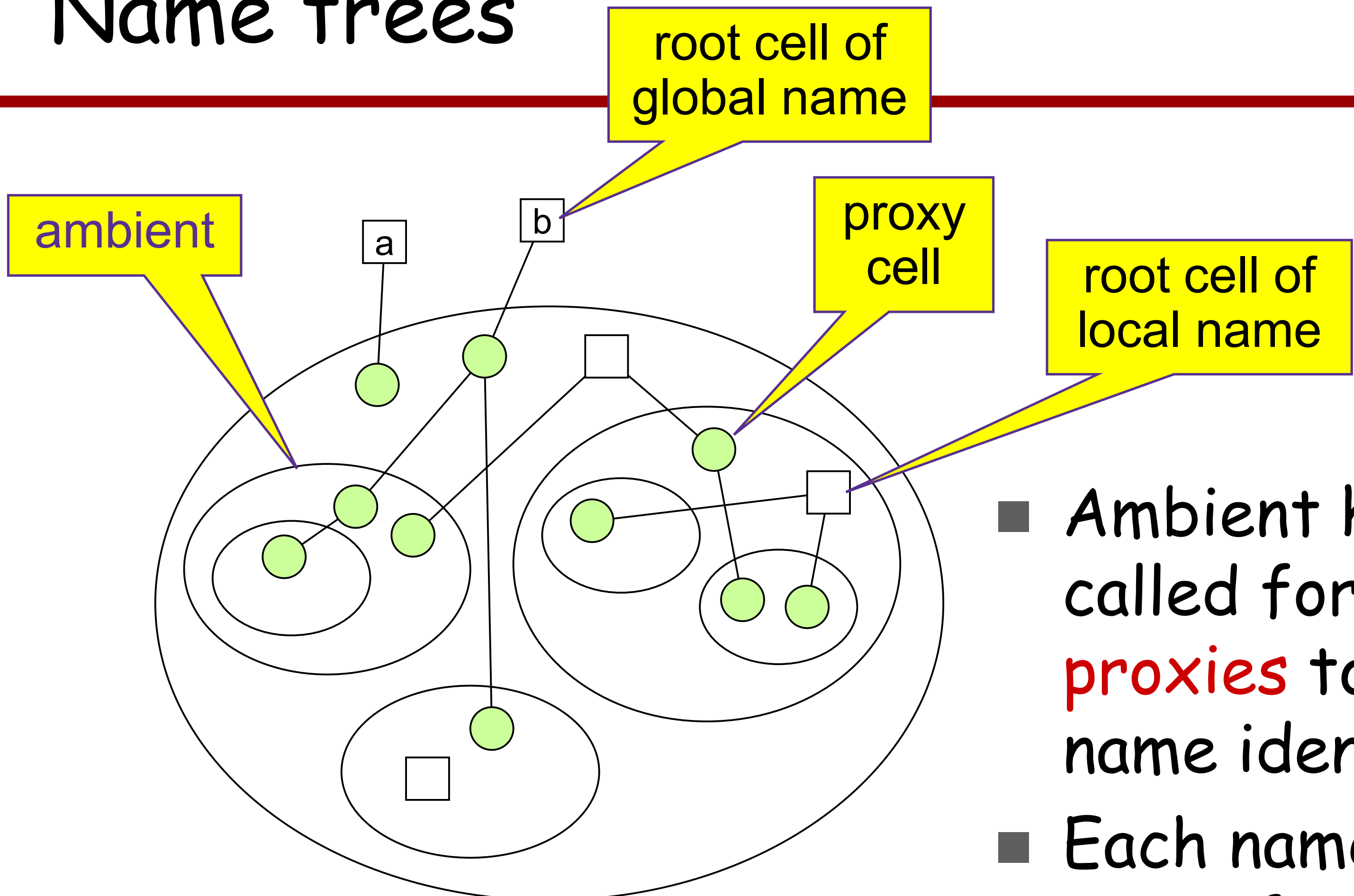
Encoding of names, two alternatives

(Ambient Calculus)		(LMNtal)
Ambient names	$\to$	atom names
Ambient names	$\to$	hierarchical graphs

- We adopt the latter
  - ★ to make reference structures explicit
  - ★ to handle local names
  - ★ to use atom names to encode fixed language constructs (in/out/open) only



Name trees



- Ambient hierarchies called for name proxies to recognize name identity locally
- Each name forms a tree of name proxies
- Normal form of a name tree should correspond to an ambient hierarchy

Encoding ambients, formally

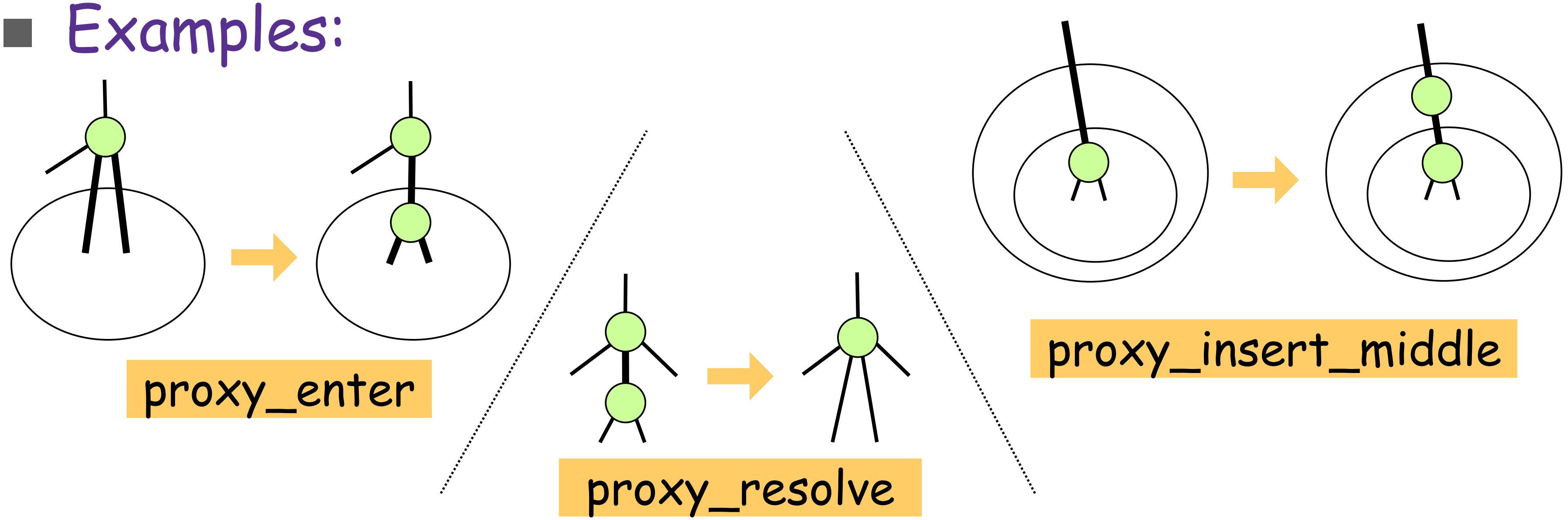
normalization of name trees

hide the name n

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

Name tree normalization

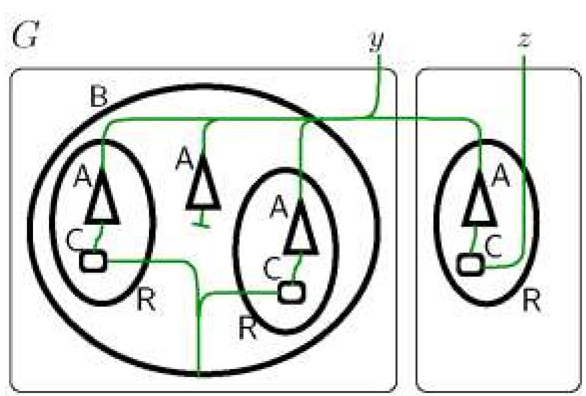
- in/out/open moves an indefinite number of name references across ambient boundaries, violating the normal form conditions temporarily
- Name trees are reformed autonomously and asynchronously
- Examples:

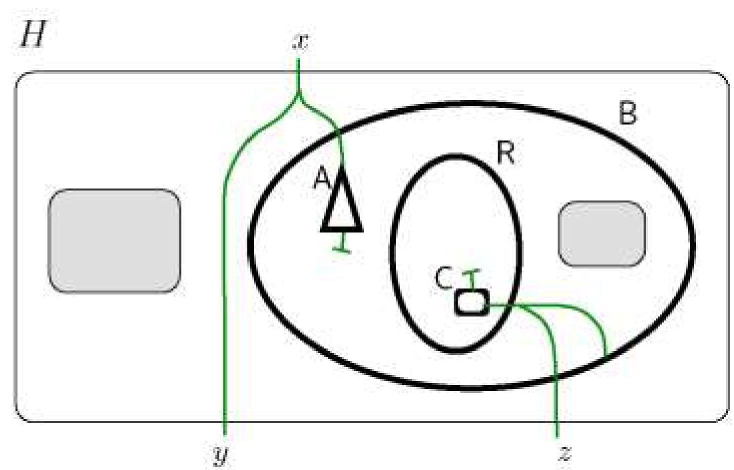


Encoding bigraphs

```
G = graphG :-
  G = map(g([],[]),
    g([{{b,
             {outlet,-Z1,-Z2},
             {r,c(Z1,X1),a(X1,Y1)},
             a(n,Y2),
             {r,c(Z2,X2),a(X2,Y3)}
          }},
       {{r,c(Z,X),a(X,Y4)}}],
      [{-Y1,-Y2,-Y3,-Y4},{-Z}])).
```

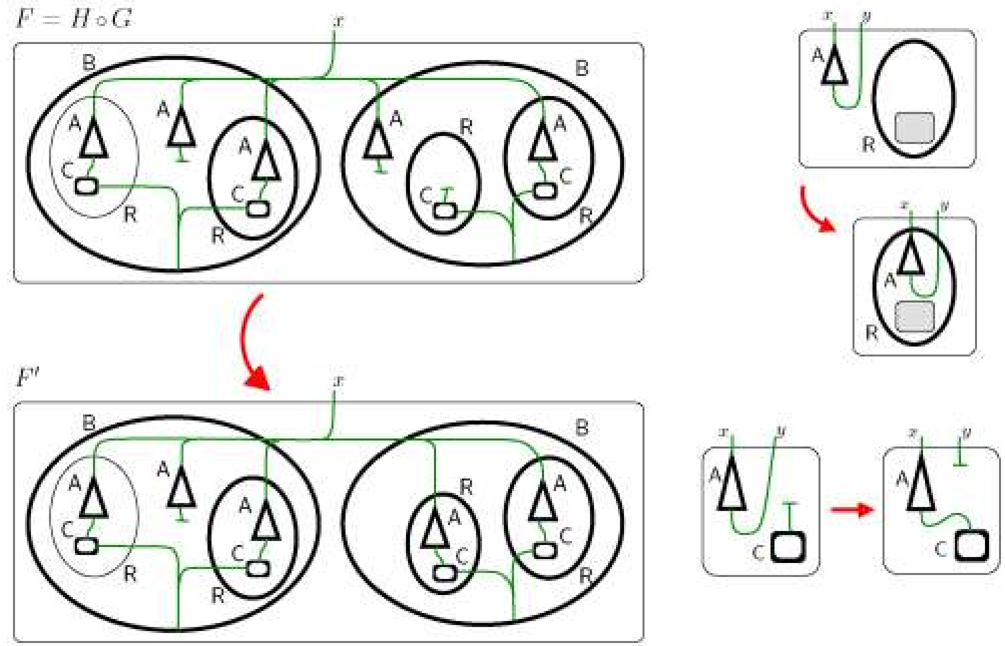
```
G = graphH :-
  G = map(g([P1,P2],[Y,Z]),
    g([{{{b,
             {outlet,-Z1}+Z,
             a(n,X1),
             {r,c(Z1,n)}
           }+P2
          } +P1
       }],
      [{-X1}+Y])).
```





Encoding bigraphs

```
compose@@
M = compose(
        map(g(Ps0,Ns0),g(Ps1,Ns1)),
        map(g(Ps2,Ns2),g(Ps3,Ns3))) :-
   M = map(g(Ps0,Ns0),g(Ps3,Ns3)),
   connect(Ps1,Ps2),
   connect(Ns1,Ns2).

connect([], []) :- .
connect([X|Xs], [Y|Ys]) :-
           cn(X,Y), connect(Xs,Ys).

{ system_ruleset.
  union@@    {$p} + cn({$q}) :- {$p,$q}.
  import@@   {$p,+X}, cn(X,Y), {$q[Y|*Z]} :-
               {{$p,+X}, cn(X,Y), $q[Y|*Z]}.
  enter@@    a(n,X), {r,$rr} :- {r, a(n,X), $rr}.
  connect@@ a(n,X), c(Z,n) :- a(U,X), c(Z,U).  }
```



Robin Milner, Pervasive process calculus, Proc. PA'05 (Bertinoro, Forli, Italy, August 2005), pp.180-183

Membranes

- Membranes provide
  - data structures (multisets, records, ...)
  - hyperlinks
  - agents / test tubes (data + rulesets)
  - control structure (sequencing, guard)
  - modules (encapsulated rulesets)
    - ordinary modules **{ module(*name*), . . . }**
    - system rulesets **{ system_ruleset, . . . }**

Ongoing and Future Work

- Language
  - Infinite multiplicity
  - Nondeterministic execution (search)
  - Hyperlinks
- Foundations
  - Type systems
  - Verification
  - Reversibility
- Implementation
  - Optimizing complation of sequential core
  - Integration with static analysis
  - Parallel, distributed, embedded implementation
  - Interoperability

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