

Constraint-Based Concurrency and Beyond

Kazunori Ueda, Waseda Univ.

August 2005

Constraint-Based Concurrency

(Also known as concurrent logic/constraint programming)

- Concurrency formalism & language based on **single-assignment channels** and **constructors**
cf. process algebra (name-based concurrency)
- Features
 - ★ asynchronous communication
 - ★ channel mobility
 - ★ poliadicity and data structuring
 - ★ nonstrictness (computing with partial info)
- Born and used as languages (early 1980's);
then recognized and studied as formalisms

Single-Assignment Channels

- Also known as **logical variables**
- Can be written at most once
 - ★ by **telling** the value of the channel (unification)
 - e.g., $\text{tell } S = [\text{read}(X) | S']$
- Reading is non-destructive
 - ★ by **asking** if a constraint is entailed (term matching)
 - e.g., $\text{ask } \exists A \exists S' (S = [A | S'])$
 - ★ covers input and match in the π -calculus

Guarded Horn Clauses (1985)

ask, choice,
reduction &
hiding

(program)	$P ::= \text{set of } R's$
(rule)	$R ::= !\forall(A . B)$
(body)	$B ::= \text{multiset of } G's$
(goal)	$G ::= T_1 = T_2 \mid A$
(non-unif. atom)	$A ::= p(T_1, \dots, T_n), \quad p \neq '='$
(term)	$T ::= (\text{as in first-order logic})$

tell

parallel
composition

Reduction Semantics

■ Concurrency

$$\frac{\langle B_1, C, P \rangle \rightarrow \langle B'_1, C', P \rangle}{\langle B_1 \cup B_2, C, P \rangle \rightarrow \langle B'_1 \cup B_2, C', P \rangle}$$

■ Tell

send t_2 through t_1
/ fuse t_1 with t_2

$$\frac{}{\langle \{t_1 = t_2\}, C, P \rangle \rightarrow \langle \phi, C \cup \{t_1 = t_2\}, P \rangle}$$

unguarded constraint is made observable

Reduction Semantics (cont'd)

■ Ask + Reduction

ask done and constraints were received by h 's args

$$\begin{aligned} &\langle \{b\}, C, P \cup \{! \forall (h.B)\} \rangle \\ &\rightarrow \langle B, C \cup \{b = h\}, P \cup \{! \forall (h.B)\} \rangle \end{aligned}$$

$$\left(\begin{array}{l} \text{if } E \models \forall (C \Rightarrow \exists \text{vars}(h)(b = h)) \\ \text{and } \text{vars}(h, B) \cap \text{vars}(b, C) = \emptyset \end{array} \right)$$

equality theory
over finite terms

h matches b under C

Communication in CBC

- In CBC,
 - ★ **tell** subsumes two operations
 - output e.g., $X=3$, $X=[\text{push}(5)|X']$
 - fusion (of two channel names) e.g., $X=Y$
 - ★ **ask** subsumes two operations
 - input (synchronization and value passing)
 - match (checking of values)
- However, match in **moded** CBC doesn't allow the checking of channel equality (cf. $L\pi$)

Channels in CBC and NBC (PA)

- CBC and NBC get closer with **type systems**:
 - ★ **mode** (= polarity type) system for CBC
 - ★ **linear types** for the π -calculus
- Both guarantees that exactly one process holds a write capability and use it once
 - ★ hence they leave no sharp difference in non-destructive and destructive read,
 - ★ except that CBC still allows channel fusion and multicasting

Channels in CBC Are Local Names

- Fallacy: constraint store is global, shared, single-assignment memory
- Channels are all created as fresh local names that cannot be forged by the third party
- A new channel can be exported and imported only by using an existing channel
 - ★ e.g.,
$$\begin{aligned} & !\forall(p([create(S)|X']) \cdot server(S), p(X')) \\ & A=[create(X)|A'], client(X), p(A') \end{aligned}$$

LMNtal (pronounce: "elemental")

— From Concurrent Constraint Programming
to Concurrent Hierarchical Graph Rewriting

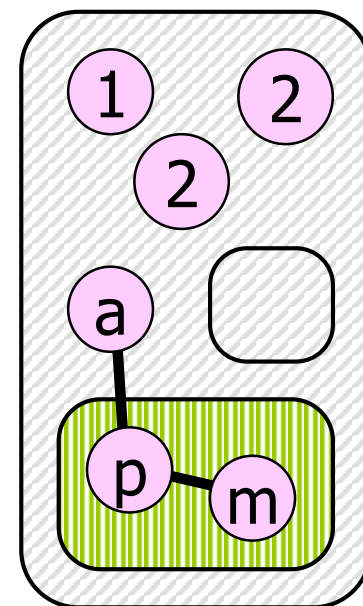
\mathcal{L} = logical links

\mathcal{M} = multisets/membranes

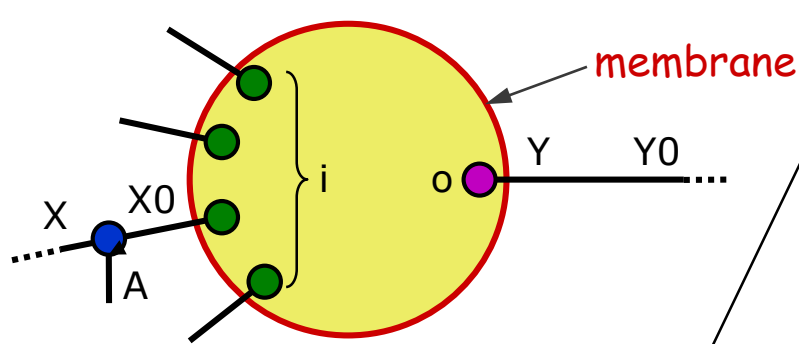
\mathcal{N} = nested nodes

ta = transformation

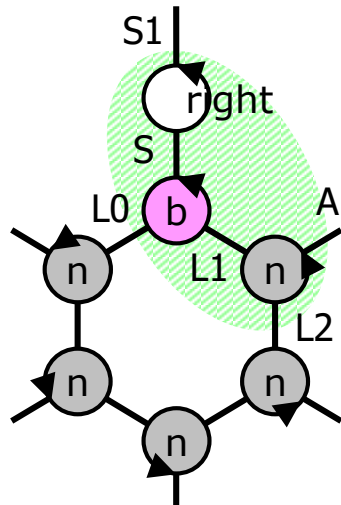
\mathcal{L} = language



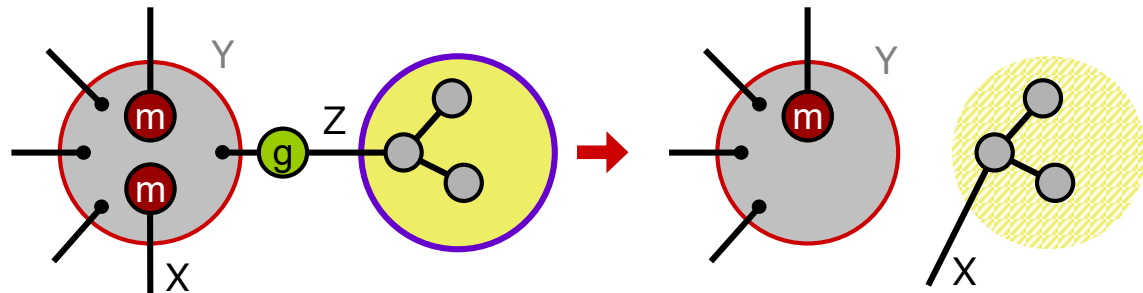
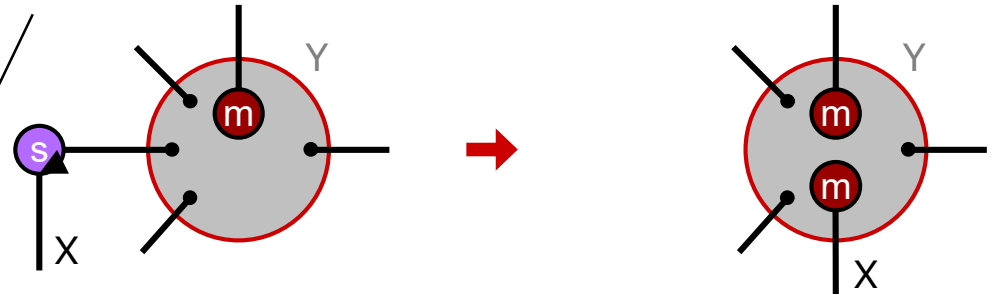
Diagrammatic representation of computation and its manipulation



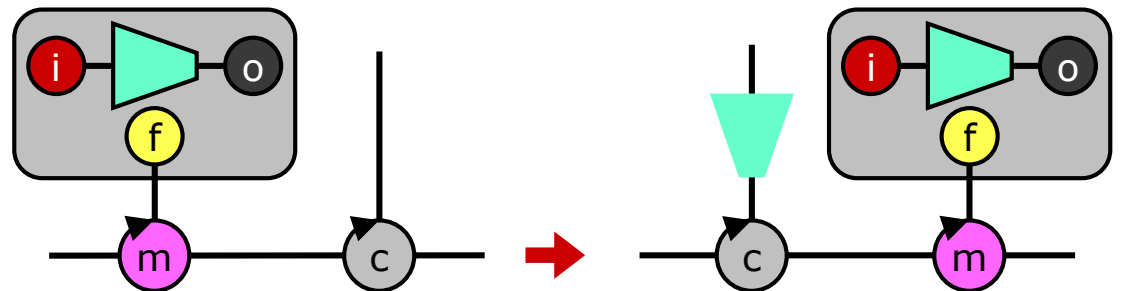
n -to-1 comm.



cyclic structures

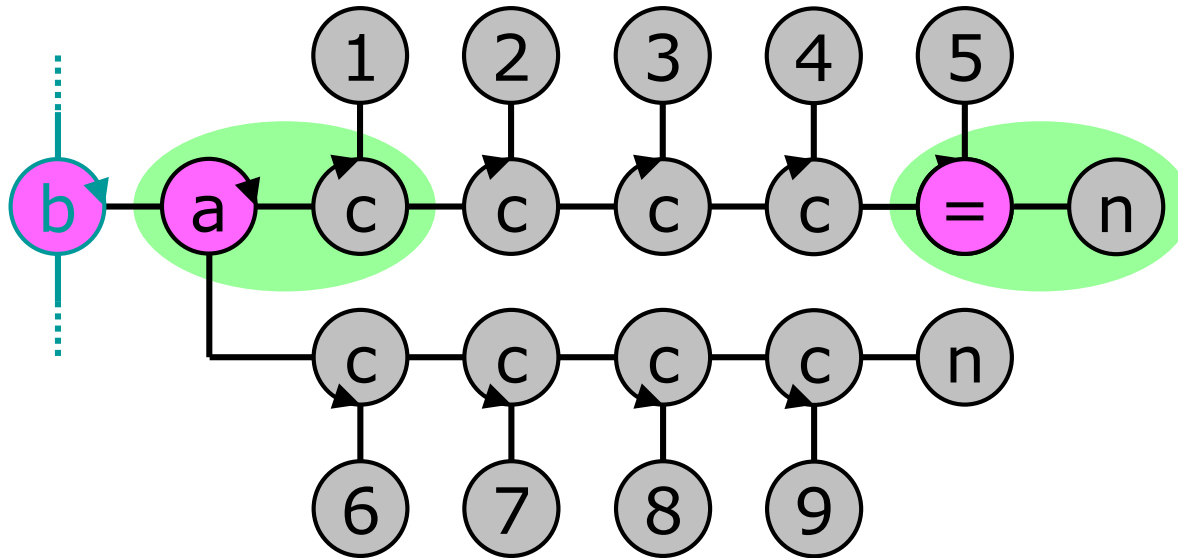


asynchronous π -calculus

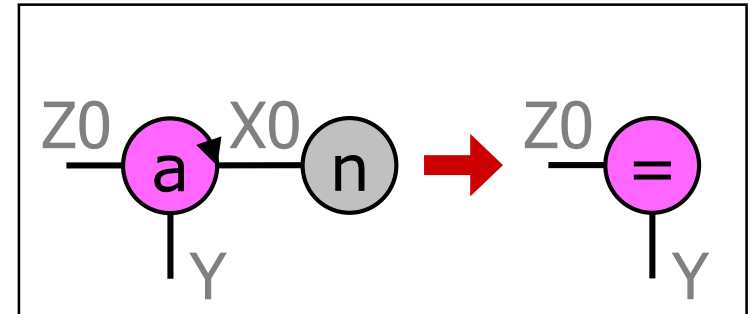
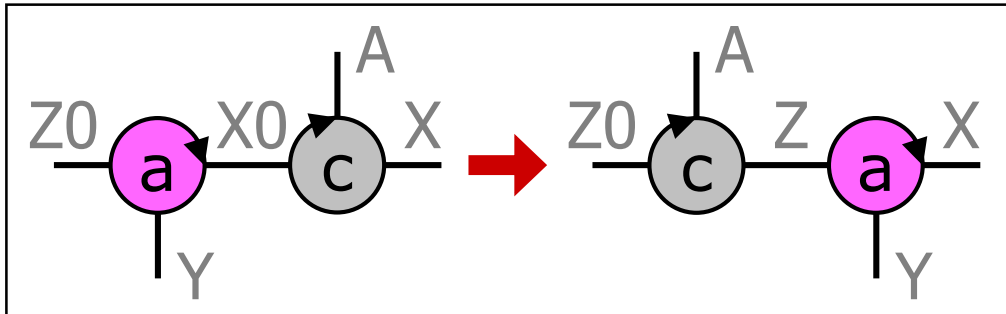


map function

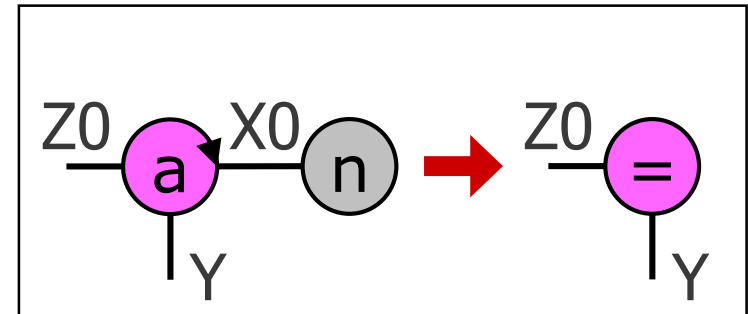
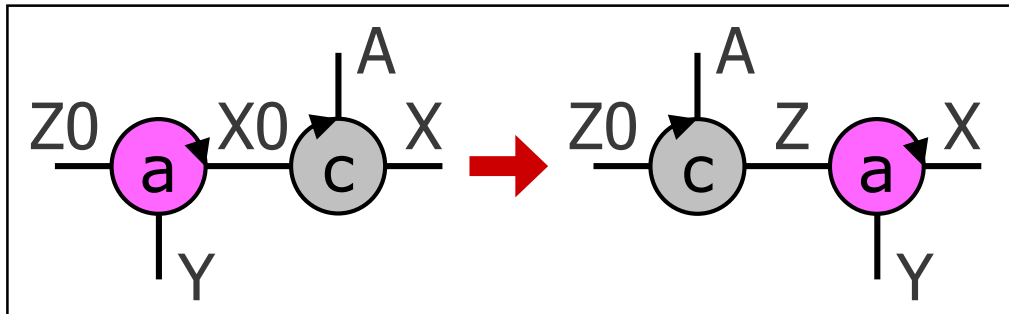
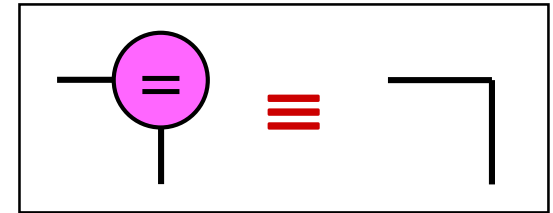
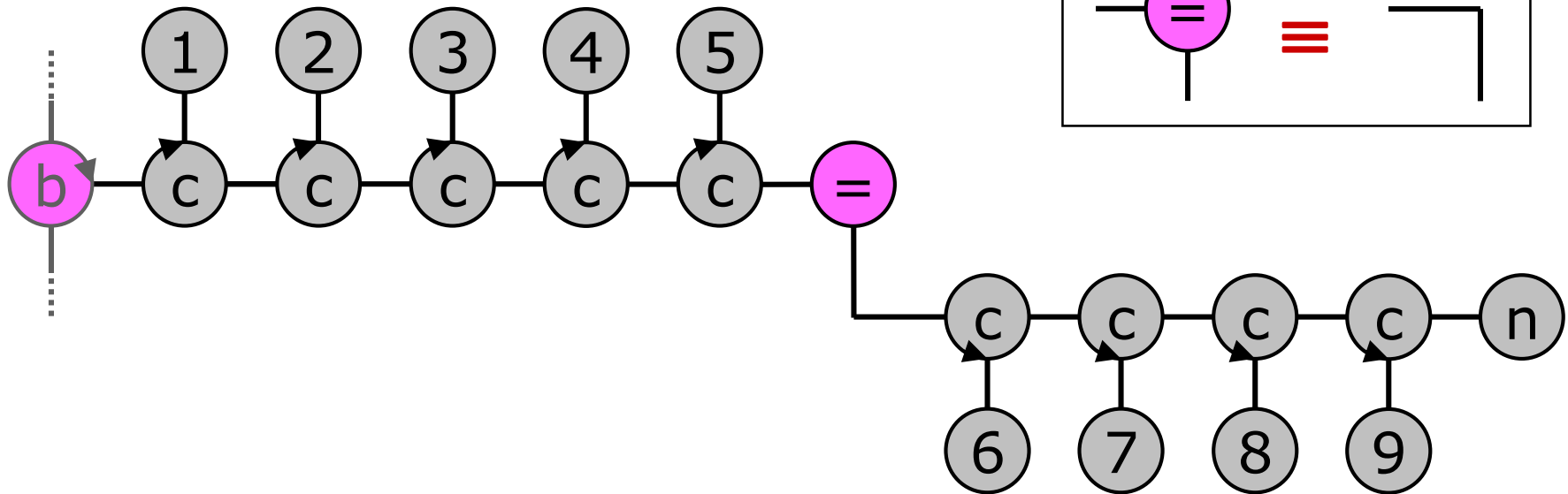
Example: append



a : append
c : cons
n : nil



Example: append



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

- No distinction between append and c(ons)

Design goals of LMNtal

- A "Turing Machine" for universal computing environments (from wide-area to nanoscale)
- Serve also as a practical programming language
 - ★ Implementation available!
- Unifying
 - ★ e.g., processes = messages = data
- Simple and versatile
 - ★ Computation is manipulation of graphs
 - ★ Membranes express multisets and locality
 - ★ Allows programming by self-organization

Fullerene (C₆₀)

```

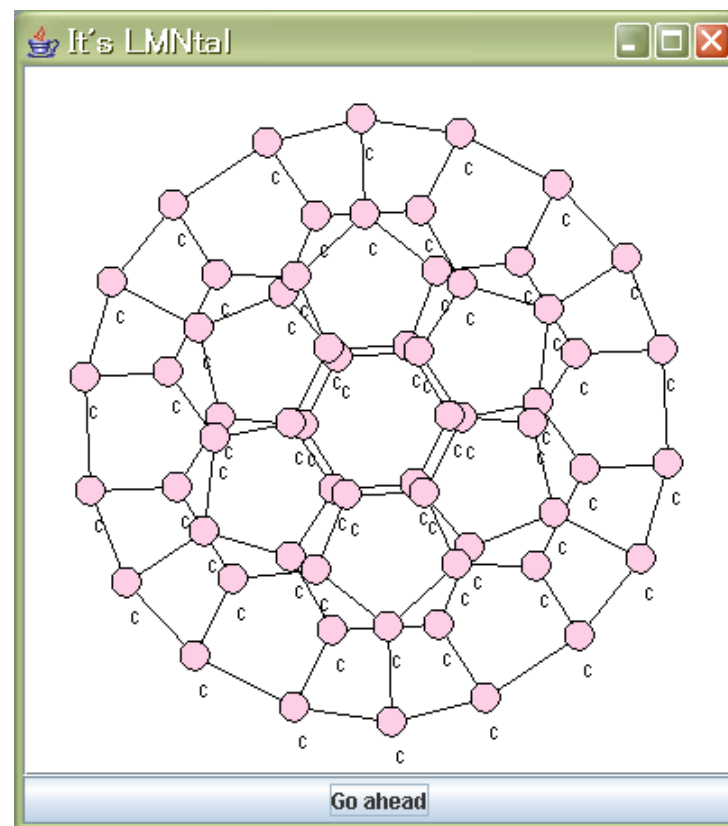
C1=c(L11,c(L12,c(L13,c(L14,c(L15,C1))))).
C2=c(L21,c(L22,c(L23,c(L24,c(L25,C2))))).
C3=c(L31,c(L32,c(L33,c(L34,c(L35,C3))))).
C4=c(L41,c(L42,c(L43,c(L44,c(L45,C4))))).
C5=c(L51,c(L52,c(L53,c(L54,c(L55,C5))))).
C6=c(L61,c(L62,c(L63,c(L64,c(L65,C6))))).
C7=c(L71,c(L72,c(L73,c(L74,c(L75,C7))))).
C8=c(L81,c(L82,c(L83,c(L84,c(L85,C8))))).
C9=c(L91,c(L92,c(L93,c(L94,c(L95,C9))))).
CA=c(LA1,c(LA2,c(LA3,c(LA4,c(LA5,CA))))).
CB=c(LB1,c(LB2,c(LB3,c(LB4,c(LB5,CB))))).
CC=c(LC1,c(LC2,c(LC3,c(LC4,c(LC5,CC))))).

```

```

L11=LB1,L12=L25,L13=L63,L14=LA4,L15=L52.
L21=LB2,L22=L35,L23=L73,L24=L64
L31=LB3,L32=L45,L33=L83,L34=L74
L41=LB4,L42=L55,L43=L93,L44=L84
L51=LB5,    L53=LA3,L54=L94
L61=LC1,L62=LA5
L71=LC2,L72=L65
L81=LC3,L82=L75
L91=LC4,L92=L85
LA1=LC5,LA2=L95

```



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 \mathcal{LMNtal}

Syntax of \mathcal{LMNtal} processes

- $P ::= \mathbf{0}$ (null)
 - | $p(X_1, \dots, X_m)$ ($m \geq 0$) (atom)
 - | P, P (molecule)
 - | $\{P\}$ (cell)
 - | $T :- T$ (rule)
- Not in Flat \mathcal{LMNtal}
- **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 \mathcal{LMNtal} process templates

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

Not in
Flat LMNtal

Structural congruence (\equiv)

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

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

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

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

$$(E5) \quad P \equiv P' \Rightarrow P, Q \equiv P', Q$$

$$(E6) \quad P \equiv P' \Rightarrow \{P\} \equiv \{P'\}$$

$$(E7) \quad X = X \equiv \mathbf{0}$$

$$(E8) \quad X = Y \equiv Y = X$$

$$(E9) \quad X = Y, P \equiv P[Y/X]$$

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

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

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

Reduction semantics

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

$$(R2) \quad \frac{P \rightarrow P'}{\{P\} \rightarrow \{P'\}}$$

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

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

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

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

X and Y are free links of P

$$(R6) \quad T\theta, (T:-U) \rightarrow U\theta, (T:-U)$$

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

Implementation Available !

- A single .jar (Java archive) file running on JDK 1.4 or higher
 - ★ <http://www.ueda.info.waseda.ac.jp/lmntal/>
- Features:
 - ★ Arithmetics (int, float)
 - ★ Modules
 - ★ Foreign-language interface (to Java)
 - ★ Visualizer
 - ★ Read-Eval-Print Loop
 - ★ Optimizer
 - ★ Redex/Rule selection strategies
 - ★ Libraries