Programming and Modelchecking with Hypergraphs

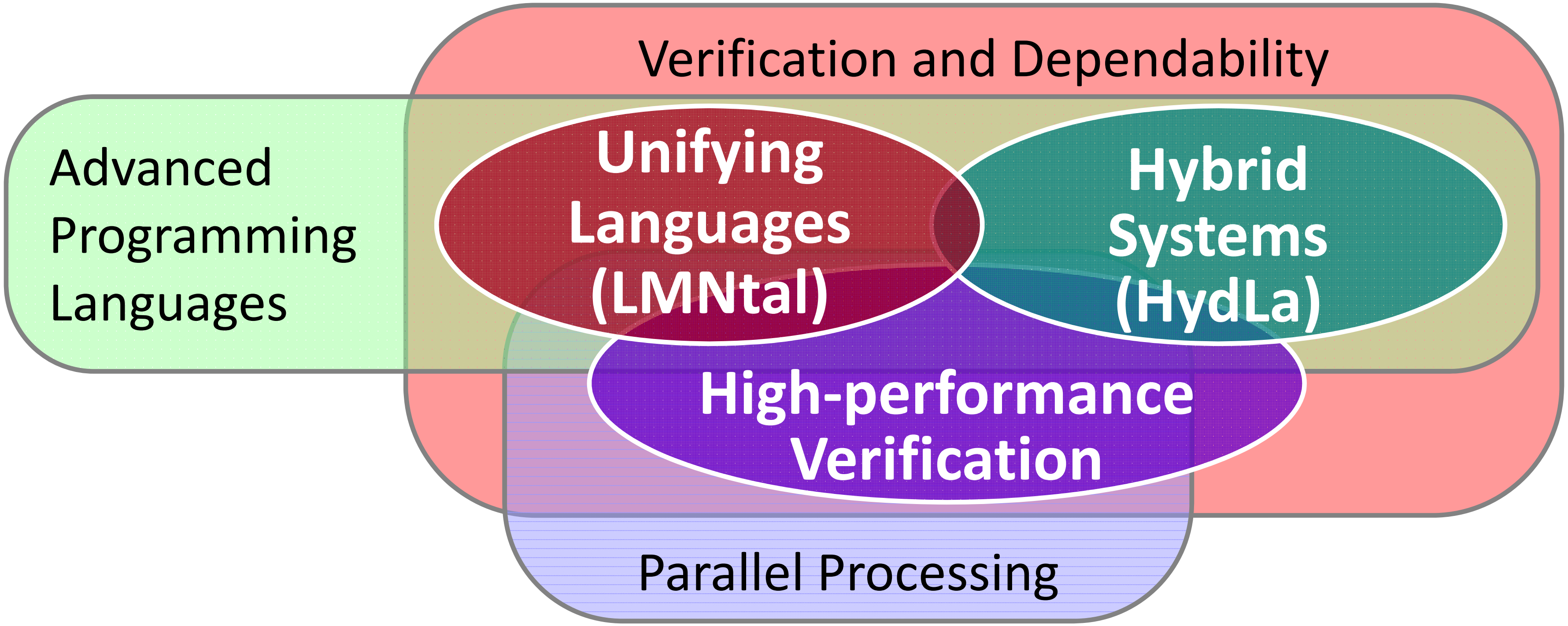
Kazunori Ueda

(special thanks to many students of mine)

Waseda University and
National Institute for Informatics

July 2013

Research Groups and Their Relationship



- ✓ Three interrelated groups
- ✓ Three cross-cutting concerns

Talk Outline

Demo-driven introduction to:

- LMNtal (pronounce: “elemental”), a programming and modeling language based on (a class of) hierarchical (hyper)graph rewriting
  - what is it about ?
  - what can it do ?
- State-space search and model checking with LMNtal
  - what are the strengths of the LMNtal model checker?
  - how does an IDE plays an important role?

LMNtal: a Unifying Language

- **Project LMNtal** (pronounce "elemental")
  - A computational model + language + system
  - Started in 2002, now working on the 4G implementation
  - >100,000 LOC involving many people over the years
  - Focused on verification (parallel state-space search, LTL model checking) since 2007
  - Provides LaViT, an IDE with visualizers
- Ready to use; very low entry barrier

  http://www.ueda.info.waseda.ac.jp/lmntal/

  - partly open-source from Google Code

** K. Ueda, *Theoretical Computer Science* **410**, 2009

* K. Ueda, *Proc. ICTAC 2009*, LNCS 5684

Programming vs. Modeling

- A **programming language** is (usually) to describe algorithms and *execute* them, often *interacting with the real world*.
  - Algorithms are usually deterministic.
- A **modeling language** is to describe models (of its target domain) and *simulate* or *verify* them in order to *reason about the real world*.
  - A model changes its state over time and forms a state space (= the set of all possible states).
  - Models are usually nondeterministic.
  - **Model checking** explores the state-space to see if the model meets its specification.

Programming vs. Modeling

- A **programming language** and a **modeling language** have different objectives and applications, and our challenge is to have a language and its implementation that handle programming and modeling in a single, unified framework.

LMNtal (pronounce: “elemental”)

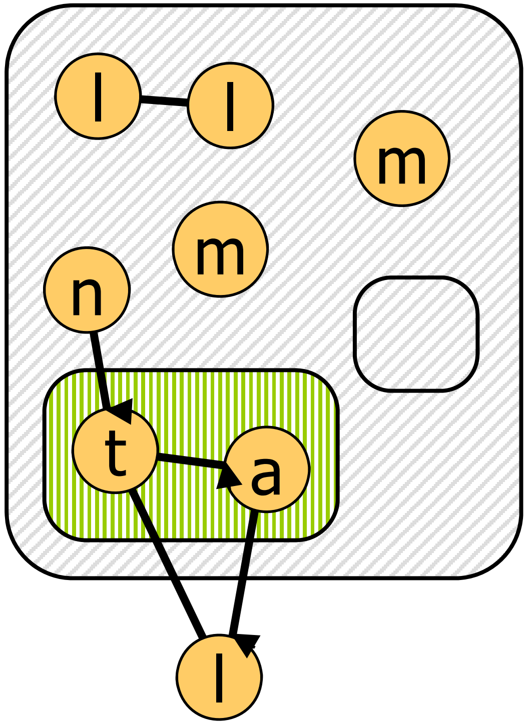
$\mathcal{L}$ = links

$\mathcal{M}$ = multisets/membranes

$\mathcal{N}$ = nested nodes

ta = transformation

ℓ = language

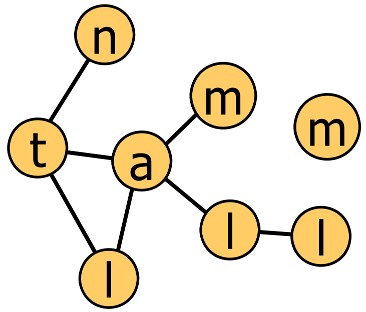


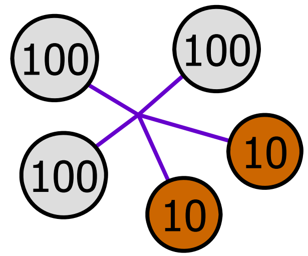
A language based on hierarchical graph rewriting

An example hierarchical graph. Its intuitive meaning should be almost clear, but you’ll find that it’s not (yet) totally clear.

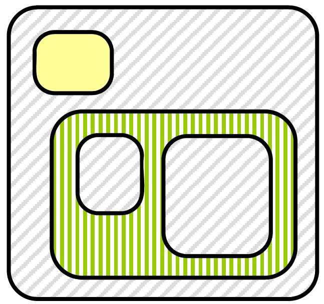
Hierarchical (Hyper)Graphs: Motivations

- Structures found in organization (of any kind) and human knowledge have one or both of the following:
  - **connectivity**
    - network, graphs, human relationships, ...





  - **hierarchy**
    - companies, addresses, domain names, ...



- Why not have a concise programming language that allows us to represent and manipulate them simultaneously and in a direct manner?

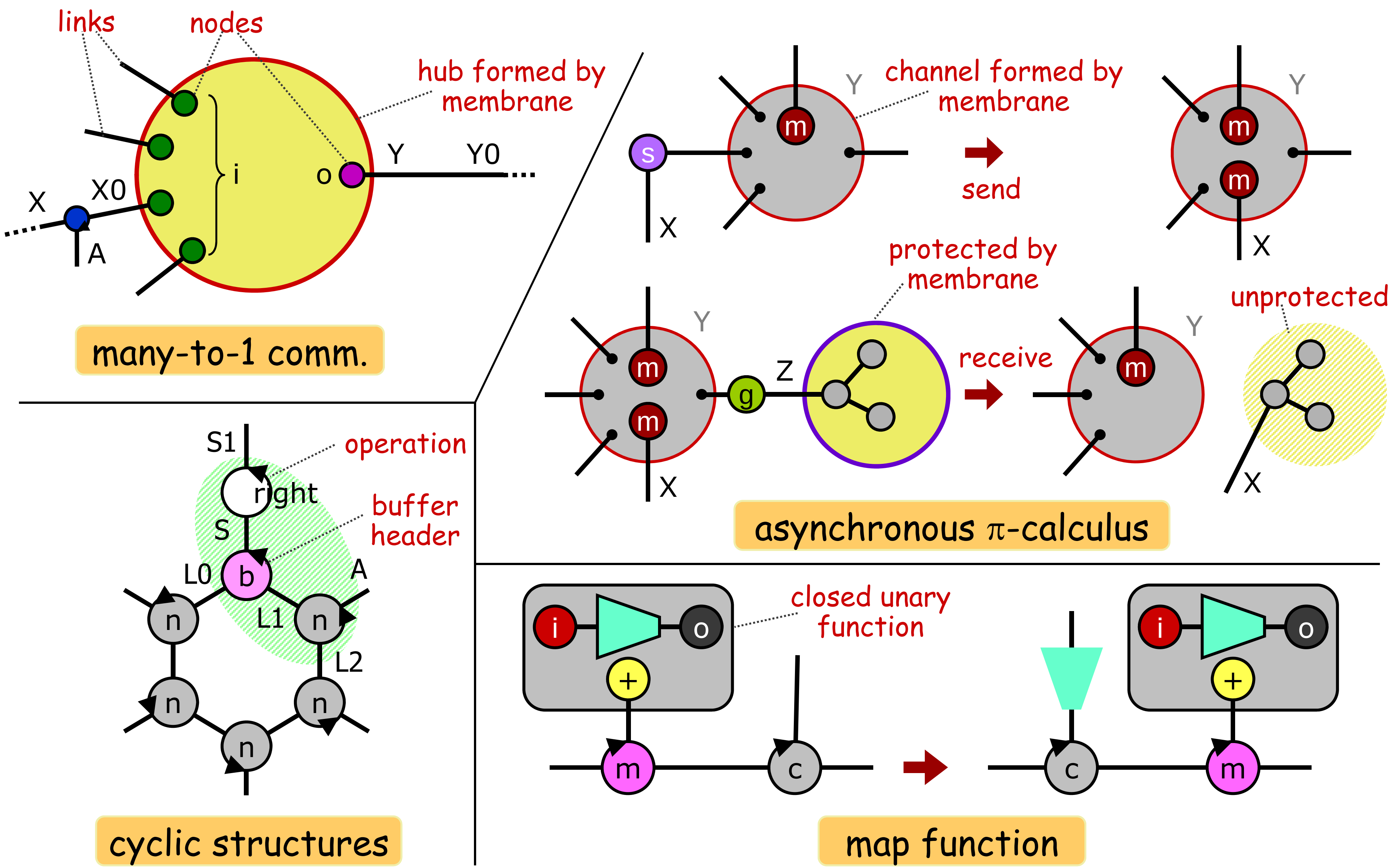
LMNtal: What and Why

- Rule-based concurrent **language** for expressing & rewriting **connectivity** and **hierarchy**
- Substrate **model** of various calculi (λ, π, ambient, etc.) *
- Computation is manipulation of **diagrams**
  - **Links** express 1-to-1 **connectivity**
  - **Hyperlinks** express multipoint **connectivity** **
  - **Membranes** express **hierarchy**, **locality** and **first-class multisets**
  - Allows **programming with sets and graphs** and **programming by self-organization**
  - Well-defined notion of **atomic actions**

* K. Ueda, *Proc. RTA 2008*, LNCS 5117

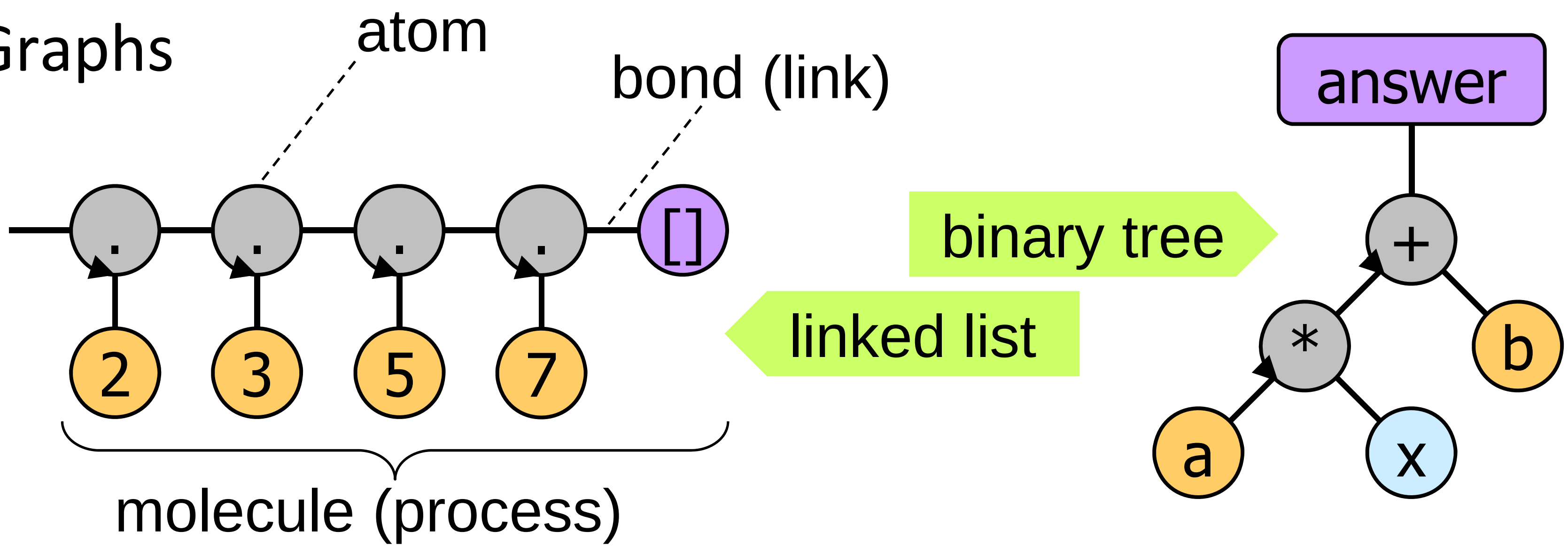
** K. Ueda and S. Ogawa, *Künstliche Intelligenz* **26**, 2012

LMNtal allows us to represent computation in terms of hierarchical graph rewriting

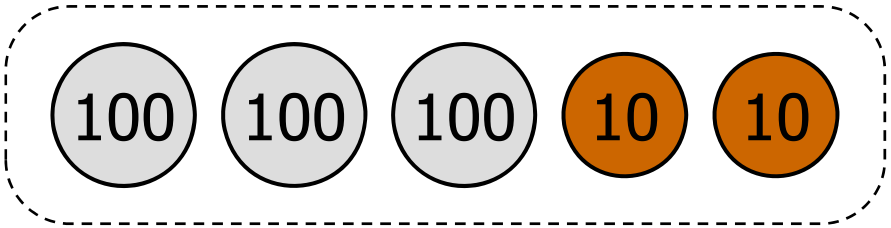


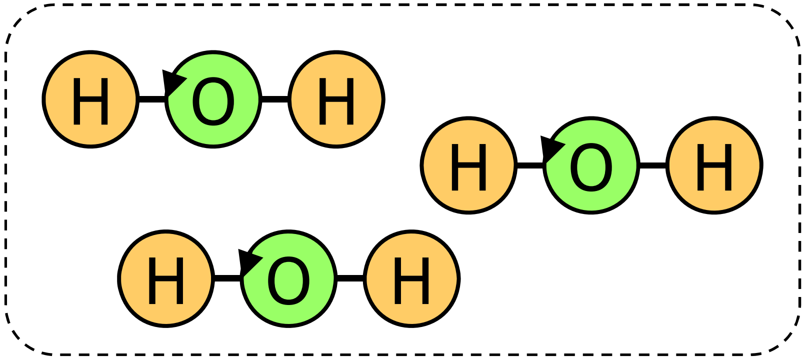
Graphs and Multisets

- Graphs



- Multisets

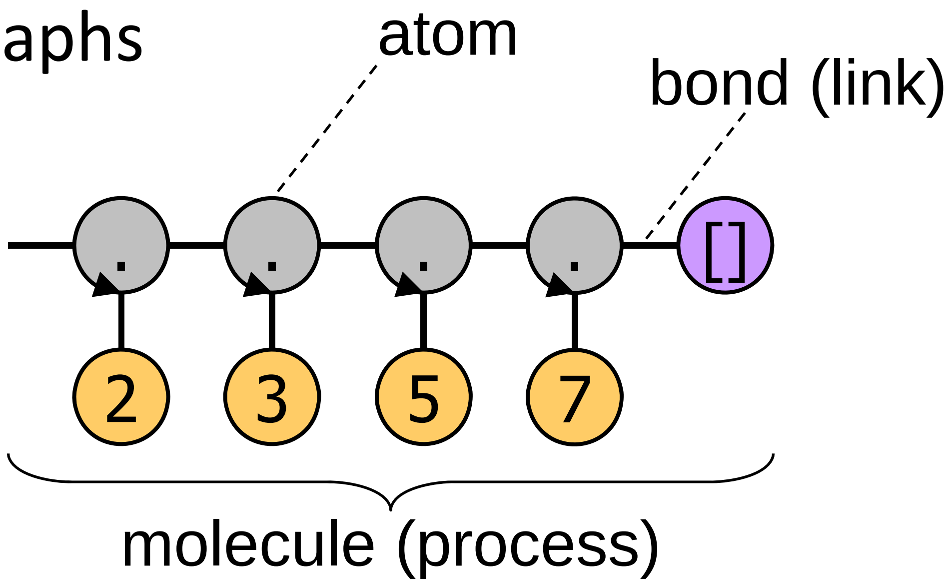


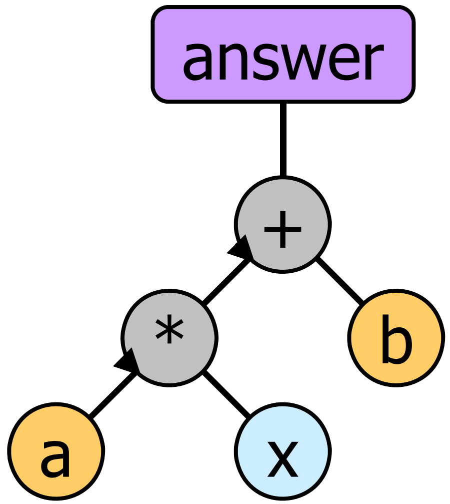


  - ... is a kind of graph.

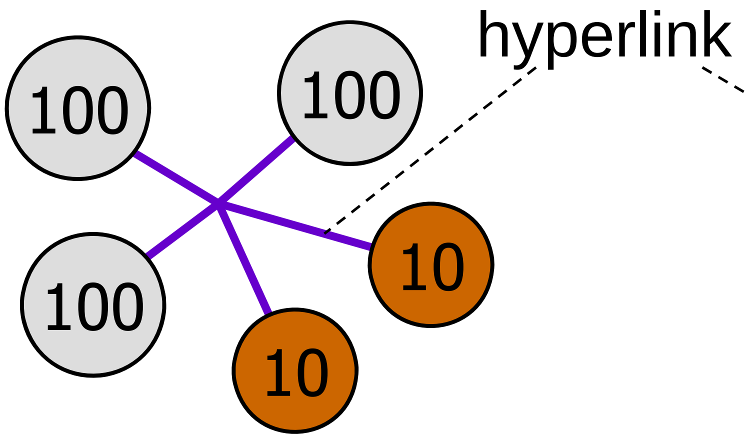
Graphs and Hypergraphs

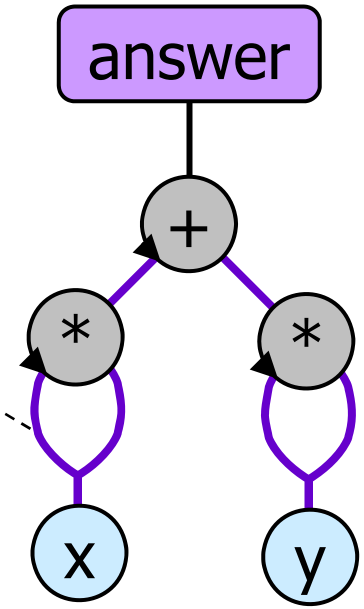
- Graphs





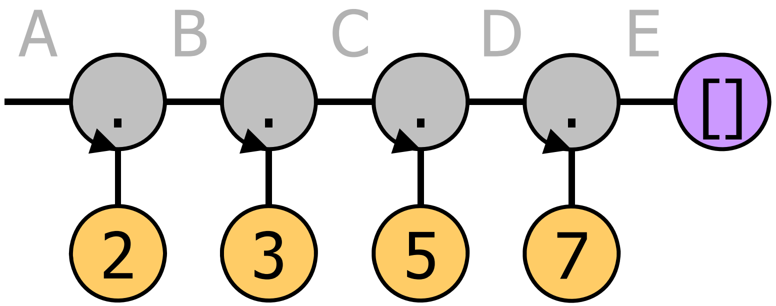
- Hypergraphs





Text Representation

- Graphs



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

– or –

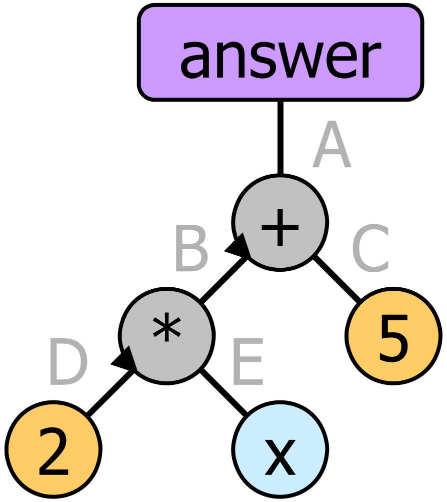
A= '.'(2,'.'(3,'.'(5,'.'(7,'[]'))))

– or –

A=[2|[3|[5|[7|[]]]]]

– or –

A=[2,3,5,7]



answer(A),
'+'(B,C,A), '*'(D,E,B),
2(D), x(E), 5(C)

– or –

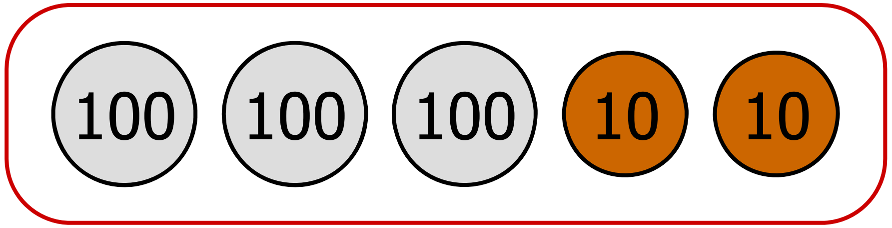
answer('+'('*'(2,x),5))

– or –

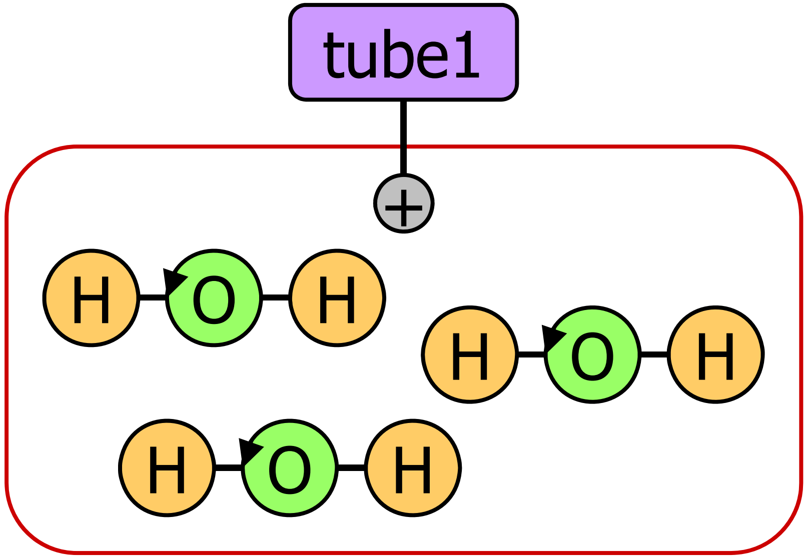
answer(2*x+5)

Text Representation

- Multisets (called a *cell* if enclosed by a membrane)



{100,100,100,10,10}




```
tube1(X), {+(X),
   'H'(A),'O'(A,B),'H'(B),
   'H'(C),'O'(C,D),'H'(D),
   'H'(E),'O'(E,F),'H'(F)}
```

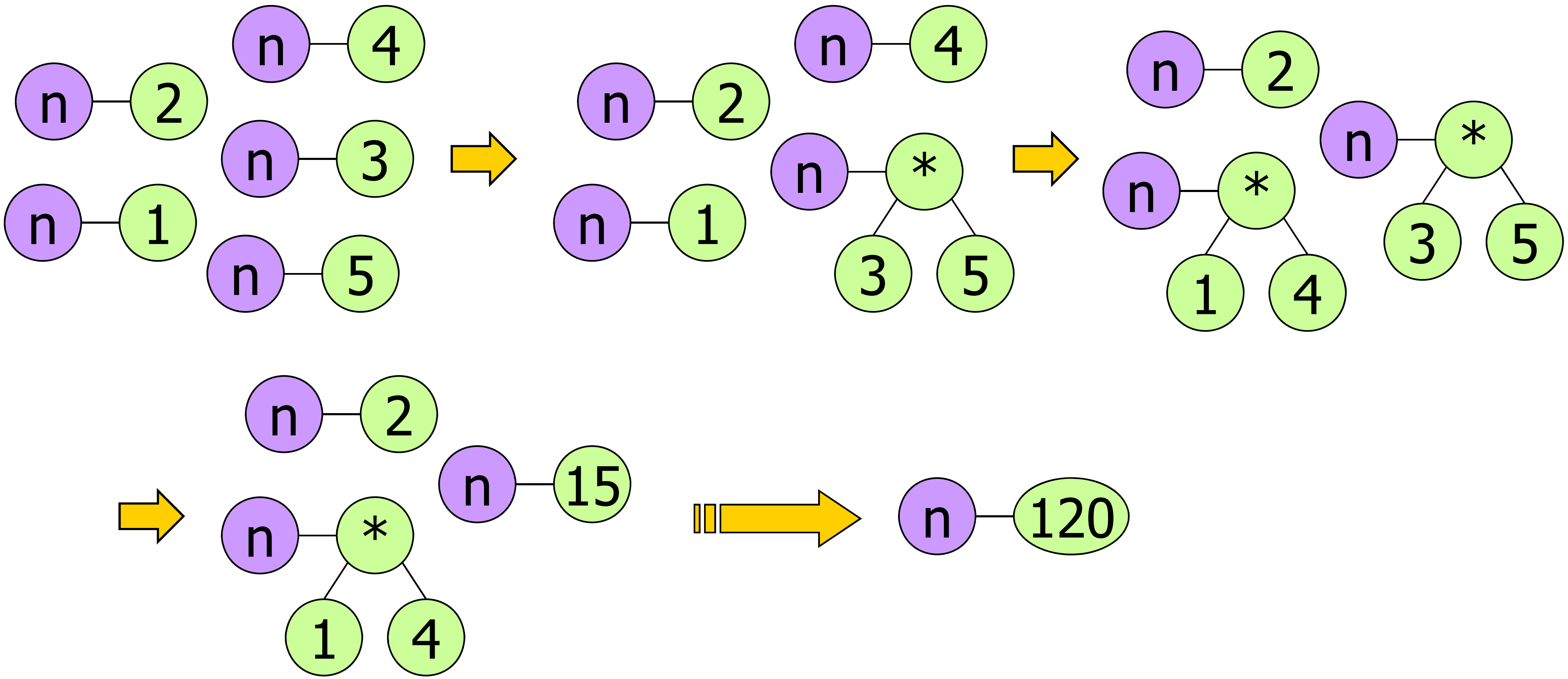
– or –

```
tube1({
   'O'('H','H'),
   'O'('H','H'),
   'O'('H','H')})
```

Demo: Factorial

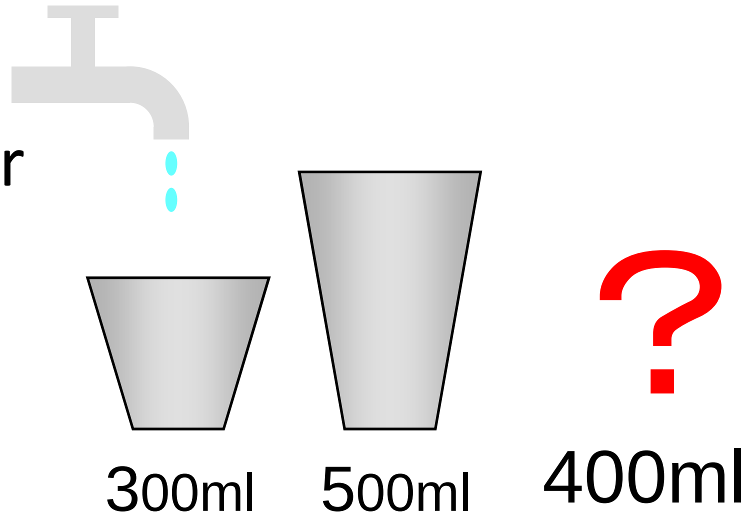
```
n(1), n(2), n(3), n(4), n(5).
n(A), n(B) :- n(A*B).
```

Numbers are unary atoms (= atom with a single link)

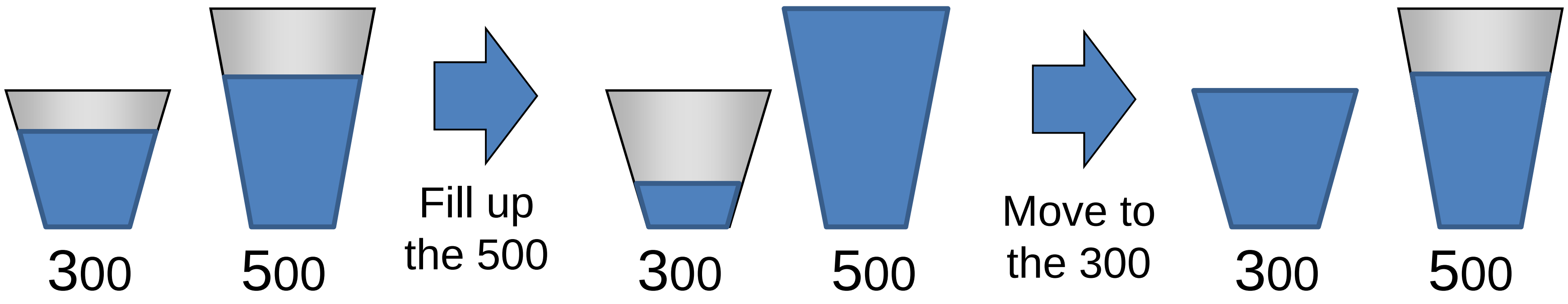


Demo: Water Jug Problem

- Given a 300ml jug and a 500ml jug, get 400ml of water

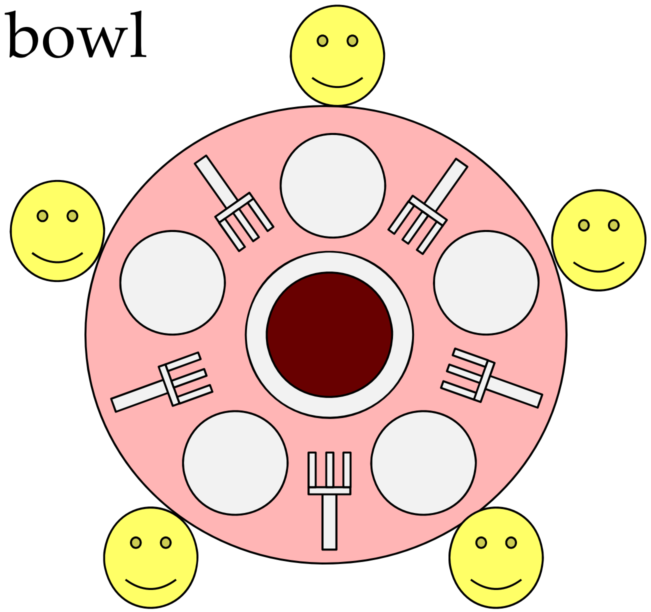


- Allowed operations:
  - Empty a jug
  - Fill up a jug with tap water
  - Move a jug’s water to the other until it’s emptied
  - Move a jug’s water until the other jug is filled up



Demo: Dining Philosophers (due to E.W.Dijkstra)

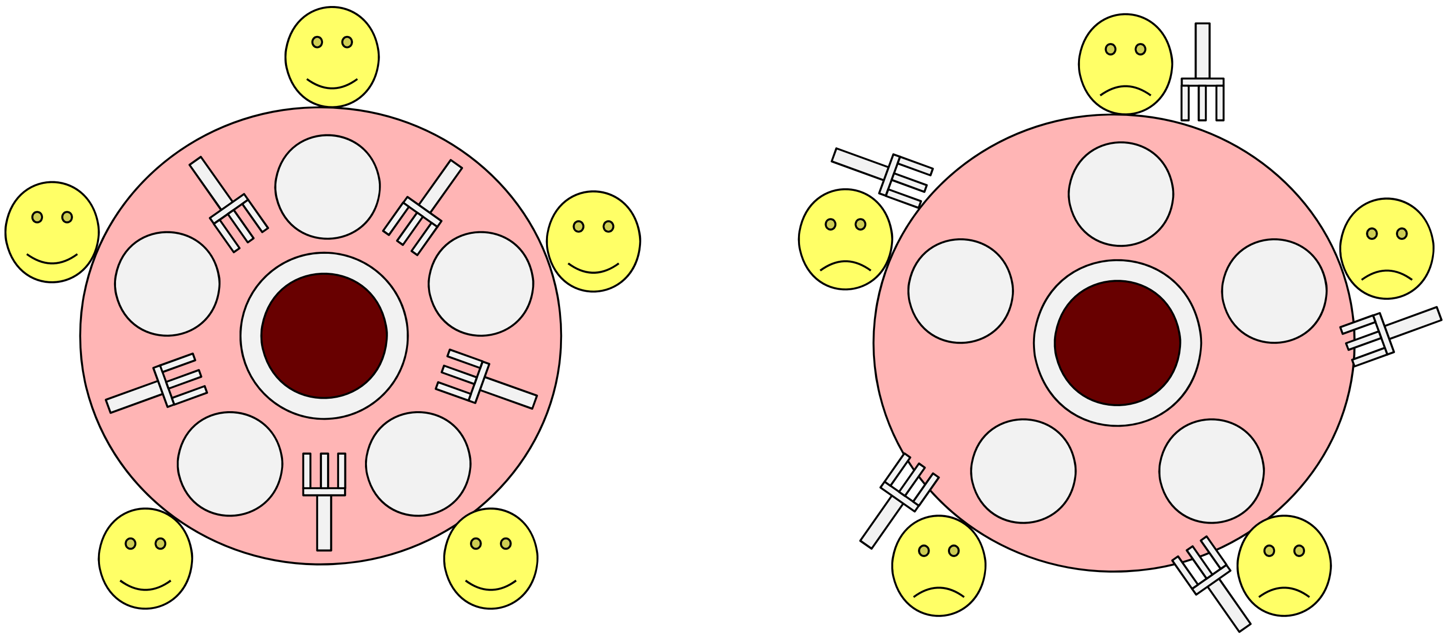
Five philosophers spend their lives thinking and eating. They share a common dining room where there is a circular table surrounded by five chairs, each belonging to one philosopher. In the center of the table there is a large bowl of spaghetti, and the table is laid with five forks. On feeling hungry, a philosopher enters the dining room, its in his own chair, and picks up the fork on the left of his plate. Unfortunately, the spaghetti is so tangled that he needs to pick up and use the fork on his right as well. When he has finished, he puts down both forks, and leaves the room.



— *C.A.R. Hoare (1978)*

Demo: Dining Philosophers

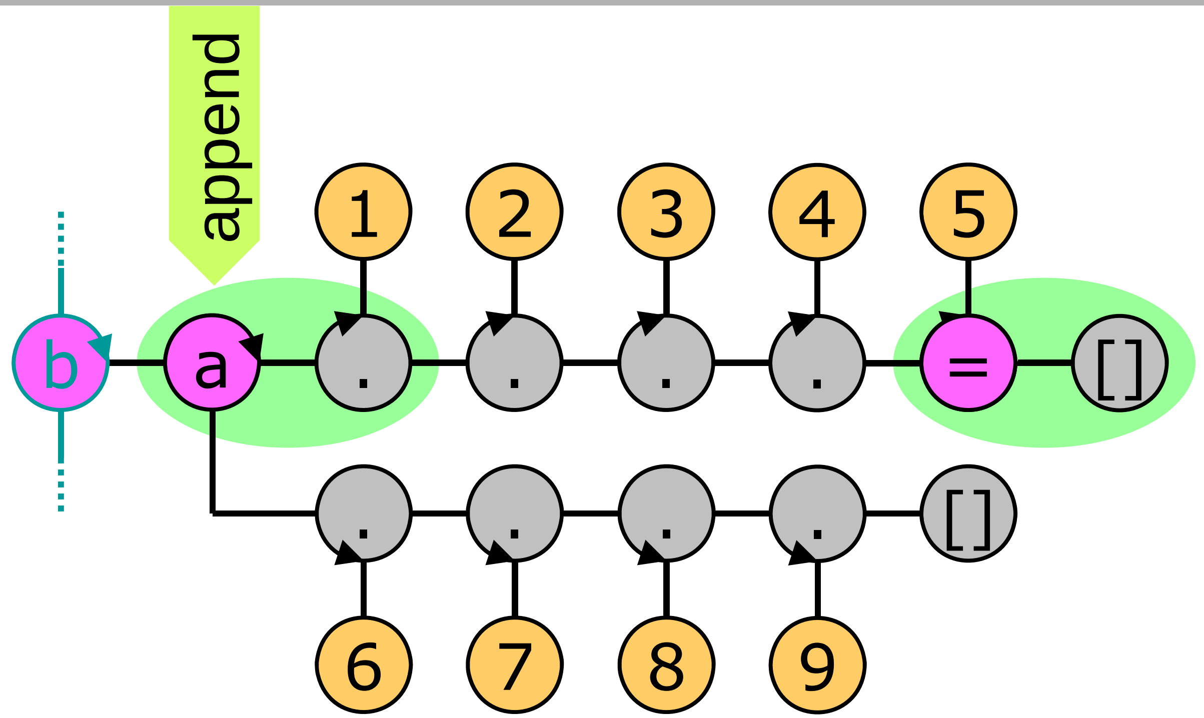
- A flock of mediocre philosophers causes deadlock . . .

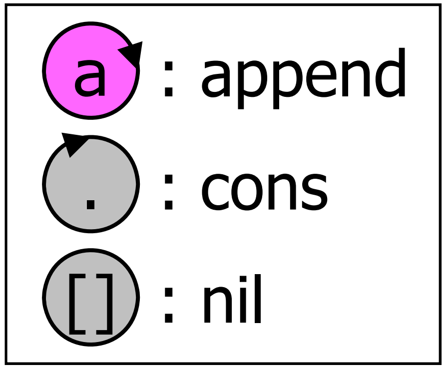


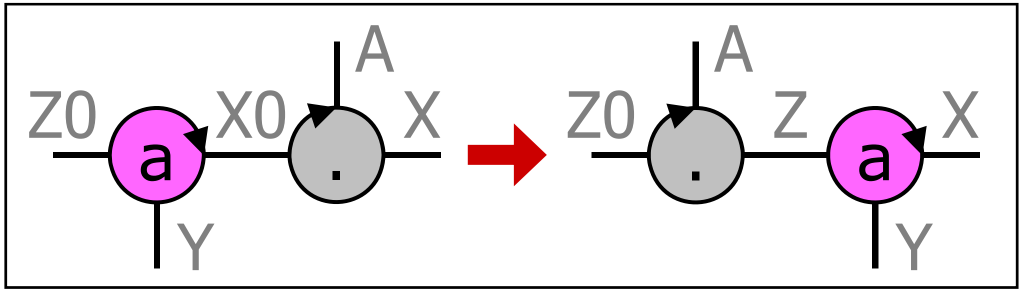
. . . but a perverse philosopher avoids deadlock!

- See how symmetry is reduced in state-space search.

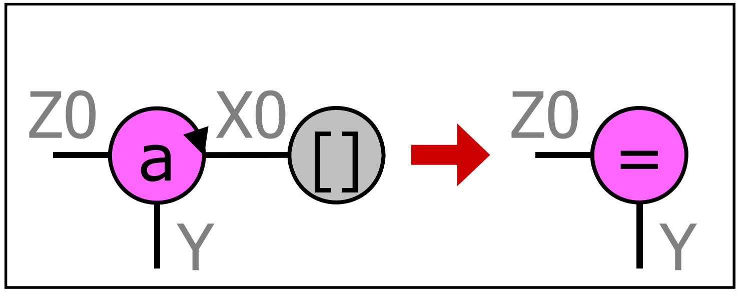
Demo: List Concatenation





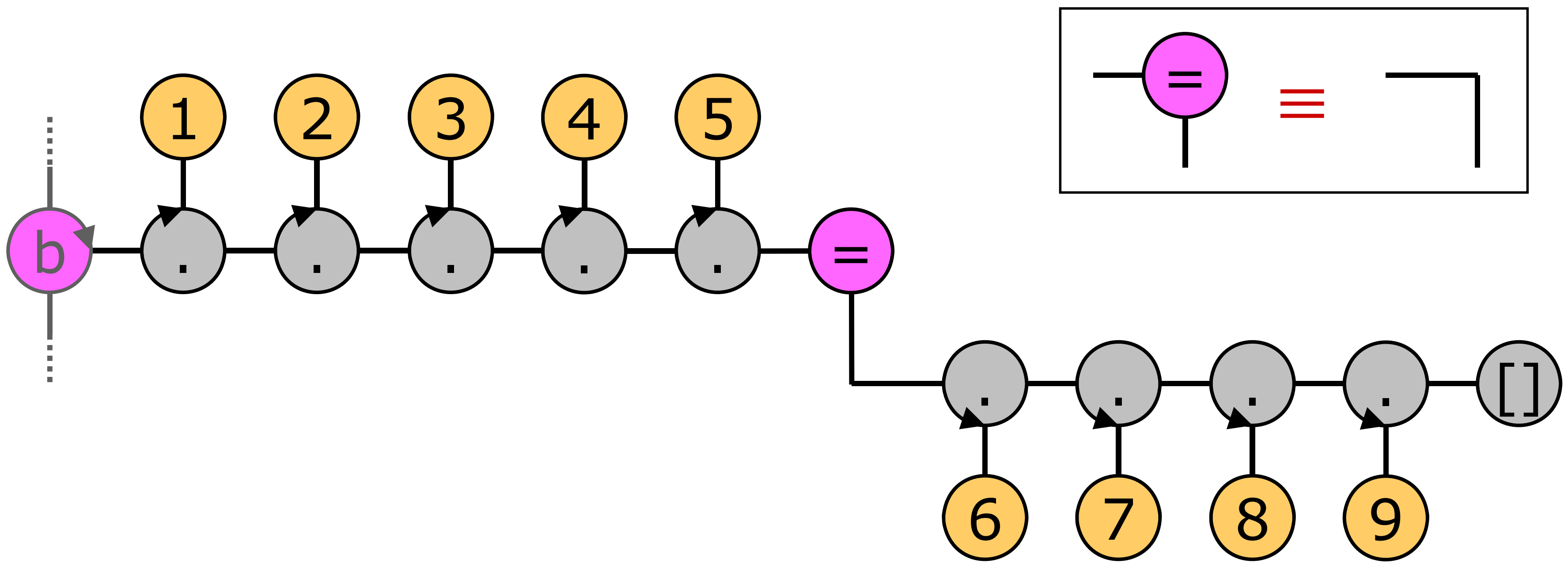


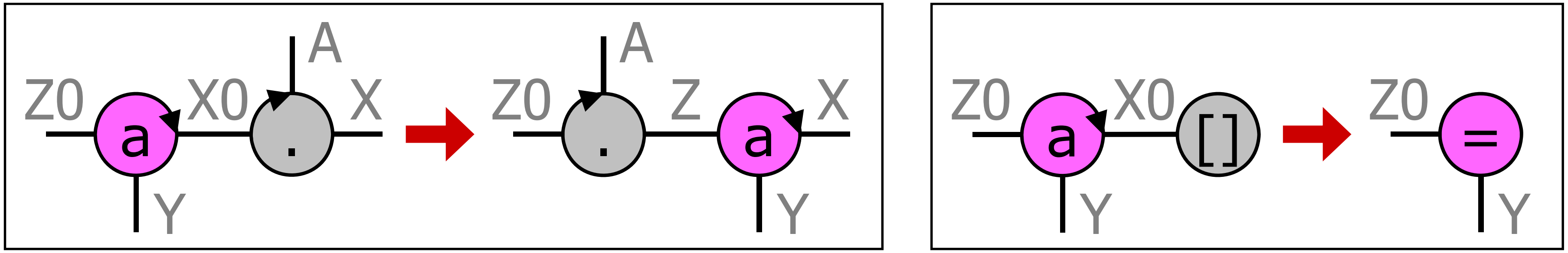
a(X0,Y,Z0), '.'(A,X,X0) :- '.'(A,Z,Z0), a(X,Y,Z)



a(X0,Y,Z0), '[]'(X0) :- Y=Z0

Demo: List Concatenation





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

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

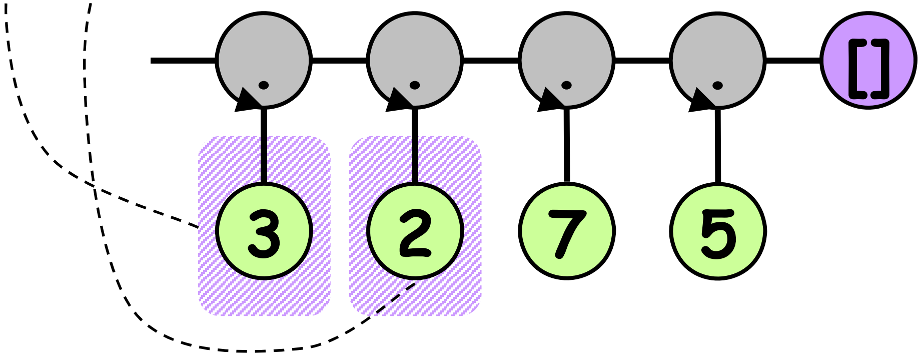
Demo: Nondeterministic Bubblesort (one rule)

typed process context

guard

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

compare and swap if $x > $y



The left-hand side may match the middle of a list.

Syntax and Semantics, In One Slide

$$
\begin{array}{rl}
\text{(process)} & P ::= 0 \mid p(X_1,\ldots,X_m) \mid P,P \mid \{P\} \mid T :- T \\
\text{(process template)} & T ::= 0 \mid p(X_1,\ldots,X_m) \mid T,T \mid \{T\} \mid T :- T \\
 & \quad \mid @p \mid \$p[X_1,\ldots,X_m|A] \mid p(*X_1,\ldots,*X_m) \\
\text{(residual)} & A ::= [\,] \mid X
\end{array}
$$

$$
\text{(E1)}\quad \mathbf{0},P \equiv P \qquad \text{(E2)}\quad P,Q \equiv Q,P \qquad \text{(E3)}\quad P,(Q,R) \equiv (P,Q),R
$$

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

$$
\text{(E5)}\quad P \equiv P' \Rightarrow P,Q \equiv P',Q \qquad \text{(E6)}\quad P \equiv P' \Rightarrow \{P\} \equiv \{P'\}
$$

$$
\text{(E7)}\quad X = X \equiv \mathbf{0} \qquad \text{(E8)}\quad X = Y \equiv Y = X
$$

$$
\text{(E9)}\quad X = Y,\ P \equiv P[Y/X] \qquad \text{if } P \text{ is an atom and } X \text{ occurs free in } P
$$

$$
\text{(E10)}\quad \{X = Y,\ P\} \equiv X = Y,\ \{P\} \qquad \text{if exactly one of } X \text{ and } Y \text{ occurs free in } P
$$

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

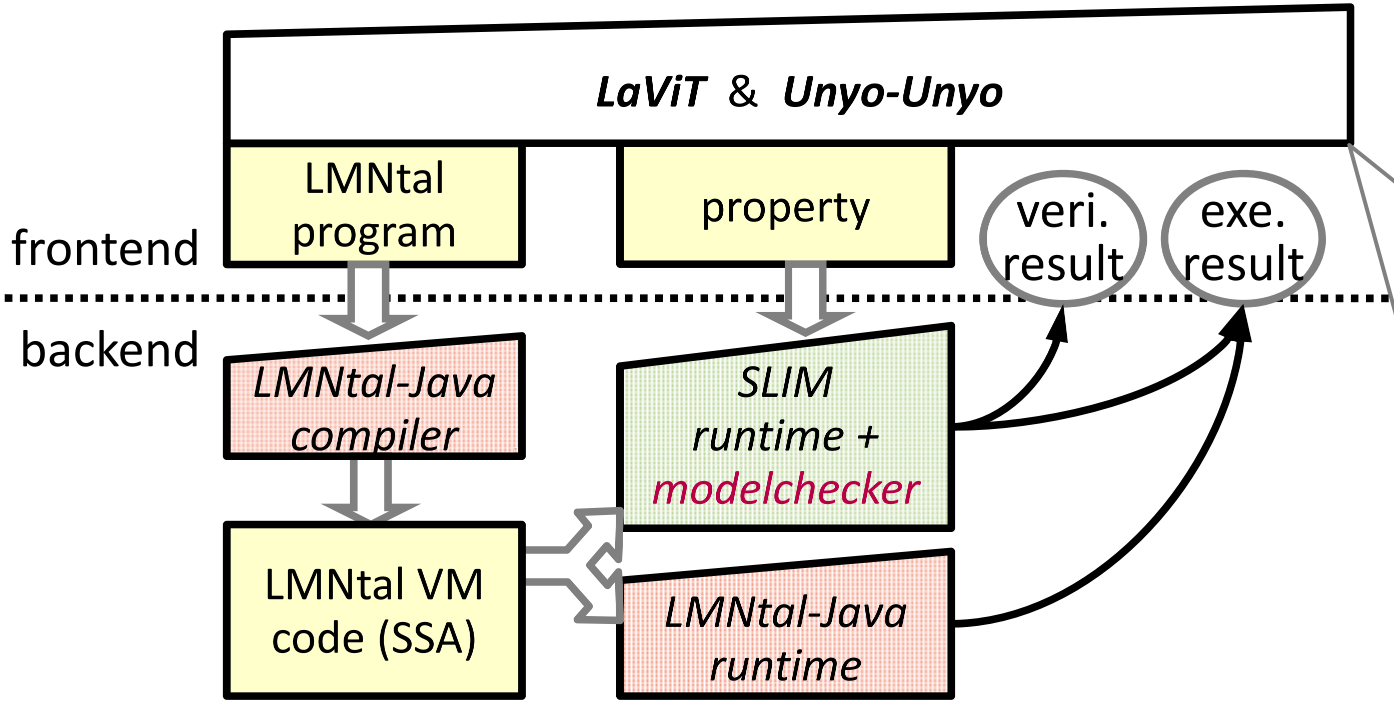
$$
\text{(R4)}\quad \{X = Y, P\} \longrightarrow X = Y,\ \{P\} \qquad \text{if } X \text{ and } Y \text{ occur free in } \{X = Y, P\}
$$

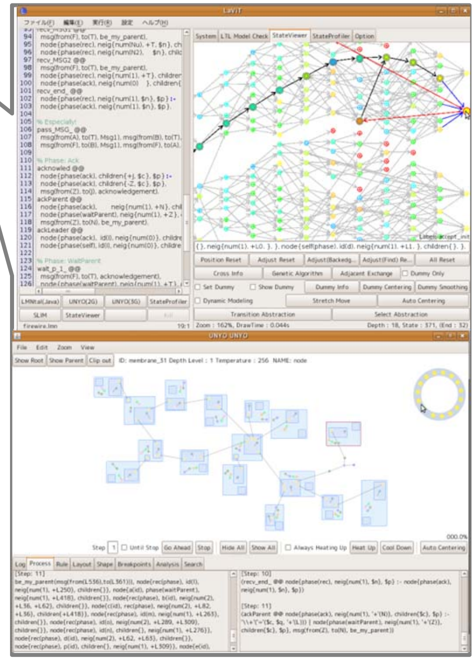
$$
\text{(R5)}\quad X = Y,\ \{P\} \longrightarrow \{X = Y, P\} \qquad \text{if } X \text{ and } Y \text{ occur free in } P
$$

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

Implementation Overview

LMNtal-Java	2002–	Java	40kLOC	compiler + runtime w/FLI
Unyo-Unyo	2006–	Java	18kLOC	execution visualizer
SLIM OSS	2007–	C	25kLOC	smaller and faster runtime parallel state-space search and model cheker
LaViT OSS	2008–	Java	18kLOC	IDE w/state-space visualizer

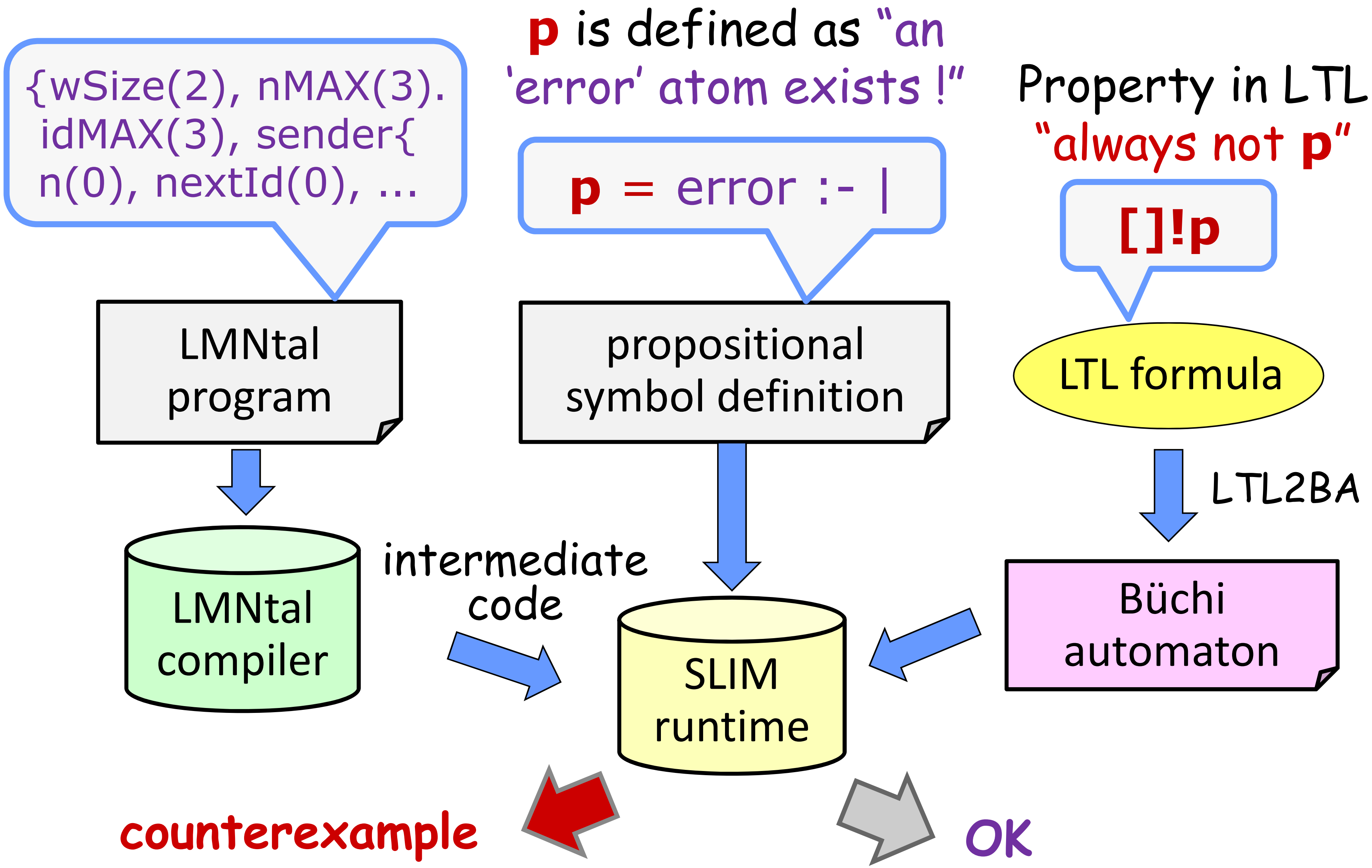




Model Checking in LMNtal: Motivations

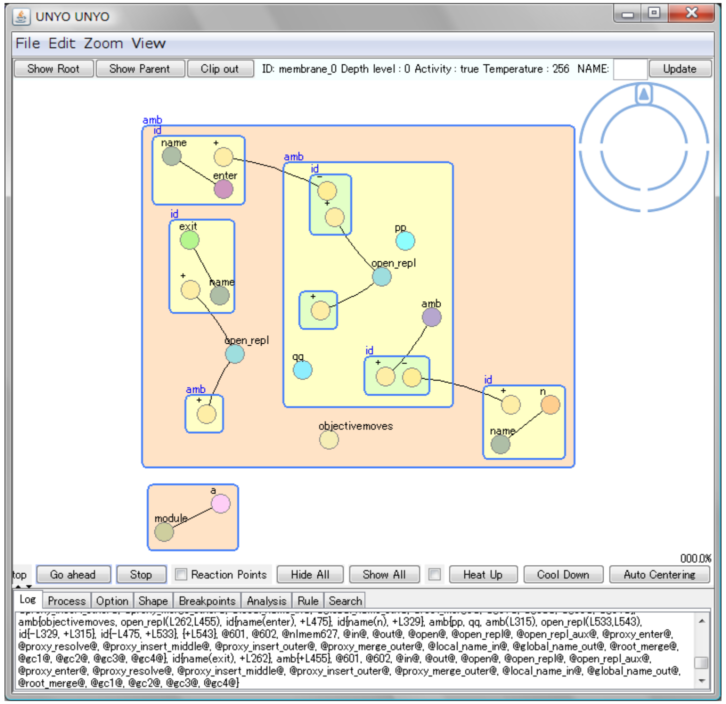
- LMNtal is good at modeling systems which computer-aided verification is concerned with, e.g.,
  - state transition systems (automata),
  - multiset rewriting systems, and
  - concurrent systems.
- LMNtal is at the same time a full-fledged programming language allowing infinite states.
  - No gap between modeling and programming languages (cf. SPIN, nuSMV, . . .)
  - As a fine-grained concurrent language, supporting verification is highly desirable
- Why not build an integrated development and verification environment?

LMNtal Model Checker

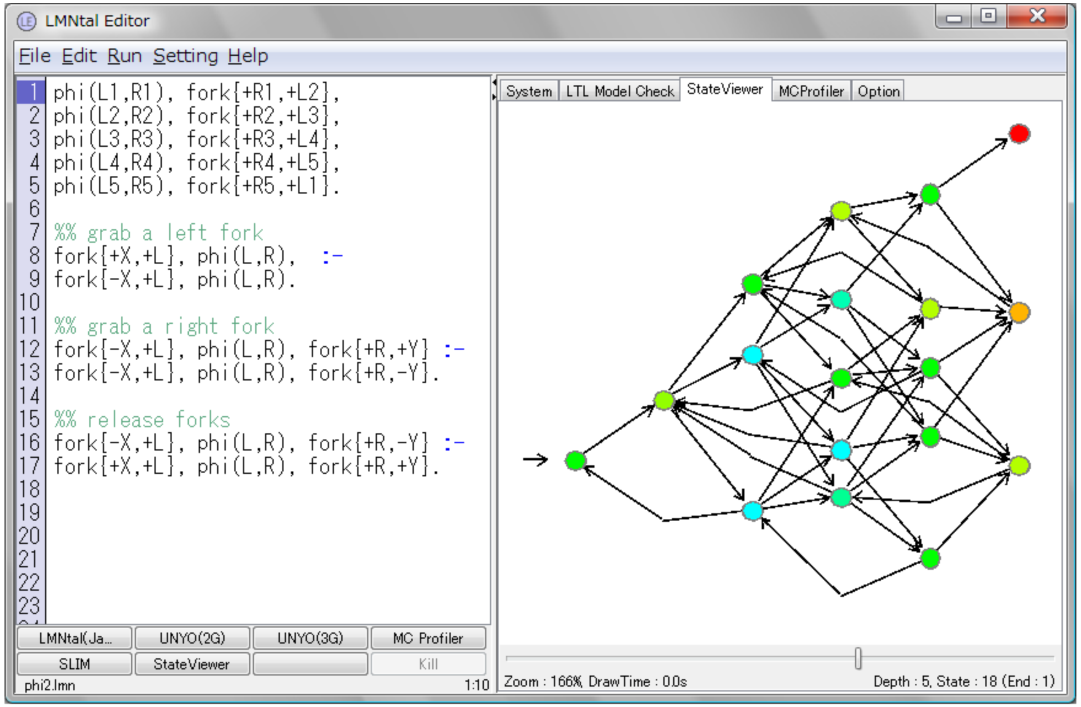


Model Checking in LMNtal: Strengths

- LaViT supports the understanding of models with and without errors, not just bug catching
  - workbench for designing and analyzing models
  - complementary to fast, black-box checkers
- Hierarchical graphs feature built-in *symmetry reduction*



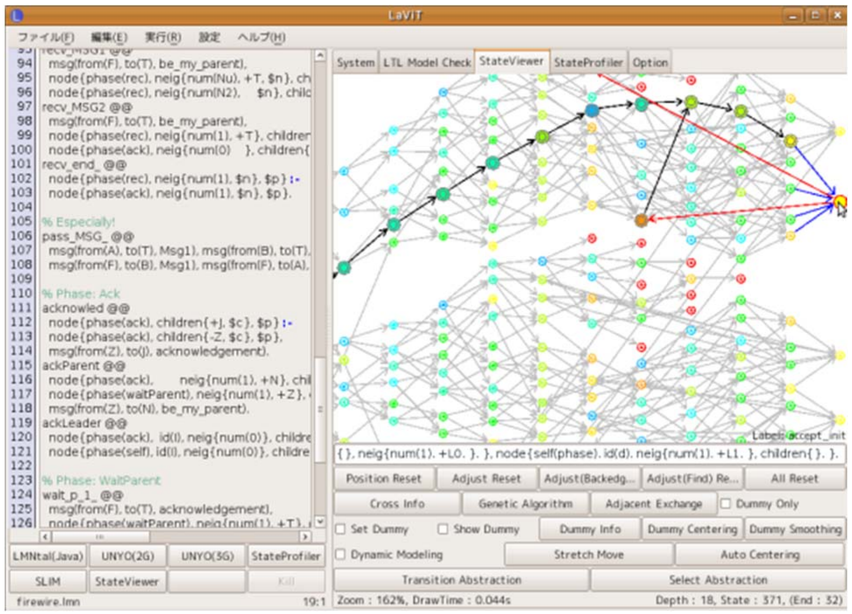
Unyo-Unyo Visualizer



StateViewer

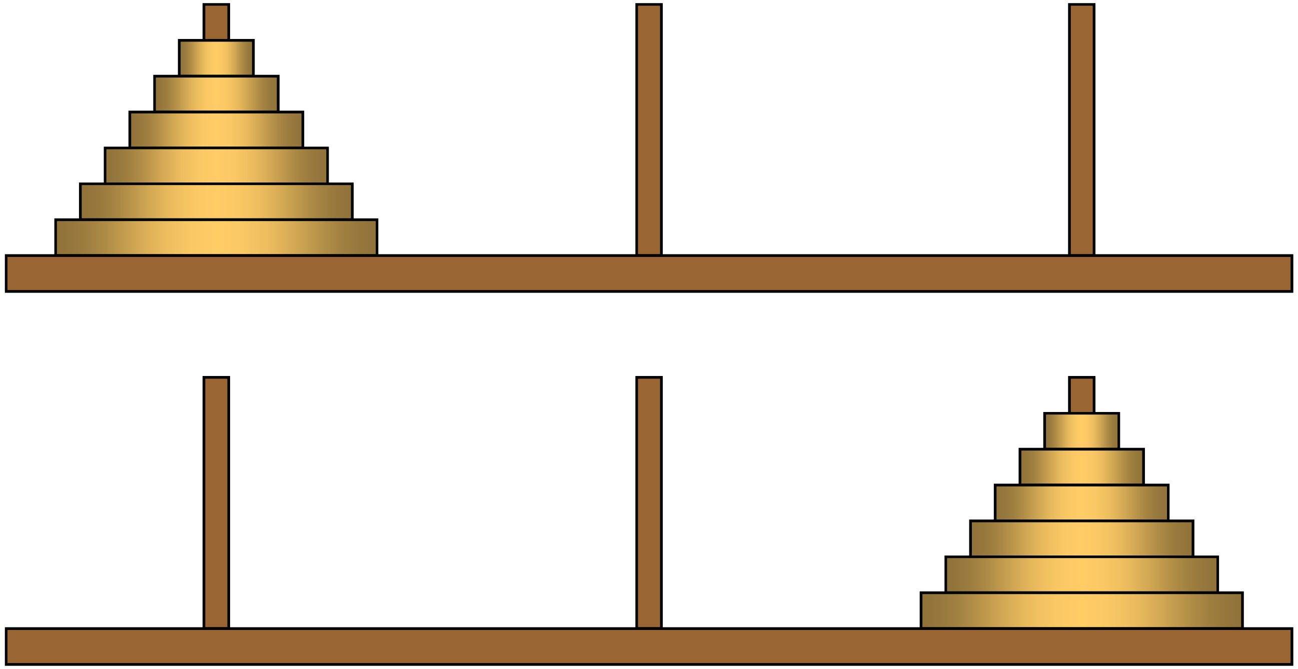
Model Checking in LMNtal: Challenges

- Challenges (some done, some ongoing):
  - States are graphs; efficient state management requires both hashing and graph isomorphism
  - Scalable parallel model checking
  - Real-time model checking
  - Extension to hypergraphs



Demo: The Tower of Hanoi

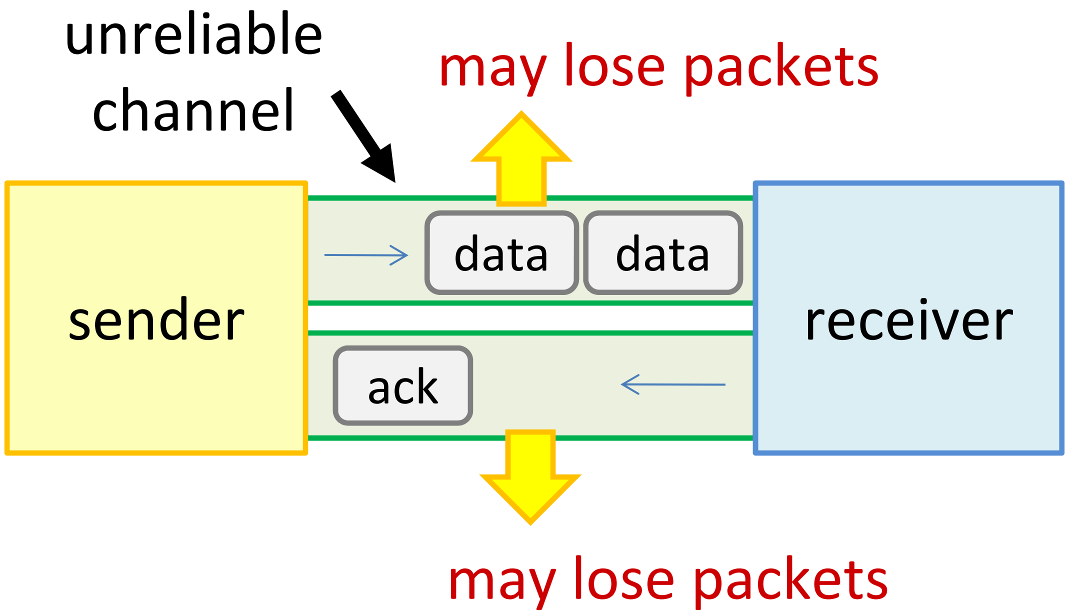
```
poles(p([1,2,3,4,5,6,99]),p([99]),p([99])).
```



```
P1=p([$h1|$t1]), P2=p([$h2|$t2]) :- $h1<$t2 |
    P1=p(T1), P2=p([$h1,$h2|$t2]).
```

Demo: Sliding Window Protocol (SWP)

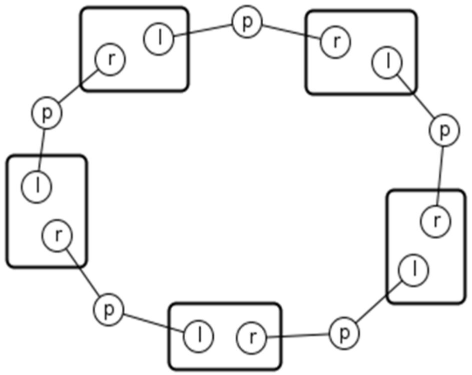
- SWP: transmission protocol used in TCP
  - Sends data packets (up to window size) without waiting for acknowlegment
    - Rollbacks if some item seems lost
  - Channels may lose data and acks



$\square(\text{send} \Rightarrow \Diamond\text{ack})$?

Coping with heavy structure and state explosion

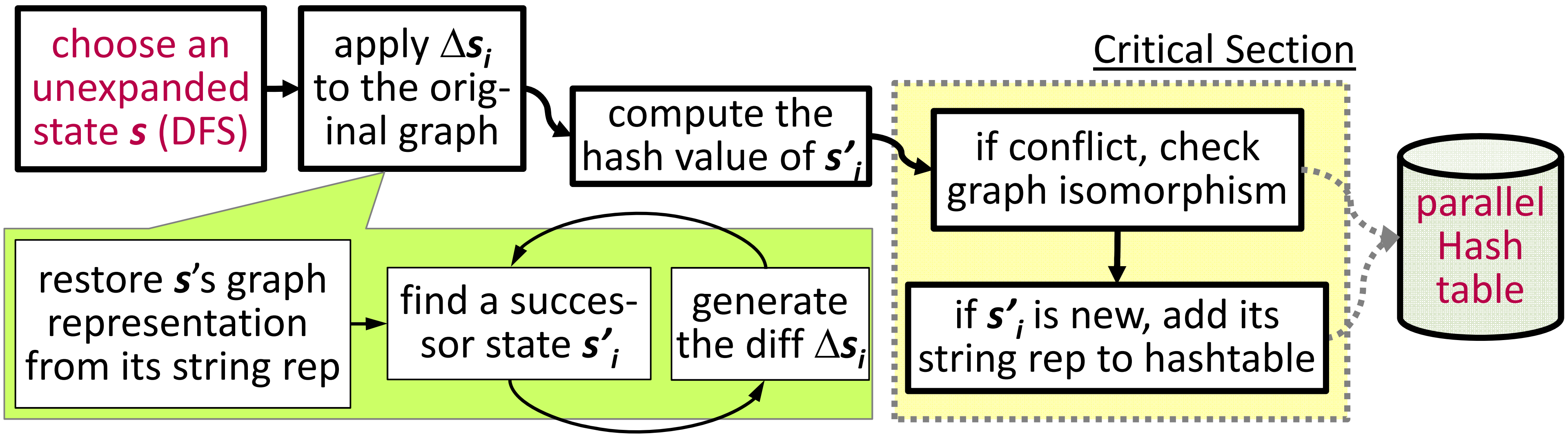
- Managing the state space of graphs requires both *space-efficient* graph representation and *time-efficient* isomorphism checking. They are supported by:
  - Hashing with parallel hashtable
  - Encoding (serialization)
    - Non-canonical encoding
    - Canonical encoding (labelling)
  - Backward execution
  - Parallel state-space search
  - Partial-order reduction



Original	>2KB
Encoded	70B

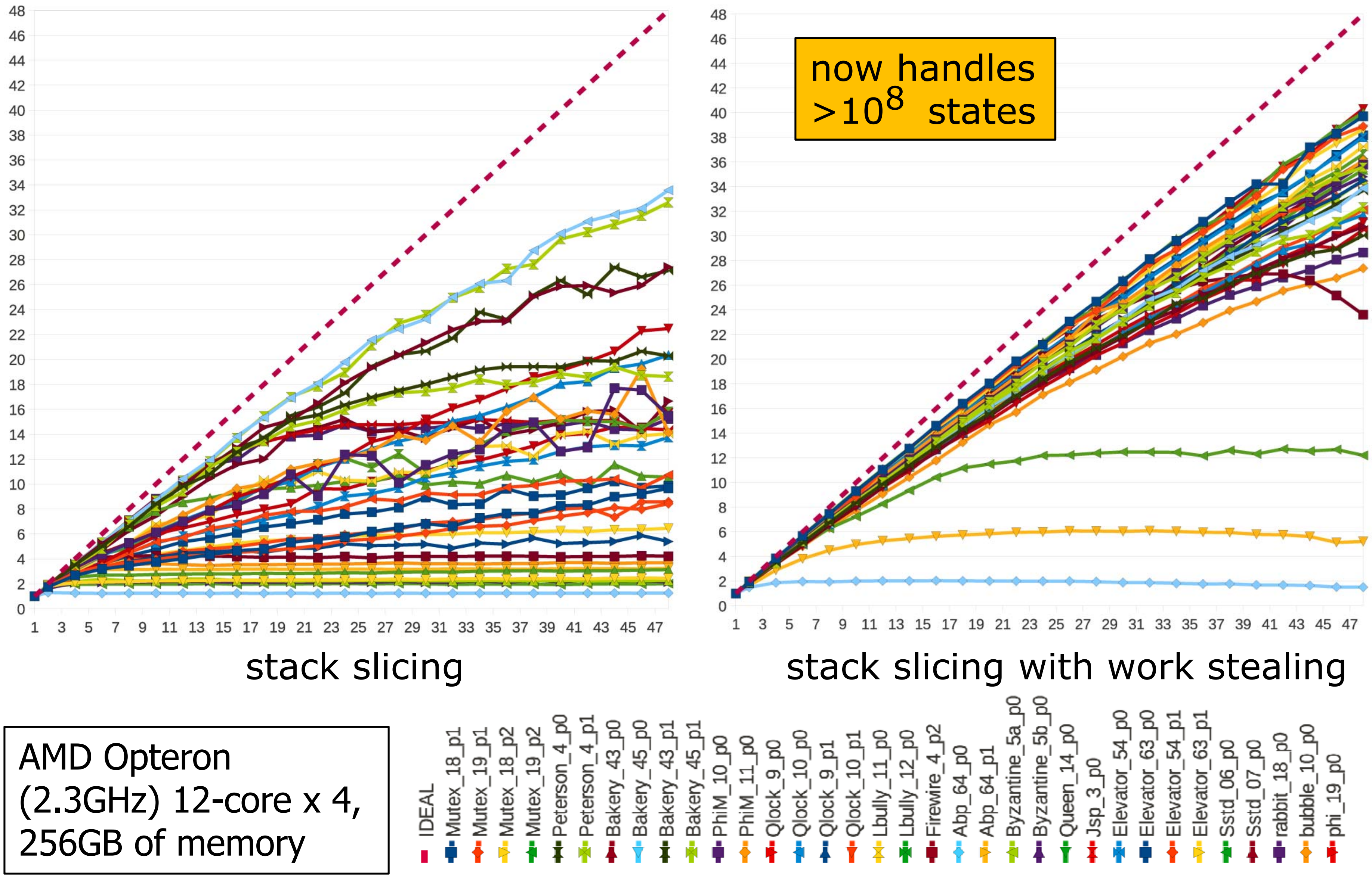
Parallelizing the LMNtal Model Checker*

- Construct a state-space graph by *stack-slicing parallel DFS*
- Apply an DiVinE-like algorithm to search a counterexample
- Built by (i) analyzing the sequential model checker (25kLOC), (ii) ensuring thread safety, and (iii) improving scalability on many-core processors
  - dynamic load balancing, parallel hash table
  - introducing parallel memory allocator



* M. Gocho, T. Hori and K. Ueda, *Computer Software* **28**(4), pp.137-157, 2011

Speedup of the LMNtal Parallel Model Checker

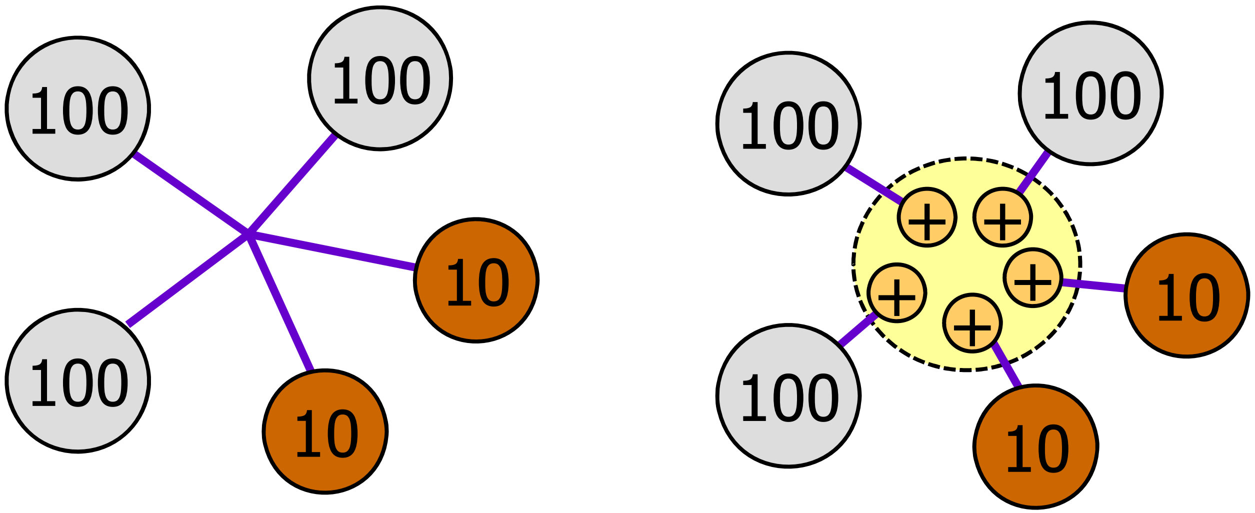


AMD Opteron (2.3GHz) 12-core x 4, 256GB of memory

M. Gocho, T. Hori and K. Ueda, *Computer Software*, **28**(4), 2011

Hyperlinks: motivations

- Need to express *references (pointers)*, *multipoint connection,* etc.
- Could be represented using membranes, but
  - membranes are heavyweight for just expressing partnership and sharing (rather than localization)
  - membranes are strictly hierarchical.



Hyperlinks: constructs

- Creating

```
H :- new($x,$att) | ... p(...$x...)...
H :- ... p(...!X:$att...)...
```

- Typechecking

```
... p(...$x...)... :- hlink($x) | B
... p(...!X...)... :- B
```

  - *... and other primitives for unary types* (==, ¥==, ...)

- Fusing

```
H :- ... | ... !X >< !Y ...
```

cf.

```
H :- ... | ... X = Y ...
```

- Cardinality

```
H :- num(!X,$n) | ...
```

- Subgraph copying

```
H :- hlground($x,$att) | p($x), p($x)
```

  - Attributes provide various views (slices) of a single hypergraph.

Links vs. Hyperlinks

Question: Links are a special case of hyperlinks. Why not unify them and support just hyperlinks?

Answer 1: No, we want to implement them in different ways.

- Implementation of links can be much simpler.

Answer 2: Yes, conceptually, and static analysis could distinguish between links and hyperlinks.

- However, programmers should probably be aware of the distinction.

Applications of Hypergraph rewriting

- Any apps that require rich data structures, e.g.,
  - Glass-box constraint programming
  - Type systems (and any formal systems in general)
  - Encoding of various calculi
    - including the λ–calculus

Encoding the Pure Lambda Calculus [RTA2008]

- [In theory terms] **Simple (re)formulation of the pure / strong / full λ–calculus** using a graph rewriting model + language **LMNtal**

- [In implementation terms] Execution engine of the pure λ–calculus that allows **out-of-order execution of (almost) O(1)-time microsteps**

The (Pure) λ-Calculus

- Syntax of lambda terms

$$
\begin{array}{lll}
M, N ::= x & & \text{(variable, name)} \\
 & \mid\ \lambda x.M & \text{(abstraction)} \\
 & \mid\ M\ N & \text{(application)}
\end{array}
$$

- Reduction relations

$$(\lambda x.M)N \to M[x := N] \qquad (\beta\text{-reduction})$$

$$\frac{M \to M'}{M\ N \to M'\ N} \qquad \frac{N \to N'}{M\ N \to M\ N'} \qquad \frac{M \to M'}{\lambda x.M \to \lambda x.M'}$$

The (Pure) λ-Calculus

éminence grise
of the λ–calculus
— Abadi et al. (1990)

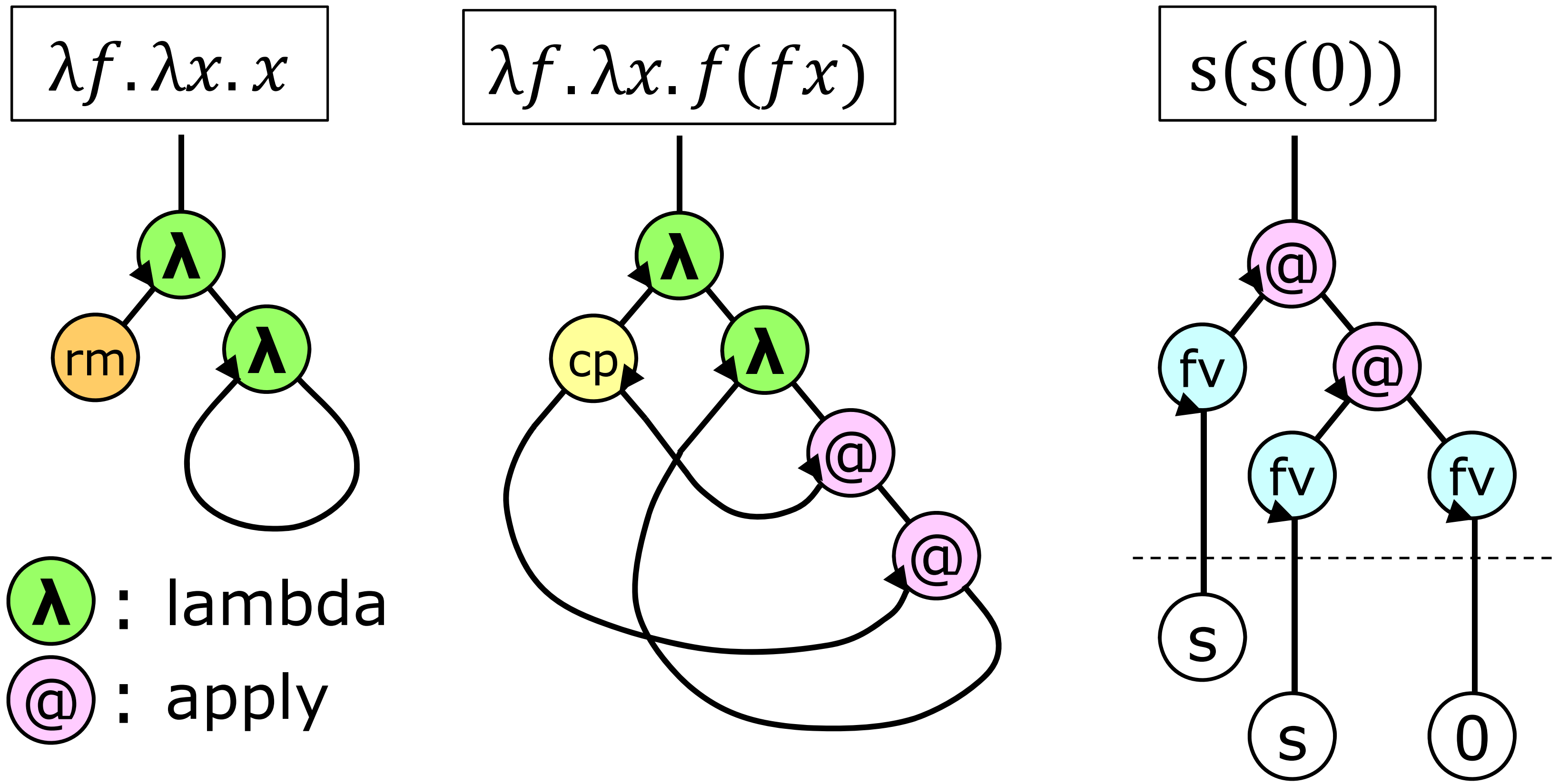
The usual impl. of functional programming languages betray the very spirit, i.e., the higher-order nature, of the λ-calculus. — Asperti (1998)

- Reduction relations

$$(\lambda x.M)N \to M[x := N] \quad (\beta\text{-reduction})$$

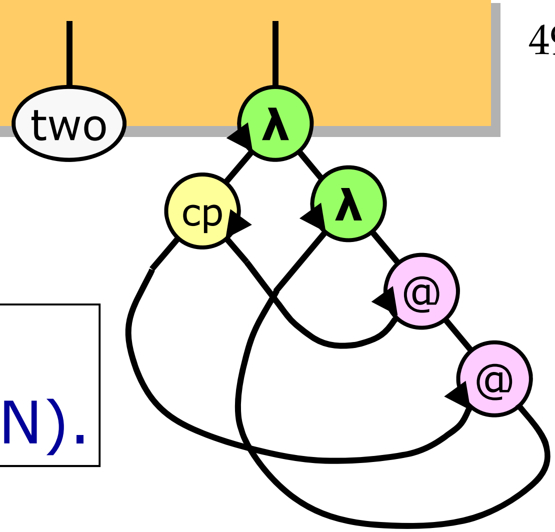
$$\frac{M \to M'}{M\,N \to M'\,N} \qquad \frac{N \to N'}{M\,N \to M\,N'} \qquad \frac{M \to M'}{\lambda x.M \to \lambda x.M'}$$

Encoding λ-Terms: Examples



Unique up to **cp**+**rm**'s equational theory
(= **ACU** : associativity & commutativity with unit)

Church Numeral Exponentiation



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

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

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

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

The Encoding (1/2)

graph copying

β-reduction

```
✓ H=apply(lambda(A,B), C)  :-  H=B, A=C.
✓ lambda(A,B)=cp(C,D,!L,!M)  :-
      C=lambda(E,F), D=lambda(G,H),
      A=cpc(E,G,!L1,!M1), B=cp(F,H,!L2,!M),
      sub(!L1,!L2,!L), subc(!M1).
✓ apply(A,B)=cp(C,D,!L,!M)  :-
      C=apply(E,F), D=apply(G,H),
      A=cp(E,G,!L,!M1), B=cp(F,H,!L,!M2), !M=jn(!M1,!M2).
✓ cp(A,B,!L1,!M1)=cp(C,D,!L2,!M2), sub(!L1,!L2,!L)  :-
      A=C, B=D, sub(!L1,!L2,!L), !L1 >< !M1, !L2 >< !M2.
✓ cp(A,B,!L1,!M1)=cp(C,D,!L2,!M2), top(!L2)  :-
      C=cpc(E,F,!L1,!M11), D=cpc(G,H,!L1,!M12),
      A=cp(E,G!L2,!M21), B=cp(F,H,!L2,!M22),
      !M1=jn(!M11,!M12), !M2=jn(!M21,!M22).
✓ fv($u)=cp(A,B,!L,!M) :-  unary($u) |
      A=fv($u), B=fv($u), !L >< !M.
```

The Encoding (2/2)

graph destruction

```
  lambda(A,B)=rm  :-  A=rmc, B=rm.
  apply(A,B)=rm  :-  A=rm, B=rm.
  cp(A,B,!L,!M)=rmc  :-  A=rmc, B=rmc, !L >< !M.
✓ cpc(A,B,!L,!M)=rm  :-  A=rm, B=rm, !L >< !M.
  A=cp(B,rm,!L,!M)  :-  A=B, !L >< !M.
  A=cp(rm,B,!L,!M)  :-  A=B, !L >< !M.
  rmc=rm  :-  .
  fv($u)=rm  :-  unary($u) | .

✓ subc(!L1), sub(!L1,!L2,!L3) :- !L2 >< !L3.
  A=cp(B,C)  :-  A=cp(B,C,!L,!M), top(!L), topc(!M).
  top(!L), topc(!L)  :-  .
  !Y=jn(!X,!X) :- !X >< !Y.
```

color management

✓: Eight essential rules (the other rules are for tidying up and initialization)

Related Work

- $\lambda\sigma$-calculus (explicit substitutions)
- Encodings into Interaction Nets have been either
  - weak (Sinot, ...)
    - $\boldsymbol{x\, M_1\, M_2 \ldots M_n}$ $(n \geq 0)$
    - $\boldsymbol{\lambda x\, .\, M}$ (abstraction body not evaluated)

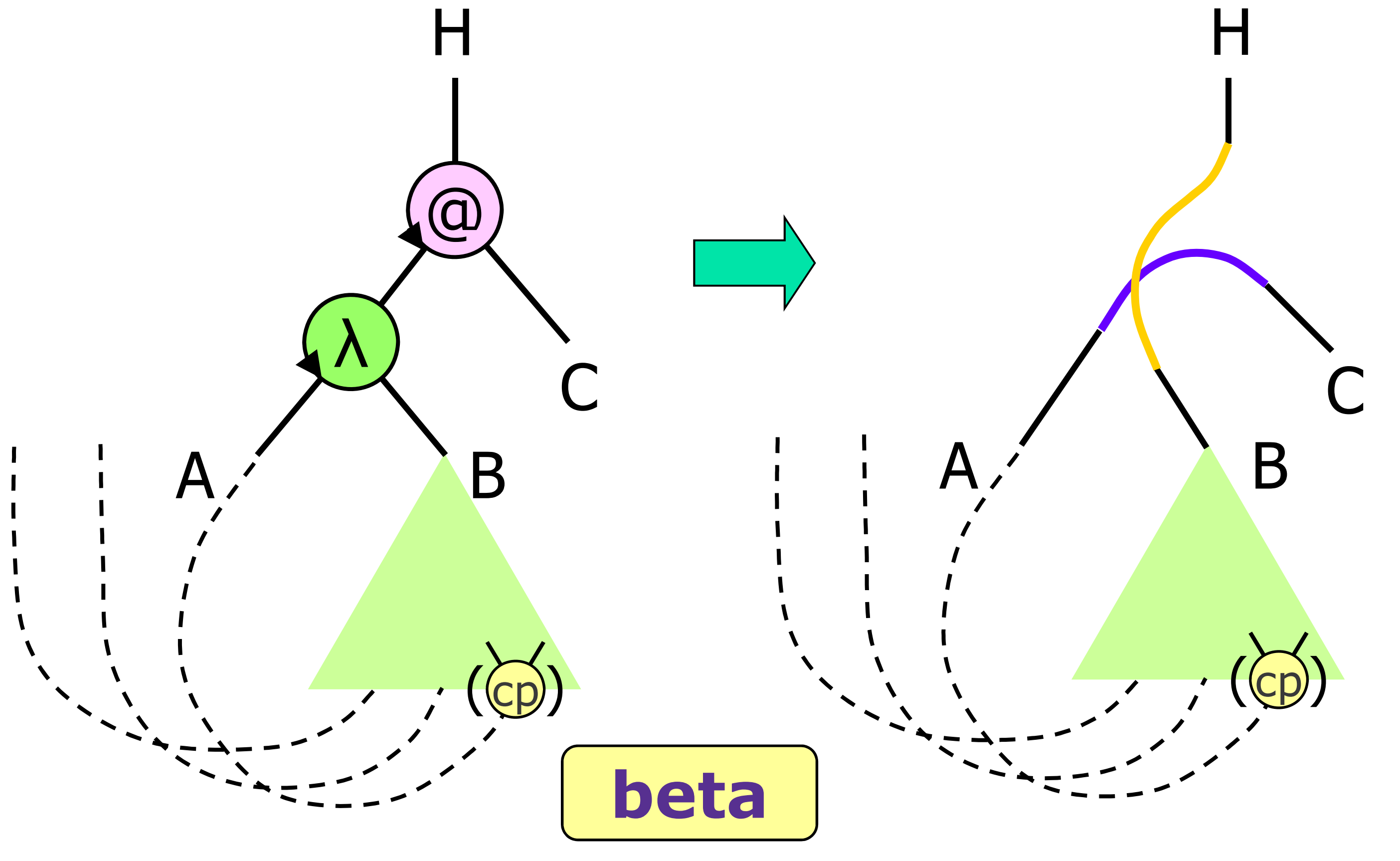
  or

  - not very simple (Lamping, Asperti, Mackie, ...)
    - in the sense that they “decorate” graphs

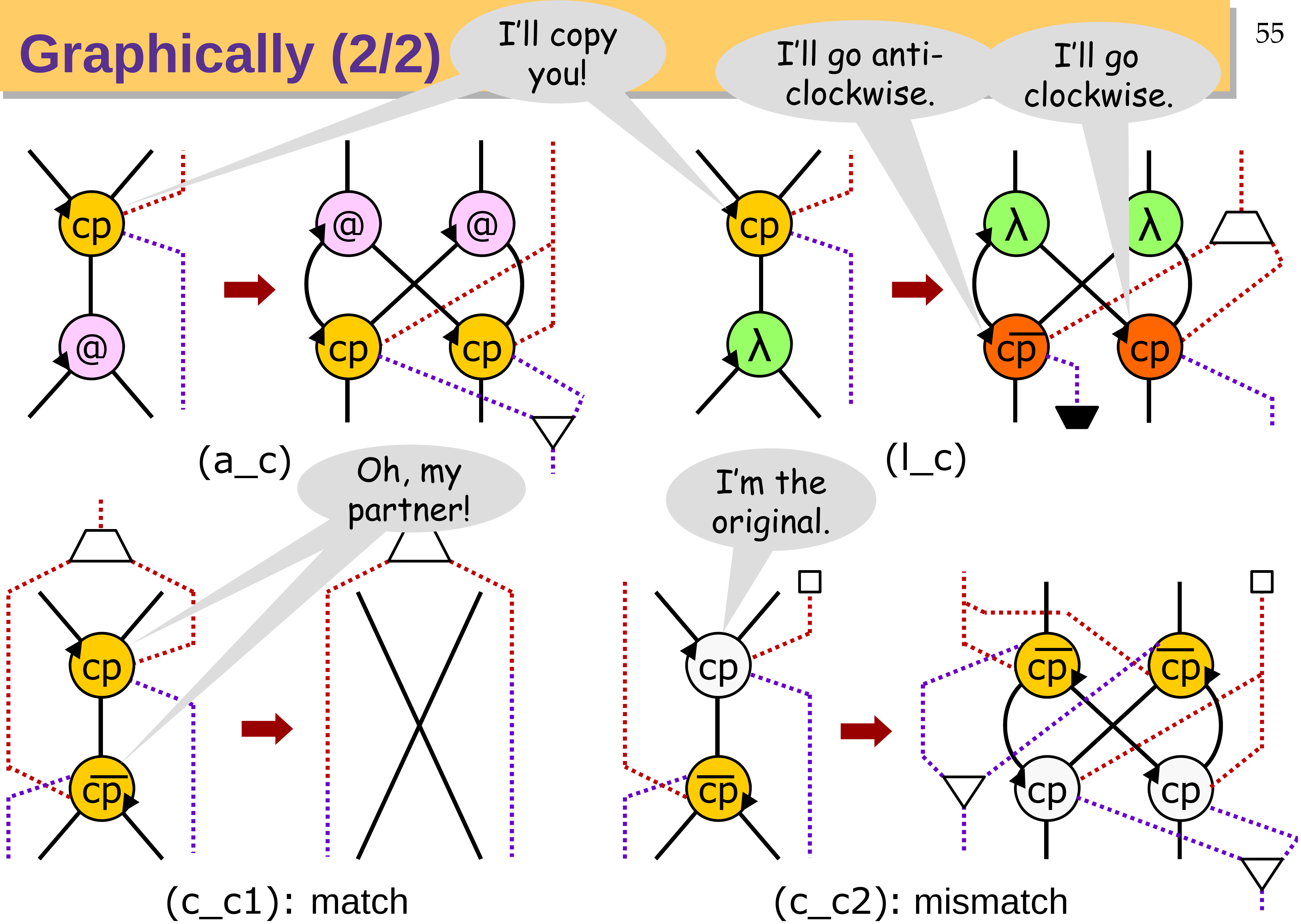
➔ Does LMNtal provide useful constructs and techniques for more concise encoding?

Graphically (1/2)

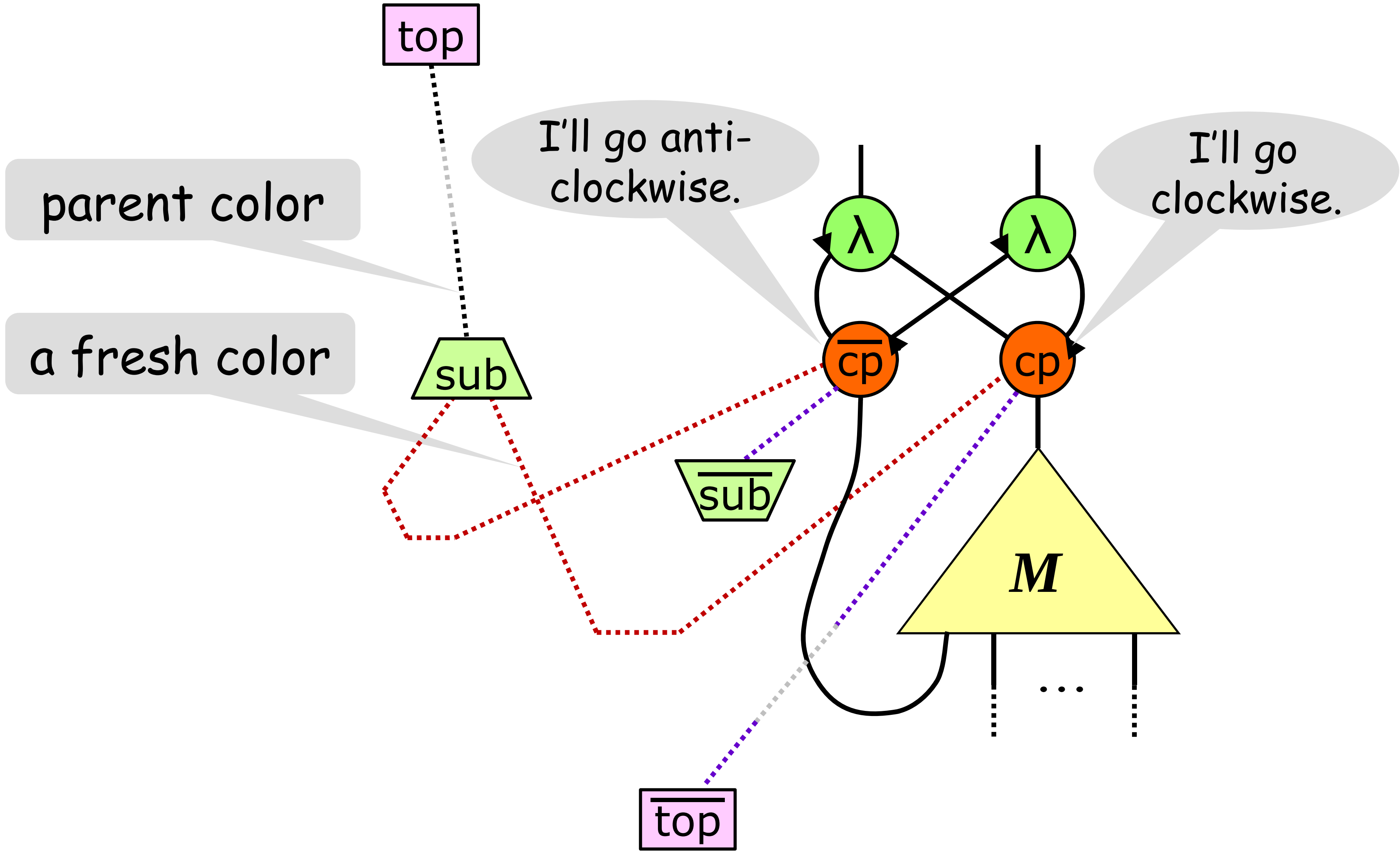
H=apply(lambda(A,B), C) :- H=B, A=C.



Graphically (2/2)



The RHS of the l_c rule

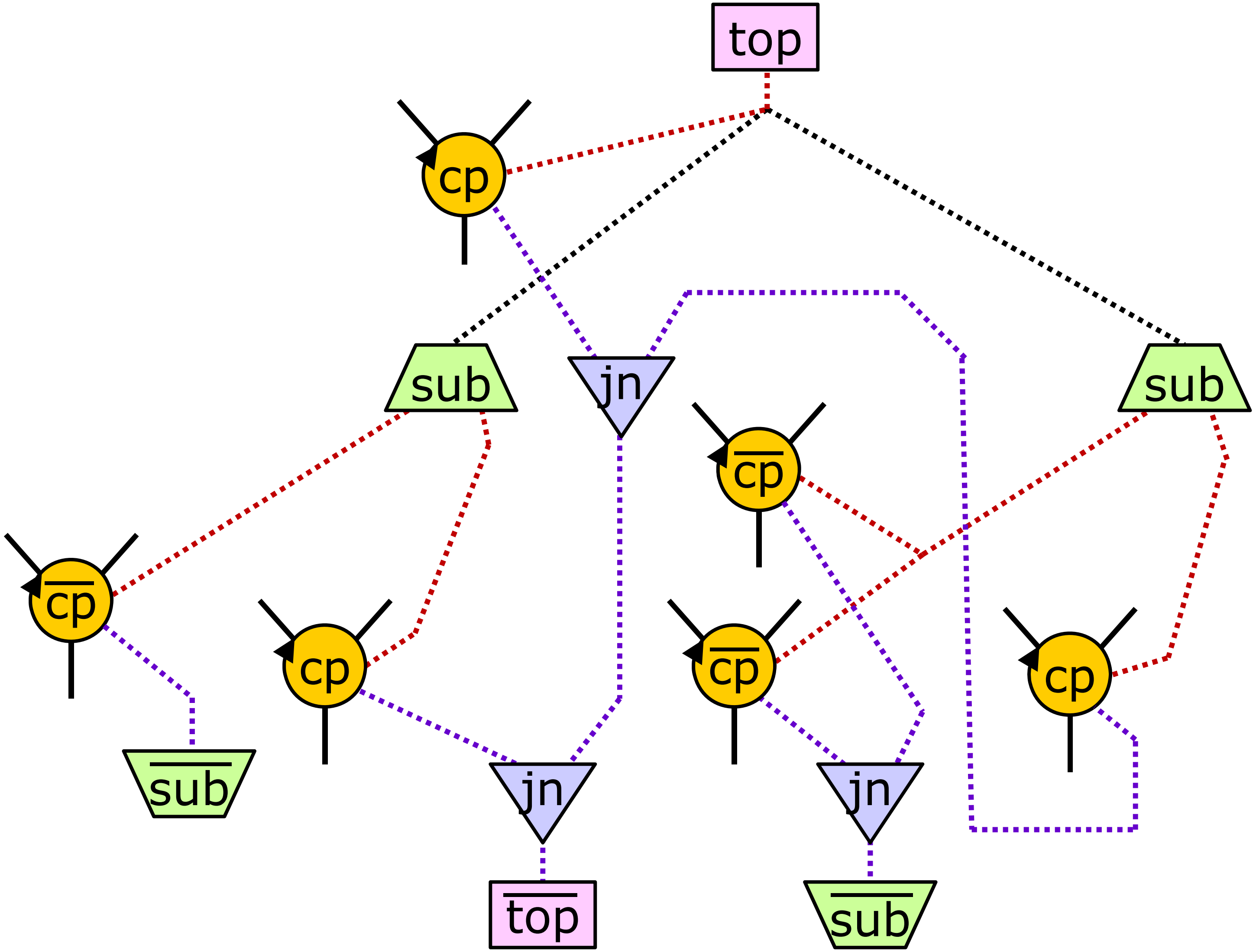


The Key Idea

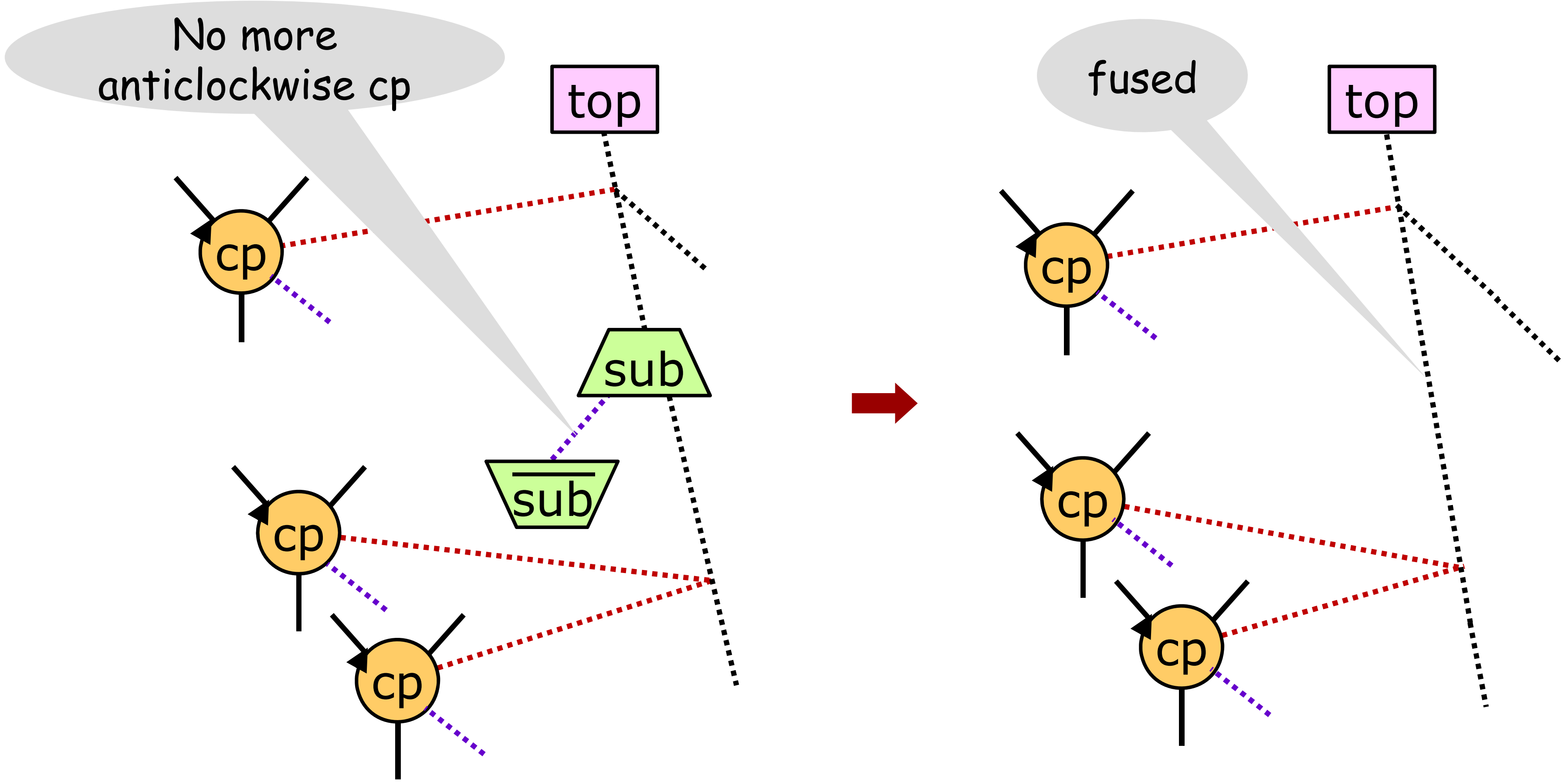
- Which of **c_c1** and **c_c2** to apply?
- Existing methods used two colors or natural numbers to label **cp**'s
- We employ hierarchical colors (= **local names**)
  - whenever a **cp** encounters a λ, two complementary **cp**'s are created to copy the abstraction.
  - when all the new **cp**'s running anti-clockwise hit their partners and disappear, the remaining **cp**'s become **cp**'s.

Color (= local name) Management

- Colors are encoded using hyperlinks.



Promotion (color fusion) (cf. "retirement")



Good news: Promotion need not be instantaneous; can be delayed safely.

Related work: Models and languages with multisets **and symmetric join**

- (Colored) Petri Nets
- Production Systems and RETE match
- **Graph transformation formalisms (e.g., Groove)**
- CCS, CSP
- Concurrent logic/constraint programming
- Linda
- Linear Logic languages
- **Interaction Nets**
- **Chemical Abstract Machines**
- **Gamma model**
- **Maude**
- **Constraint Handling Rules**
- Mobile ambients
- P-system, membrane computing
- Amorphous computing
- **Bigraphs**

Models and languages with membranes + hierarchies

* : some versions feature hierarchies

- (Colored) Petri Nets
- Production Systems and RETE match
- **Graph transformation formalisms** *
- CCS, CSP
- Concurrent logic/constraint programming
- Linda *
- Linear Logic languages
- **Interaction Nets**
- **Chemical Abstract Machines**
- **Gamma model**
- **Maude**
- **Constraint Handling Rules**
- Mobile ambients
- P-system, membrane computing
- Amorphous computing
- **Bigraphs**
- Statecharts
- Seal calculus
- Kell calculus
- Brane calculi
- κ

Experiences and Conclusions

- Designed and implemented (Hyper)LMNtal as a unifying computational model offering fine-grained concurrency
- Graph-based model checking (with up to ~10^8 states) works with many implementation techniques
  - Need to maintain one than one algorithm
- Built an LMNtal IDE as a unified framework of computation and verification
  - Visualization turned out to be very useful for understanding systems
- http://www.ueda.info.waseda.ac.jp/lmntal/

  (choose LaViT)