Hierarchical Port Hypergraphs:

Two Dacades Toward a Unifying Structure for Declarative Languages

Kazunori Ueda

Waseda University, Tokyo, Japan

FLOPS 2026, Tsukuba, May 27, 2026

Declarative languages

- Umbrella term with a broad spectrum ranging
  from higher-order terms
  to concurrent processes with various forms of communication

- Research Question: Can many—if not all—of them can be *unified* within a simple yet expressive formalism, rather than merely *combined* into a larger formalism?

- Graphs and graph rewriting provide a promising framework,
  e.g., trees augmented with binding structures
  reconfigurable network of communicating processes
  but what *language constructs* cover this spectrum in a unified manner?

This talk

- will describe our efforts toward the "unifying structure" (to be detailed soon) for declarative languages.
- Specifically, we focus on the design and implementation of LMNtal, which was conceived as an attempt to unify
  - Constraint-Based Concurrency (a.k.a. Concurrent Constraint Programming) and
  - Constraint Handling Rules (CHR),

  two notable extensions to logic programming, which then acquired an interpretation as a hierarchical (1) port (2) hypergraph (3) rewriting formalism/language.

LMNtal ("elemental")

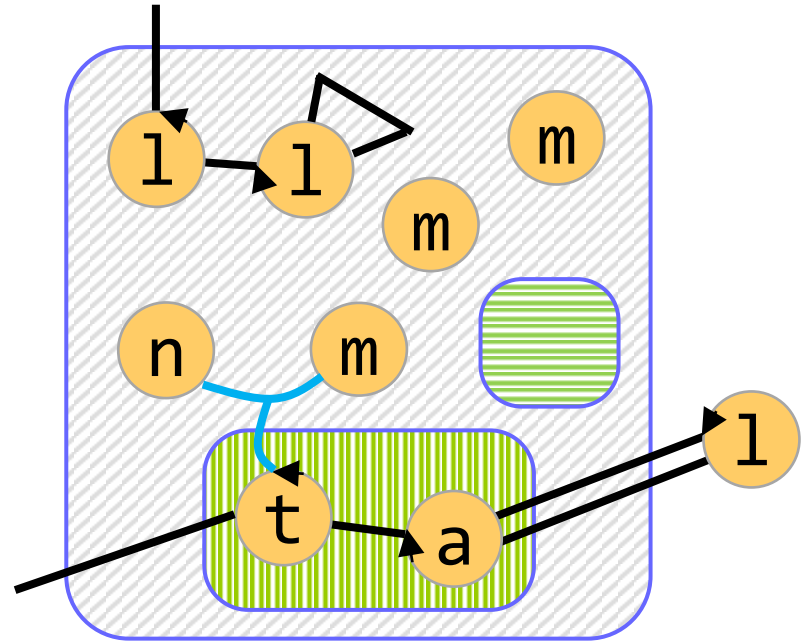
$\mathcal{L}$ = "logical" links

$\mathcal{M}$ = multisets/membranes

$\mathcal{N}$ = nested nodes

ta = transformation

$\mathcal{l}$ = language



node-labelled,
open,
hierarchical,
undirected
port-(multi)(hyper)graph

Portal: https://www.uedalab.jp/lmntal/
GitHub: https://github.com/lmntal/

Project LMNtal (2002–)

- Computational model + language + toolchain
  based on (a class of) *graph rewriting*
- >100,000 LOC involving many people over the years
- Features *model checking* since 2007
- Provides LaViT, an IDE with *visualizers*

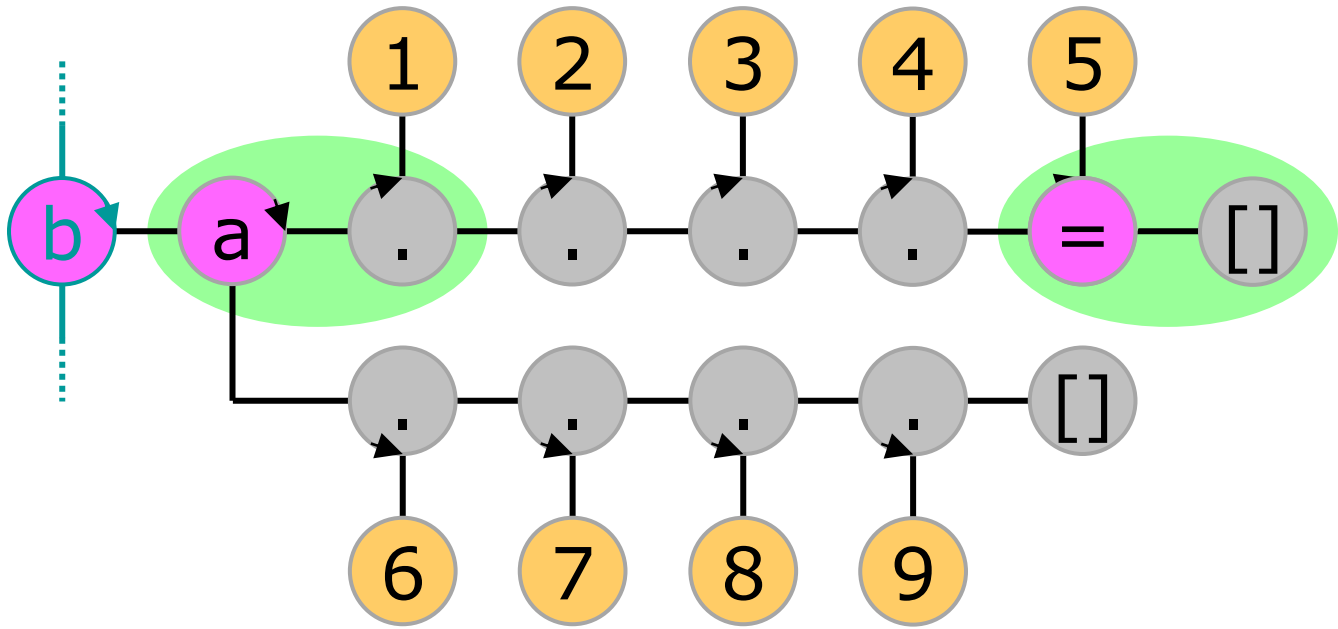
Ready to use; very low entry barrier

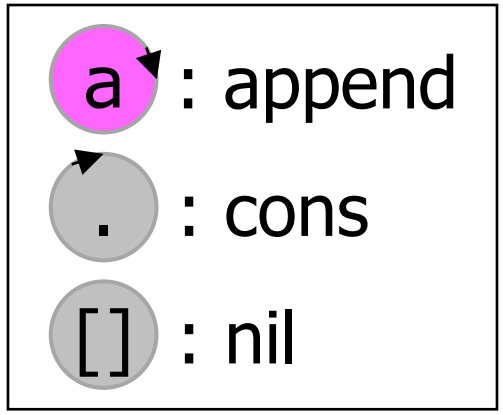
http://www.uedalab.jp/lmntal/

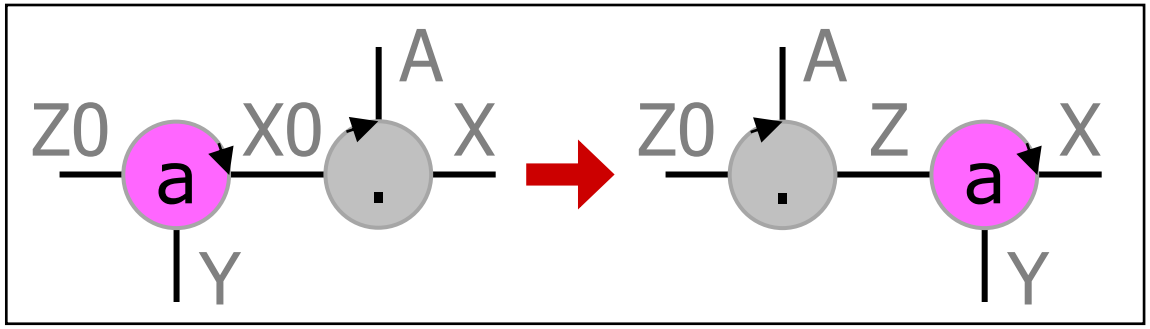
- open-source from GitHub

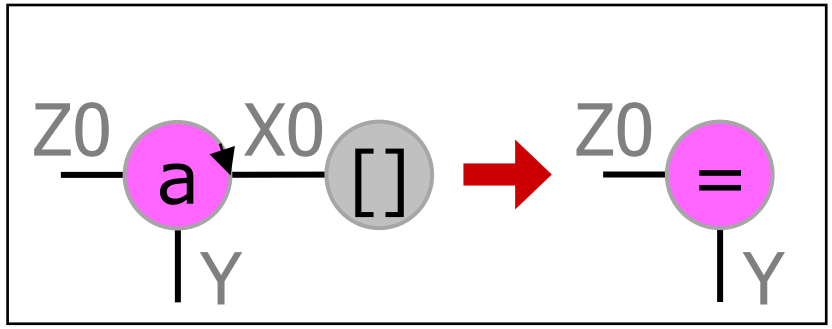
K. Ueda, *Theoretical Computer Science* **410**, 2009
K. Ueda et al., *Proc. ICTAC 2009*, LNCS 5684

Demo: List concatenation (*a la* Interaction Nets)





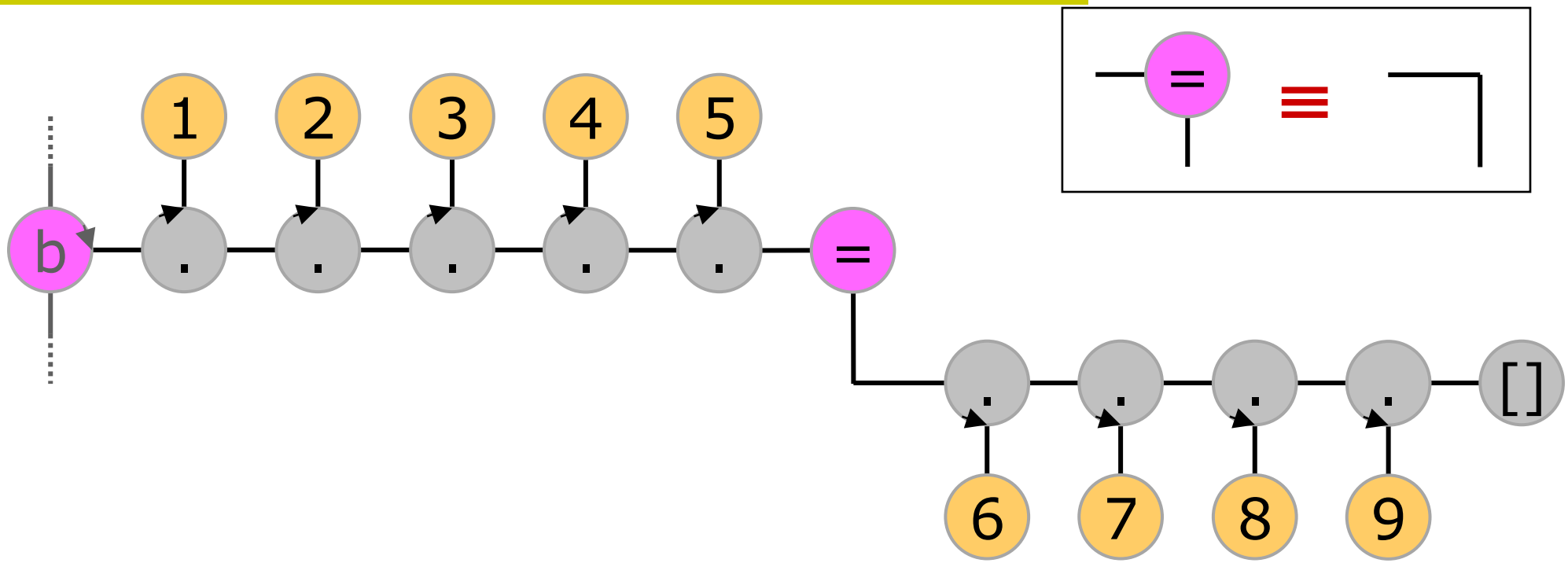


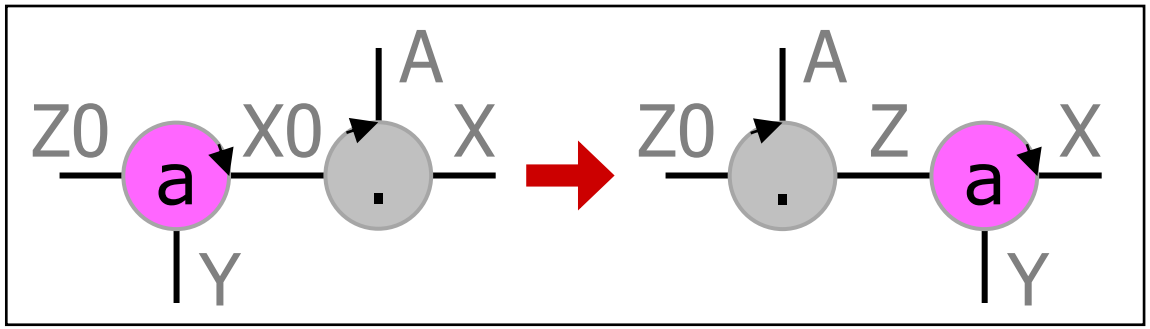


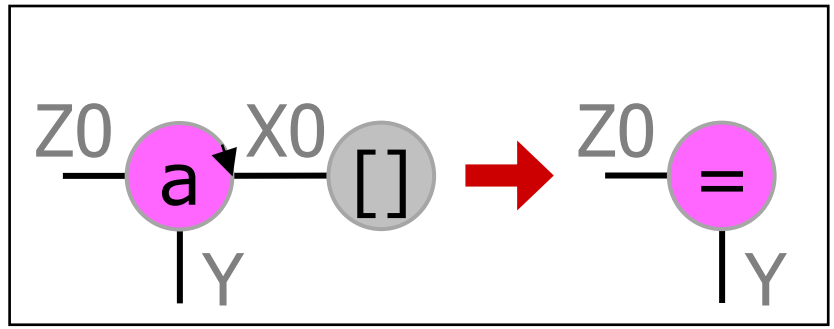

```
a(X0,Y,Z0), '.'(A,X,X0) :- '.'(A,Z,Z0), a(X,Y,Z)
```

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

Demo: List concatenation (*a la* Interaction Nets)







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

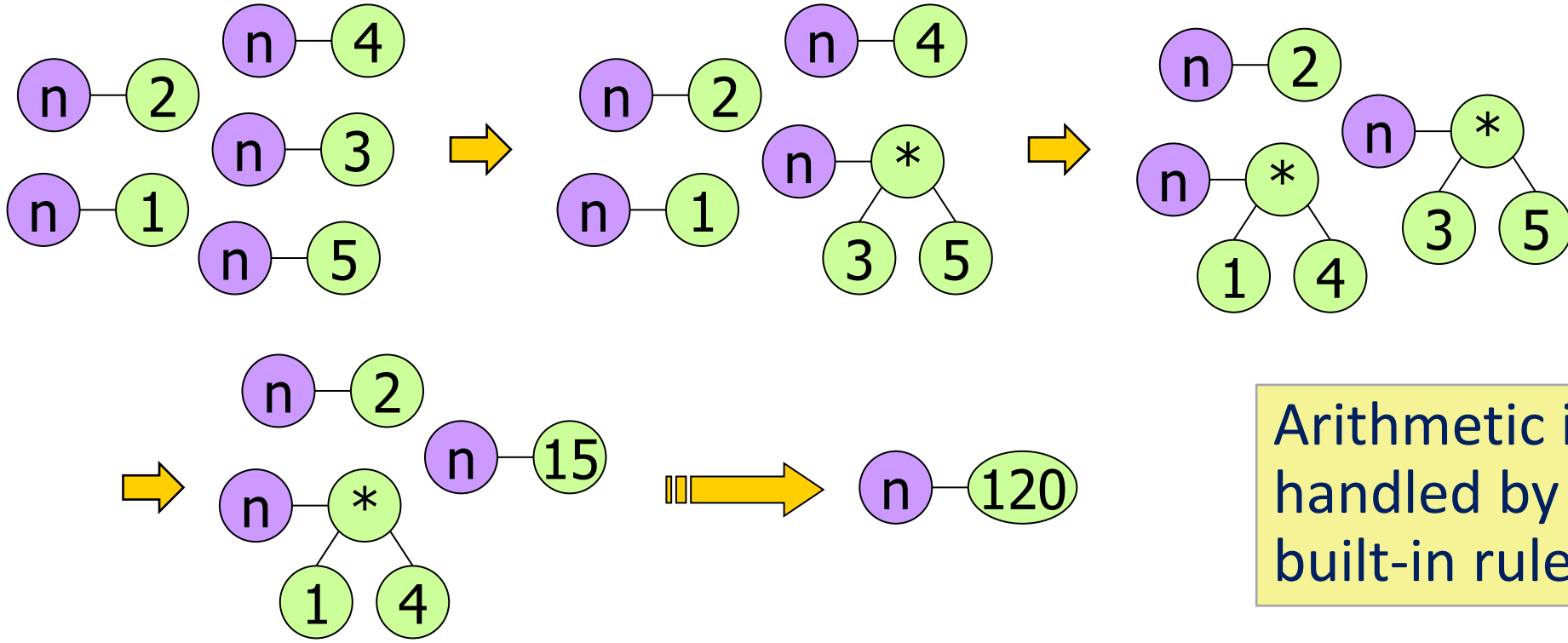
a(X0,Y,Z0), '[]'(X0) :- Y=Z0

Demo: Factorial (multiset rewriting *a la* Gamma)

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

Evaluation order

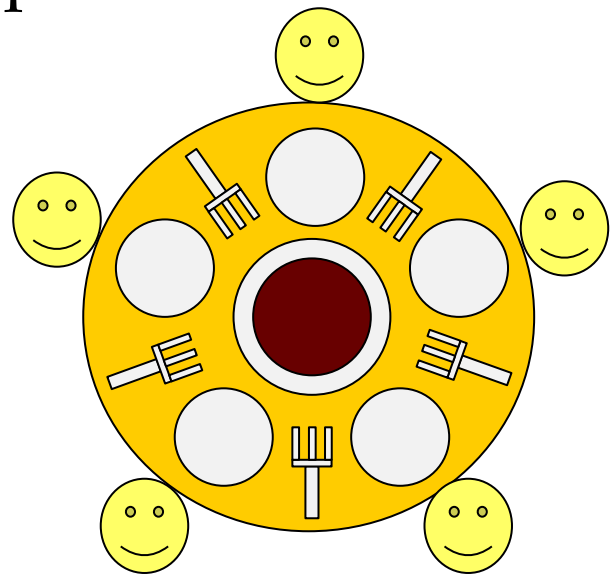
Intentionally left unspecified



Arithmetic is handled by built-in rules

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

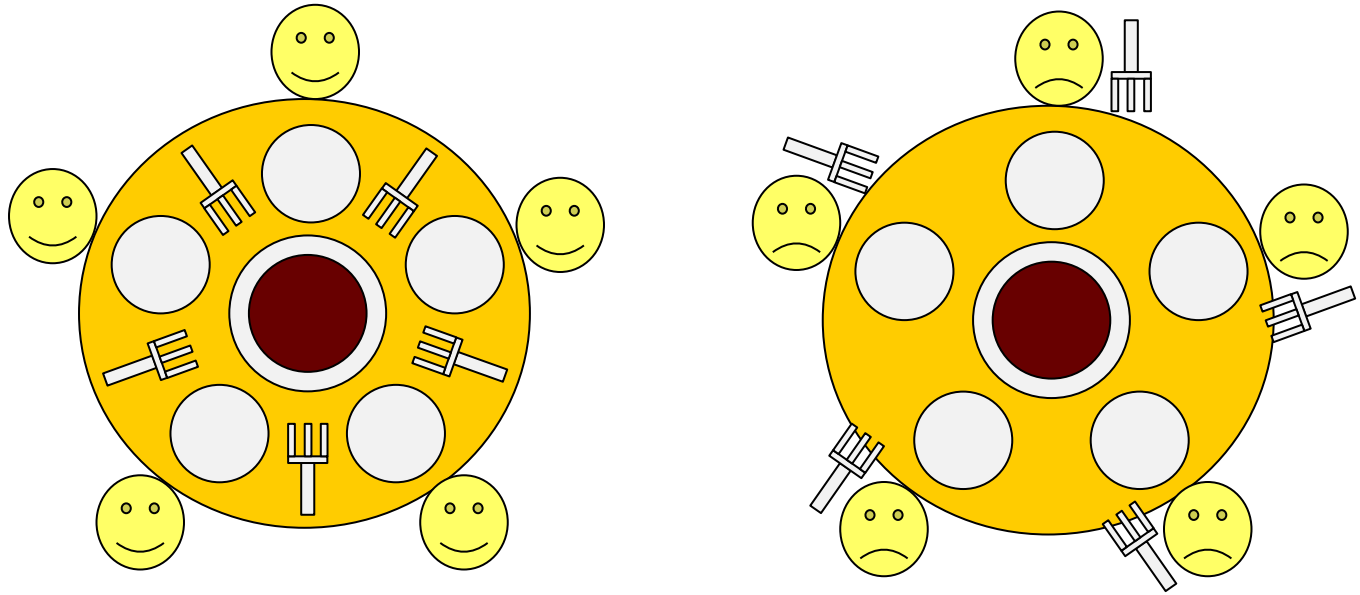
Five philosophers spend their lives thinking and eating. They share a common dining room where there is a circular table surrounded by five chairs, each belonging to one philosopher. In the center of the table there is a large bowl of spaghetti, and the table is laid with five forks. On feeling hungry, 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 would cause deadlock . . .



. . . but a perverse philosopher makes everybody happy!

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

What do we mean by “unifying” ?

Unify various programming concepts, e.g.:

- functions and data
- processes and messages

} (they just react!)

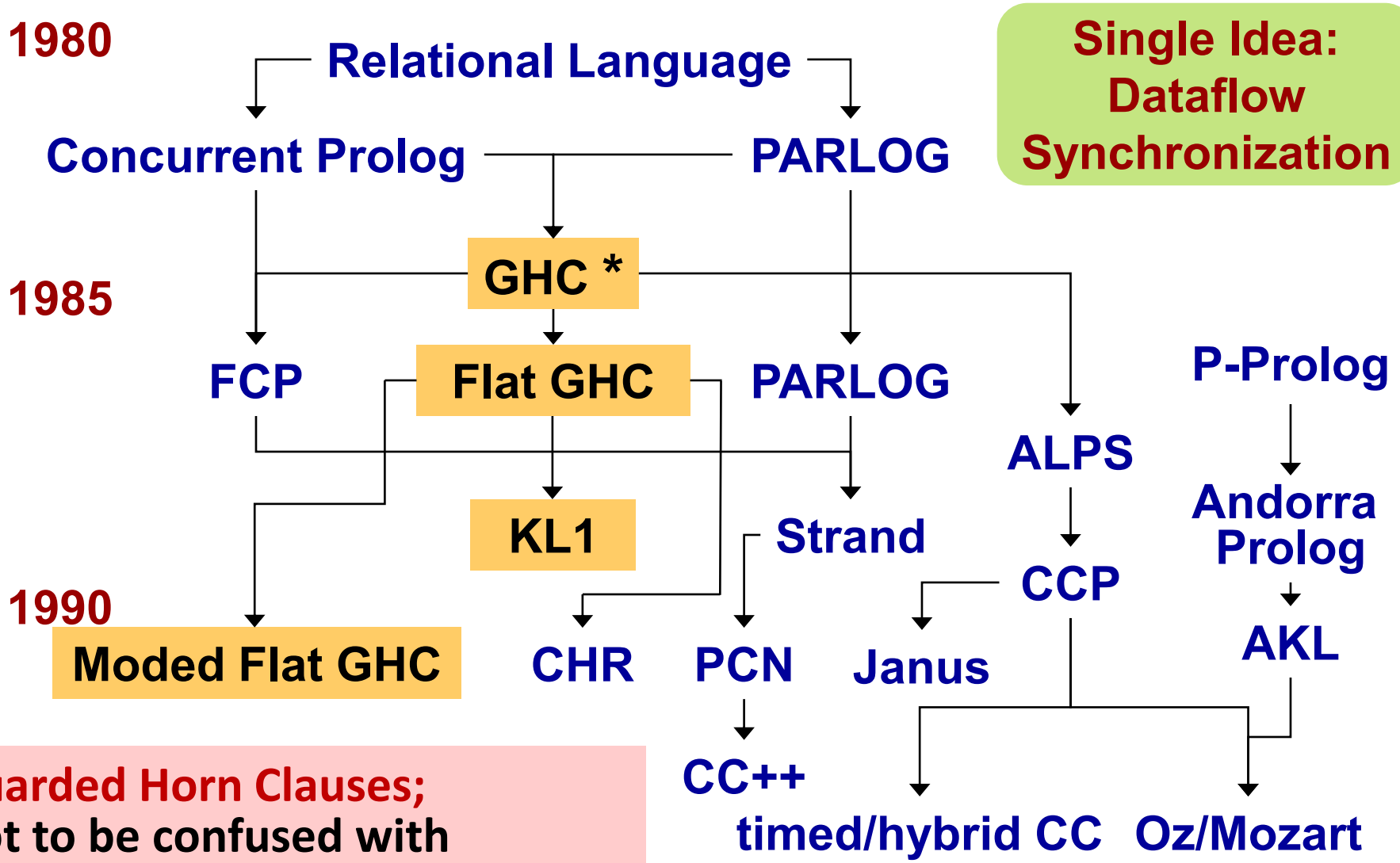
- data structures (passive), process structures (autonomous) and control structures (handled separately from data)

  ➔ “unifying structures”

- synchronous and asynchronous communication

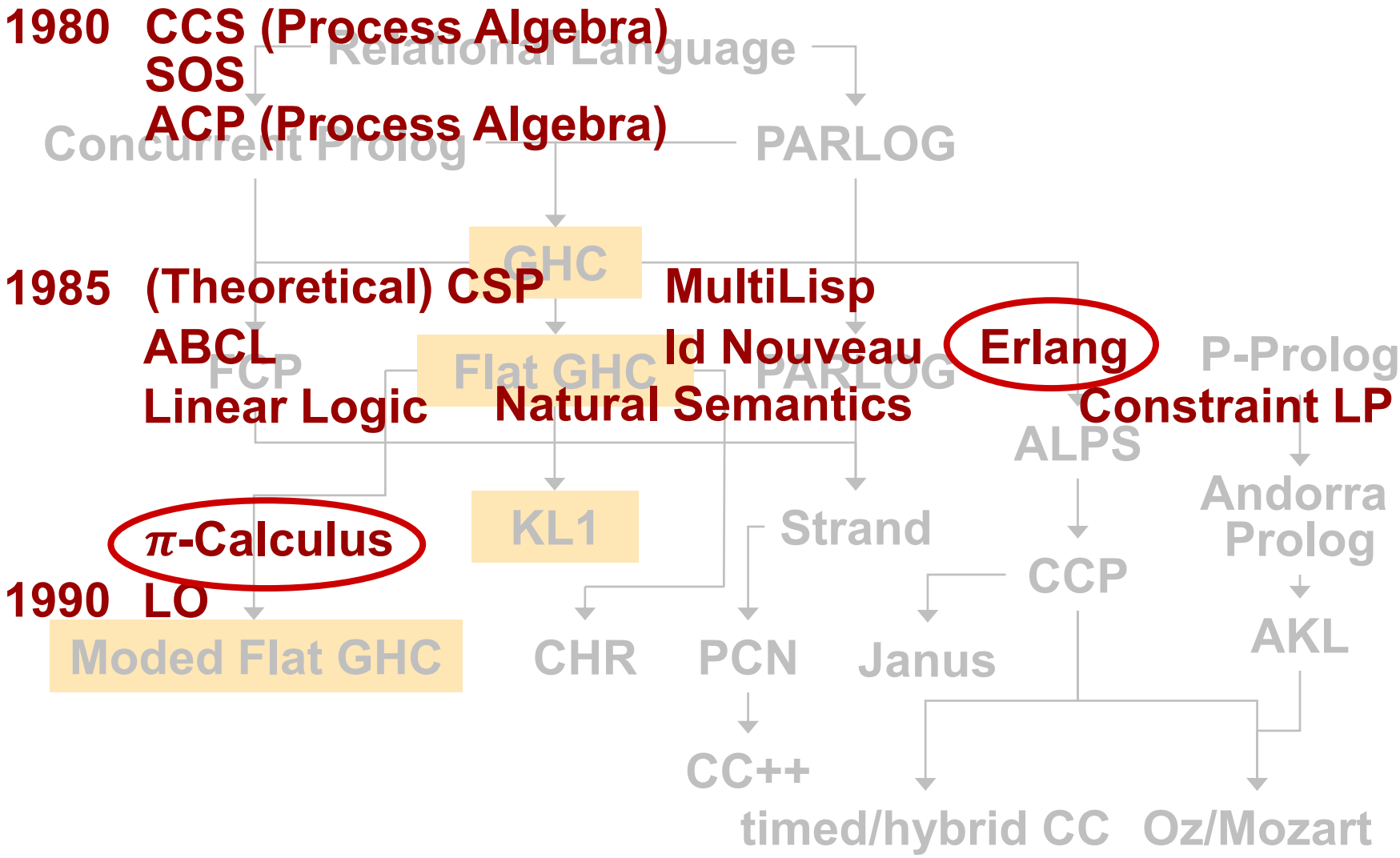
- programming and modeling
- computation and verification

Prehistory: Concurrent Logic/Constraint Programming



* Guarded Horn Clauses;
Not to be confused with
Glasgow Haskell Compiler (1992)

Prehistory overlaid with other models and languages



Concurrent Logic/Constraint Programming

- Topic of my FLOPS 2016 keynote (+ accompanying SCP paper)
- In a nutshell, it achieved *programming with reconfigurable process networks* using two simple ingredients:
  - **write-once channels**
  - **data constructors**
- The functionalities thus realized included:
  - channel passing using channels (prior to π-calculus)
  - messages containing reply boxes
  - channel fusion (prior to fusion calculus)

  demonstrating the power of *"computing with partial information (or non-strict data structures)"*

Evolution into graph rewriting (2002)

- Experiences with concurrent logic programming:

  The same structure (e.g. trees) can be expressed in two ways

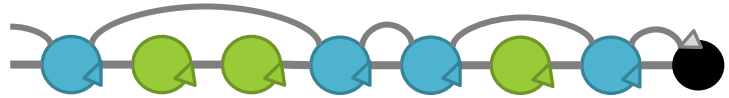
  1. as *data* structures (formed by constructors)
  2. as (more autonomous and general) *process* structures (formed by relations (processes) and variables (channels))

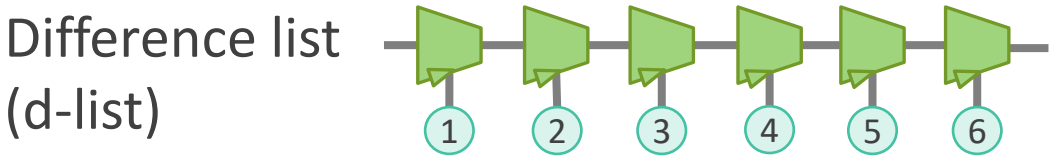
- RQ: Can we do away with constructors (constant/function symbols) and just use relations and variables (which are now never instantiated) for uniformity?

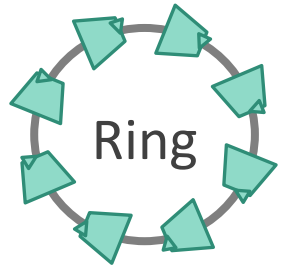
  ➔ Yes, and this is (an instance of) graph rewriting!

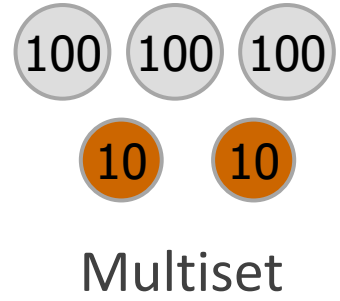
Handling (un)linked structures easily and safely

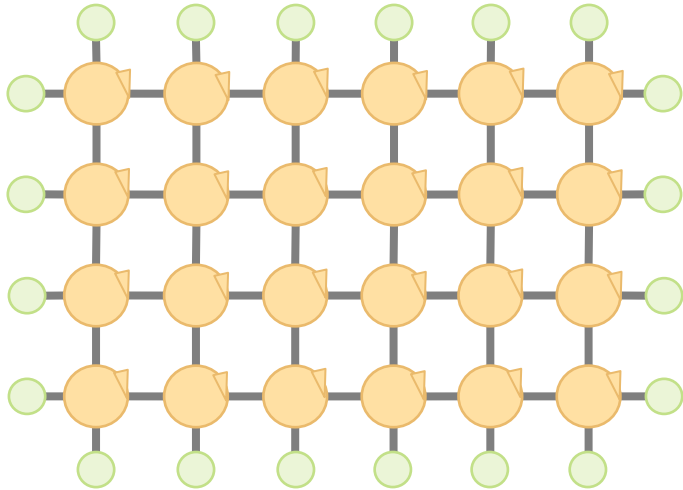
General Graph Structures



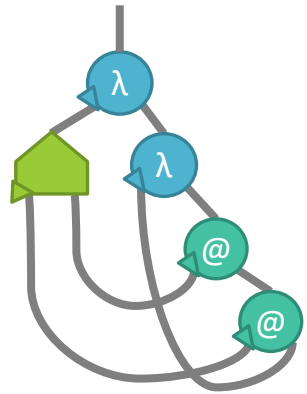








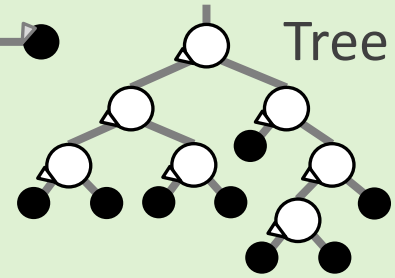
Grid



Lambda term
$\lambda f x.\, f\,(f\,x)$

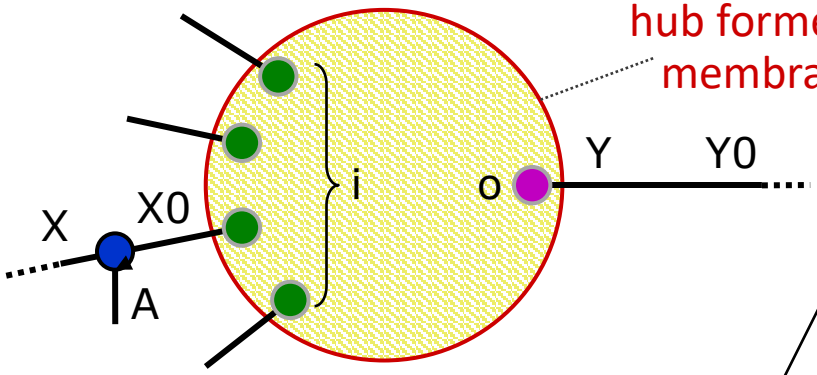
Algebraic Data Types

Linear list

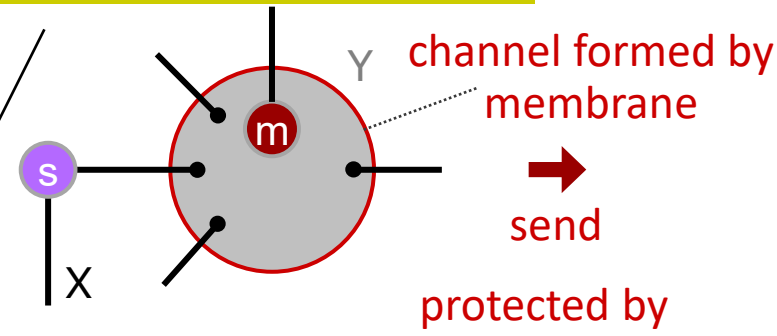


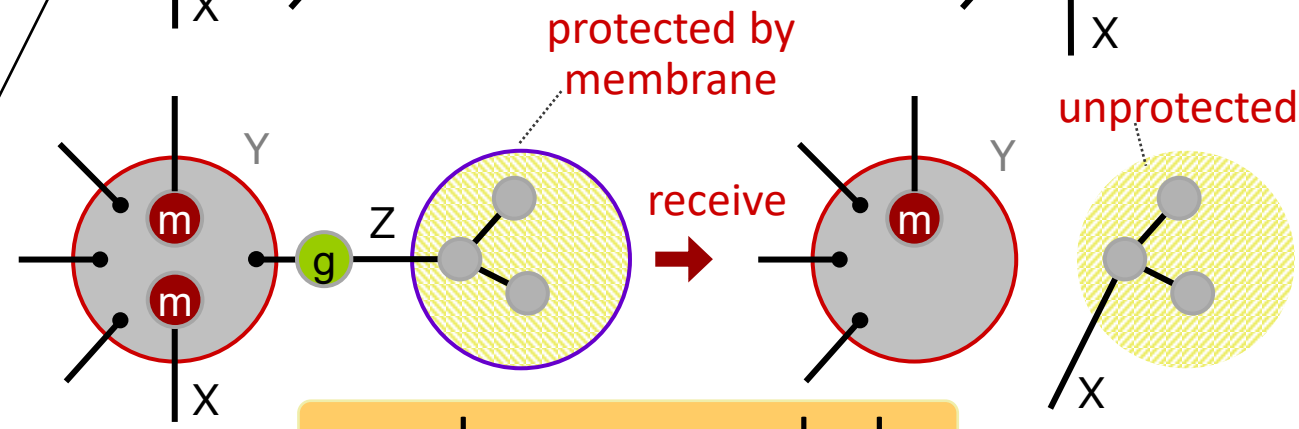
† W. Pugh: Skip lists: A probabilistic alternative to balanced trees, Comm. ACM, 33(6), 1990.

Computation as (hierarchical) graph rewriting

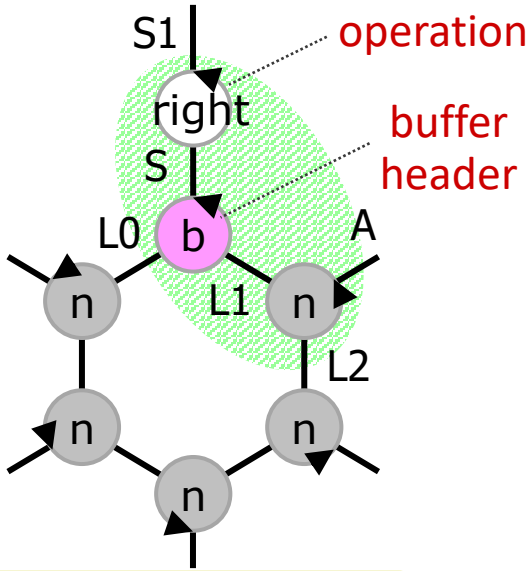


n-to-1 communication

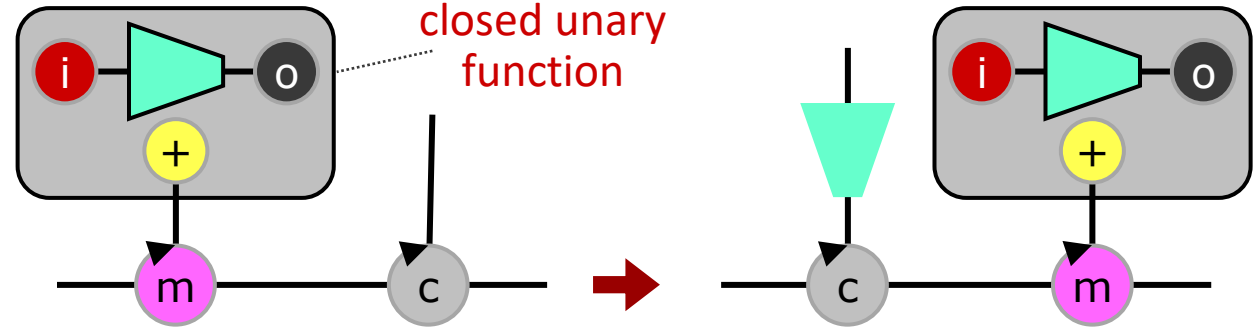




asynchronous π-calculus



cyclic structures



map function

Graph Transformation has a long history (1970s–)

- … based on algebraic/categorial approaches (DPO etc.).
- We seek graph transformation from the PL perspective.
- The primary concern is to establish a *core calculus* with
  - *inductively defined* syntax and
  - *syntax-directed* semantics (= structural operational semantics)

  for graphs and graph transformation, and then have
  - *language constructs for (bigger) graph operations*
- Other PL concerns and interests include *composition*, *abstraction*, *and encoding of other calculi*.

GT from the PL perspective, diagrammatically

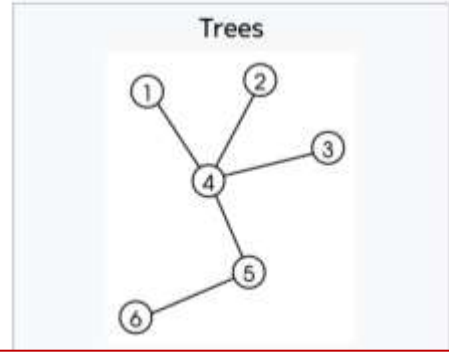
Tree (graph theory)

38 languages

Article Talk | Read Edit View history Tools

From Wikipedia, the free encyclopedia

In graph theory, a **tree** is an undirected graph in which any two vertices are connected by *exactly one* path, or equivalently a connected acyclic undirected graph.[1] A **forest** is an undirected graph in which any two vertices are connected by *at most one* path, or equivalently an acyclic undirected graph, or equivalently



Tree (data structure)

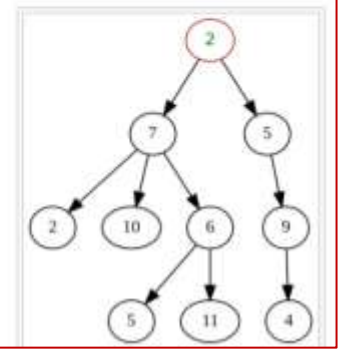
43 languages

Article Talk | Read Edit View history Tools

From Wikipedia, the free encyclopedia

Not to be confused with Trie, a specific type of tree data structure.

In computer science, a **tree** is a widely used abstract data type that represents a hierarchical tree structure with a set of connected nodes. Each node in the tree can be connected to many children (depending on the type of tree), but must be connected to exactly one parent,[1] except for the *root* node, which has no parent (i.e., the root node as the top-most node in the tree hierarchy). These constraints mean



Graph (discrete mathematics)

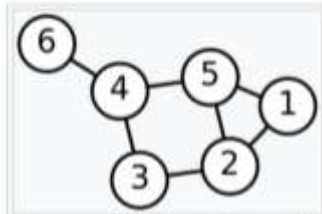
65 languages

Article Talk | Read Edit View history Tools

From Wikipedia, the free encyclopedia

This article is about sets of vertices connected by edges. For graphs of mathematical functions, see Graph of a function. For other uses, see Graph (disambiguation).

In discrete mathematics, and more specifically in graph theory, a **graph** is a structure amounting to a set of objects in which some pairs of the objects are in some sense "related". The objects are represented by abstractions called *vertices* (also called *nodes* or *points*) and each of



A graph with six vertices

?

(Wikipedia entries accessed on July 3, 2024)

GT from the PL perspective, non-solution

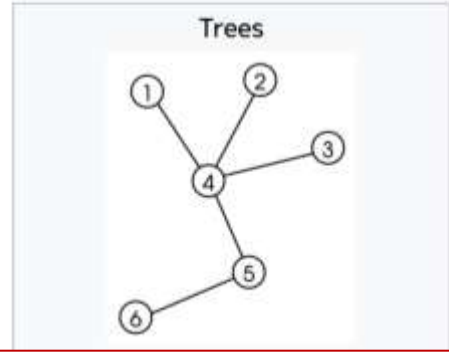
Tree (graph theory)

38 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

In graph theory, a **tree** is an undirected graph in which any two vertices are connected by *exactly one* path, or equivalently a connected acyclic undirected graph.[1] A **forest** is an undirected graph in which any two vertices are connected by *at most one* path, or equivalently an acyclic undirected graph, or equivalently



Tree (data structure)

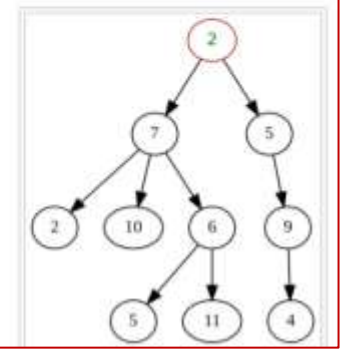
43 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

Not to be confused with Trie, a specific type of tree data structure.

In computer science, a **tree** is a widely used abstract data type that represents a hierarchical tree structure with a set of connected nodes. Each node in the tree can be connected to many children (depending on the type of tree), but must be connected to exactly one parent,[1] except for the *root* node, which has no parent (i.e., the root node as the top-most node in the tree hierarchy). These constraints mean



Graph (discrete mathematics)

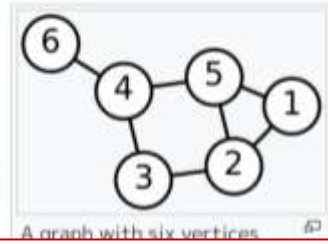
65 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

This article is about sets of vertices connected by edges. For graphs of mathematical functions, see Graph of a function. For other uses, see Graph (disambiguation).

In discrete mathematics, and more specifically in graph theory, a **graph** is a structure amounting to a set of objects in which some pairs of the objects are in some sense "related". The objects are represented by abstractions called *vertices* (also called *nodes* or *points*) and each of



A graph with six vertices

Graph (abstract data type)

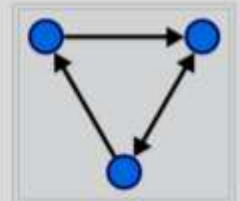
25 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

In computer science, a **graph** is an abstract data type that is meant to implement the undirected graph and directed graph concepts from the field of graph theory within mathematics.

A graph data structure consists of a finite (and possibly mutable) set of *vertices* (also called *nodes*


A directed graph

(Wikipedia entries accessed on July 3, 2024)

Graph Transformation in different guises

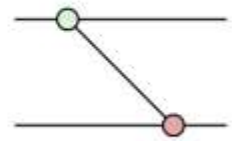
Home Tutorial Publications The ZX Seminar Accessibility Map PyZX Demo

The ZX-calculus

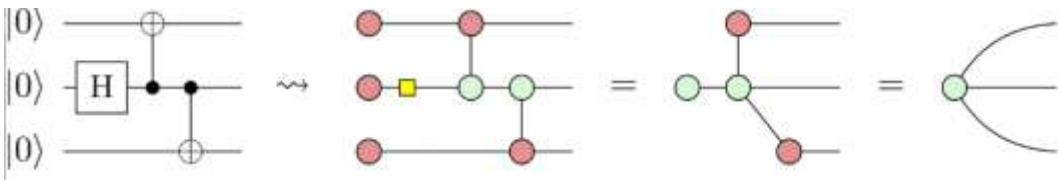
Quantum computing

The ZX-calculus is a graphical language that goes beyond circuit diagrams. It 'splits the atom' of well-known quantum logic gates to reveal the compositional structure inside. The calculus works by generalising the ideas of Z and X operations, allowing us to break out of the circuit model while maintaining soundness of reasoning. In doing so we can show properties of circuits, entanglement states, and protocols, in a visually succinct but logically complete manner.

The ZX-calculus is forging the next generation of quantum software. Using the calculus gives optimisation strategies that performs state-of-the-art T-count reduction (an important metric for fault-tolerant computing) and gate compilation. The generators of the calculus correspond closely to the basic operations of lattice surgery in the surface code, giving a visual design and verification language for these codes; and ZX has also been used to discover novel error correction procedures. It comes with a scalable notation capable of representing repeated structures at arbitrary qubit scales. The calculus also acts in the crucial role of an intermediate representation in a new commercial quantum compiler.



https://zxcalculus.com/



String Diagram Rewrite Theory I: Rewriting with Frobenius Structure

FILIPPO BONCHI and FABIO GADDUCCI, University of Pisa
ALEKS KISSINGER, University of Oxford
PAWEL SOBOCINSKI, Tallinn University of Tech
FABIO ZANASI, University College London

Category theory

Transformation Theory and Applications Home Seminars How to participate Team GReTA-ExACT

Graph Rewriting as a Foundation for Science and Technology (and the Universe)

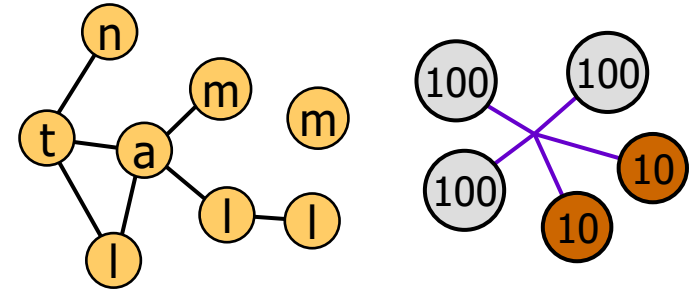
Stephen Wolfram

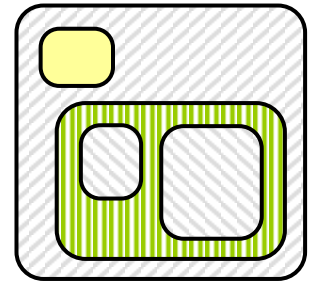
Follow

Every-thing!

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, ...





RQ: Can we have a concise PL that allows us to represent and manipulate them simultaneously and in a direct, safe manner? (cf. Bigraphs (Milner))

append in LMNtal

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

- Constructors ('.' and '[]') are in relational form, but LMNtal provides a *term (or functional) notation*:

Expand (E9) Embed

```
    '[]'(X0)  ≡  X0=Y, '[]'(Y)     ≡  X0='[]'
'.'(A,X,X0)   ≡  X0=Y, '.'(A,X,Y)  ≡  X0='.'(A,X)
```

```
append(X0,Y,Z0), X0='.'(A,X) :- Z0='.'(A,Z), append(X,Y,Z)
```

```
Z0 = append('.'(A,X),Y)  :-  Z0 = '.'(A,append(X,Y))
```

Models/languages with multisets and *symmetric join*

- (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 Machine, reflexive CHAM, Join Calculus
- Gamma model
- Maude
- Constraint Handling Rules (CHR)
- Mobile ambients
- P-system, membrane computing
- Amorphous computing
- Bigraphical Reactive Systems

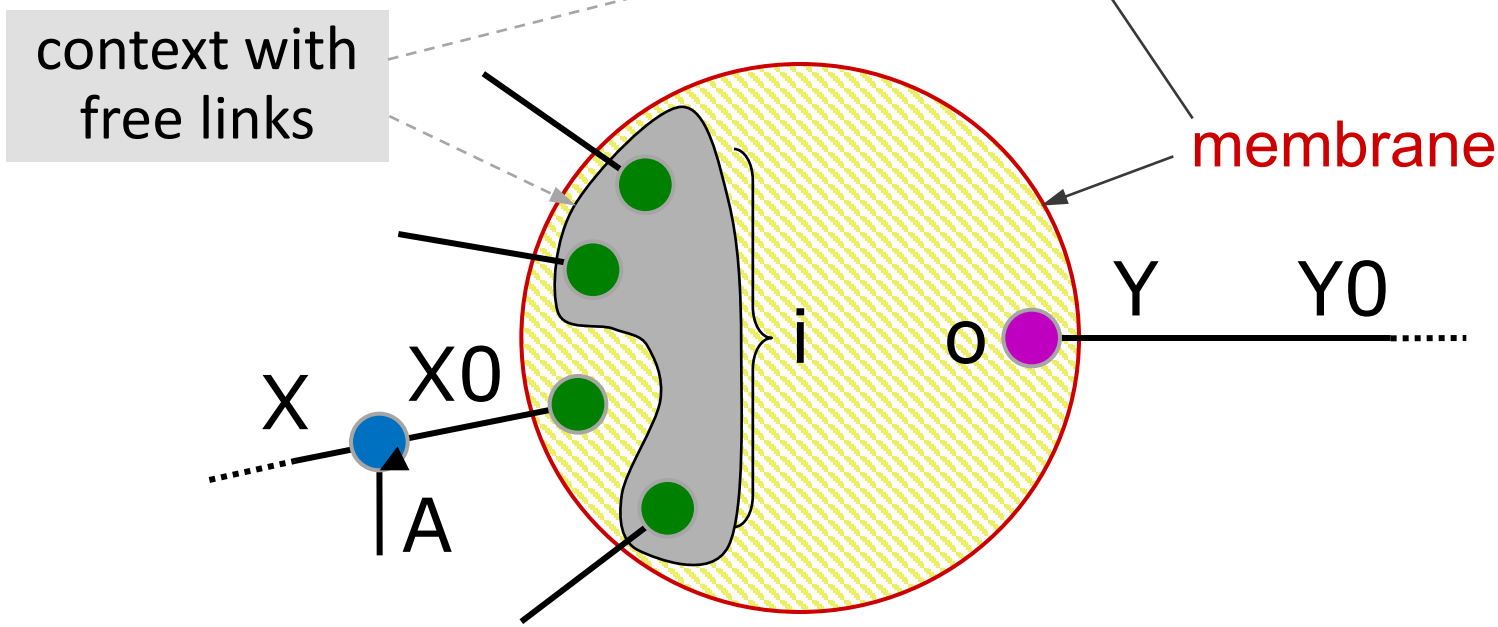
Models/languages with *membranes + hierarchies*

- (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 Machine, reflexive CHAM, Join Calculus
- Gamma model
- Maude
- Constraint Handling Rules
- Mobile ambients
- P-system, membrane computing
- Amorphous computing
- Bigraphical Reactive Systems
- Statecharts
- Seal calculus
- Kell calculus
- Brane calculi
- κ calculus

* : some versions feature hierarchies

Example: Many-to-one channel communication

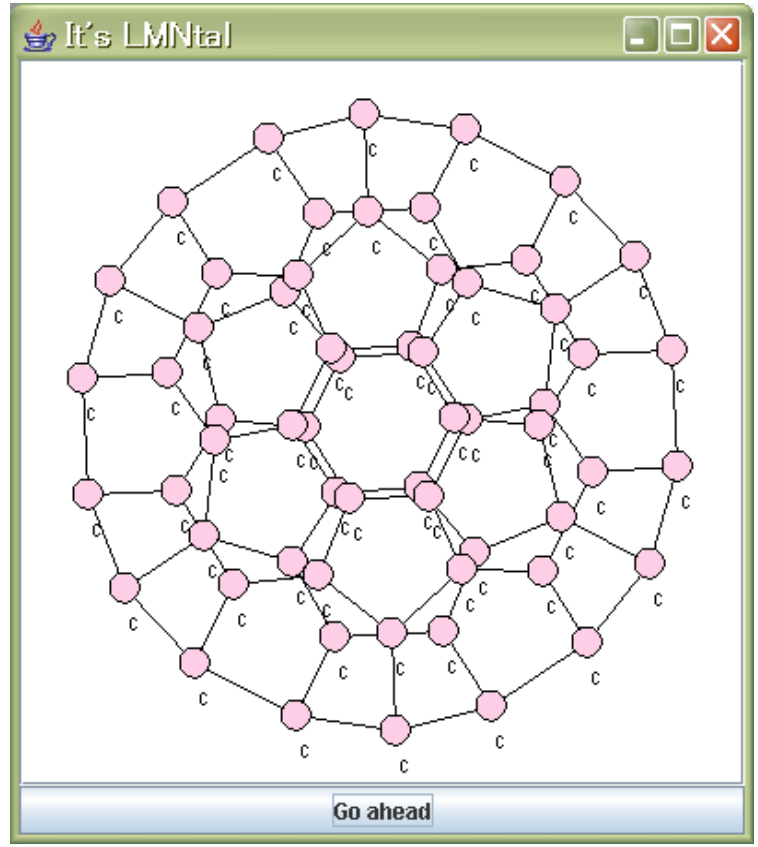
```
c(A,X,X0), { i(X0), o(Y0), $p } :-
           { i(X), o(Y), $p }, c(A,Y,Y0)
```



- The number of free links of { } remain unchanged

Demo: 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).
```



Elements of LMNtal

Hypergraphs with nonlinear links will be discussed later

1 & 2. Labelled **nodes** (called *atoms*) with ordered **links**

- Origin: atomic formula $p(t_1, \ldots, t_n)$ in first-order logic
- Each atom has a fixed arity (degree), and its links are totally ordered (cf. nodes in graph theory)
  - *a.k.a.* "(node of a) *port graph*"
- Links are linear, *zero-assignment logical* variables
  - linear = occurring twice (1-to-1 communication)
  - zero-assignment = not instantiated to terms
  - logical (*a.k.a.* immutable) = link identity changes after message sending (cf. π-calculus)
  - not directed (like chemical bonds)

Elements of LMNtal

3. **Membranes** (to represent *first-class multisets*)

- Not many languages feature multisets (or forests) as *first-class* citizens
- Used both as data structures and control structures:
  - representing *records* (*a.k.a.* feature structures, KVS)
  - representing *graph nodes with variable degrees*
  - creating and managing *fresh local names*
  - *localization and control* of computation (cf. ambients, join calculus, Unix processes, etc.)
  - *termination detection*

Elements of LMNtal

4. **Rewrite rules**

- Can be placed inside a membrane for *local reaction*
- Key design issue: proper handling of *free links* (*a.k.a. open/half/dangling edges, loose ends, links with boundary/exterior vertices*, etc.)
  - Recall: LMNtal handles open graphs (*a.k.a. semigraphs*)
  - Handling of contexts is the key
    - matching (LHS)
    - duplication / migration / deletion (RHS)

Syntax: Preliminaries

- Two presupposed syntactic categories:
  - *X* : link names (linear & local)
    - In concrete syntax, start with capital letters
  - *p* : atom names (nonlinear & global)
    - Works as *node labels*
    - In concrete syntax, use identifiers different from links (e.g., `cons`, `314`, `'+'`, `"string"`)
- Atom names are uninterpreted except for `'='` (a connector)

Syntax of LMNtal processes

historical terminology

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 conditions:

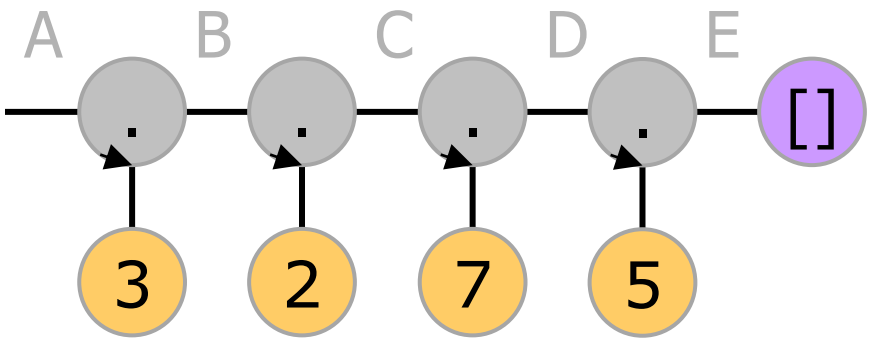
- Each (non-hyper)link name in P occurs at most twice
- Each link name in a rule occurs exactly twice.
  - Free link of P = link occurring only once
  - P is closed = has no free links

Syntax of LMNtal process templates

Not in Flat LMNtal

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

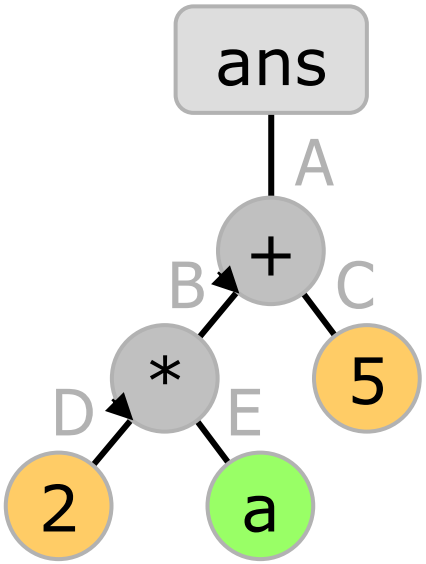
Lists, trees, bags (*cells*)




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

– or –

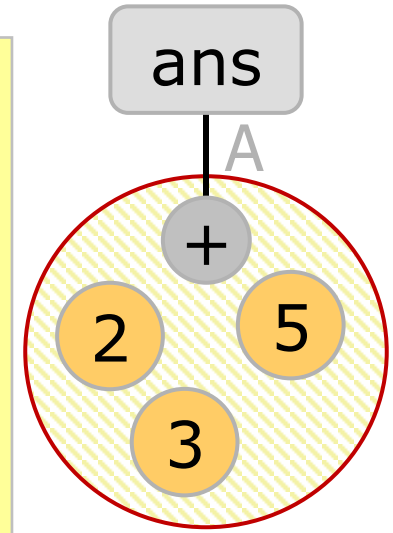
```
A =  '.'(3, '.'(2, '.'(7, '.'(5,'[]'))))
A = [3 | [2 | [7 | [5 | []]]]
A = [3, 2, 7, 5]
```




```
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})
ans={2, 3, 5}
```

Term notation

remove the final arg and embed the rest

'.'(A1,X1,X0), '.'(A2,X2,X1), '.'(A3,X3,X2), '[]'(X3)

≡ '.'(A1, '.'(A2, '.'(A3, '[]')), X0)

expand using =

≡ X0 = Y, '.'(A1, '.'(A2, '.'(A3, '[]')),Y)

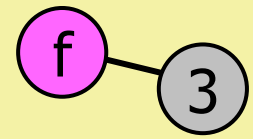
fold again

≡ X0 = '.'(A1, '.'(A2, '.'(A3, '[]')))

list notation

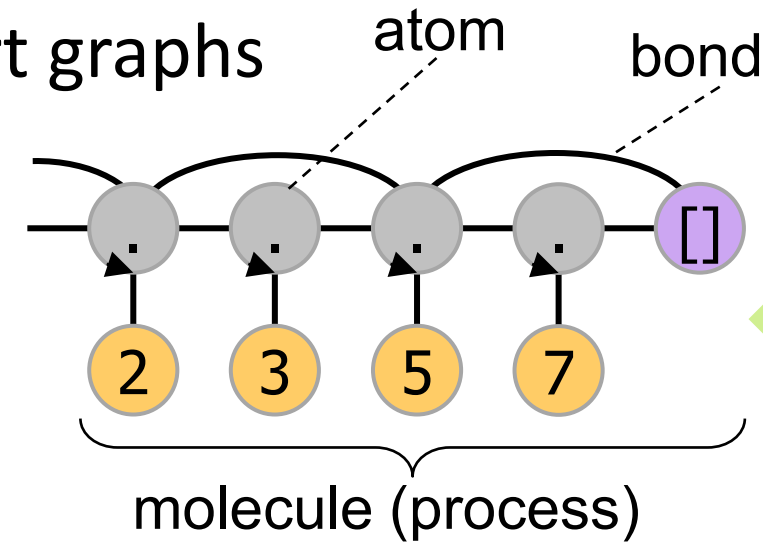
≡ X0 = [A1, A2, A3]

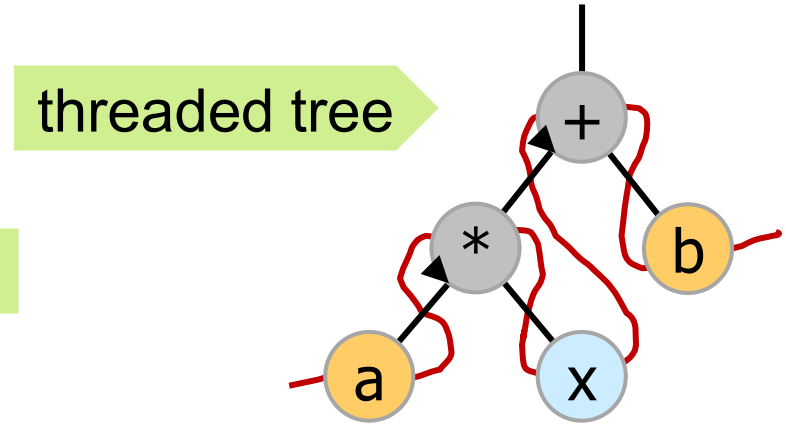
f(X),3(X) ≡ f(3) ≡ 3(f) ≡ f = 3 ≡ 3 = f ≡



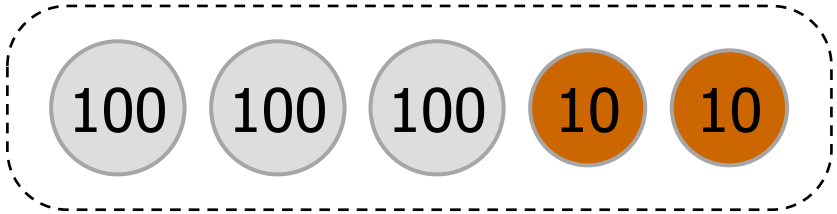
(Port) graphs and multisets

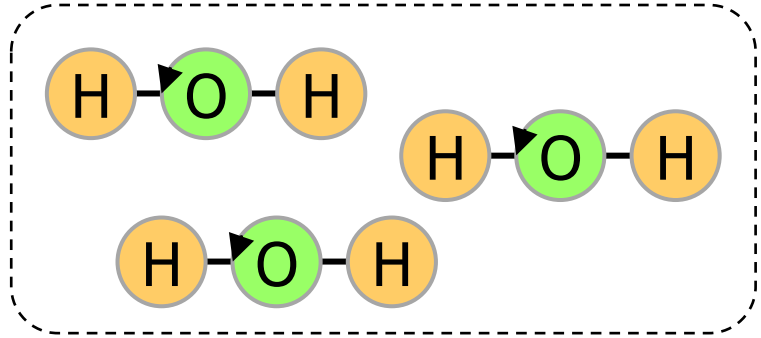
- Port graphs





- Multisets

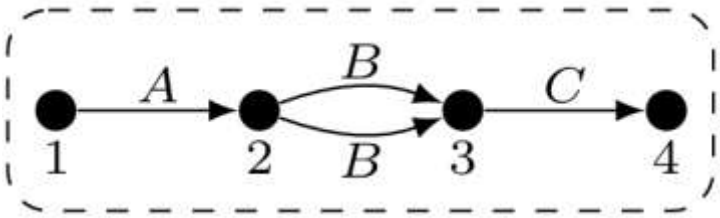




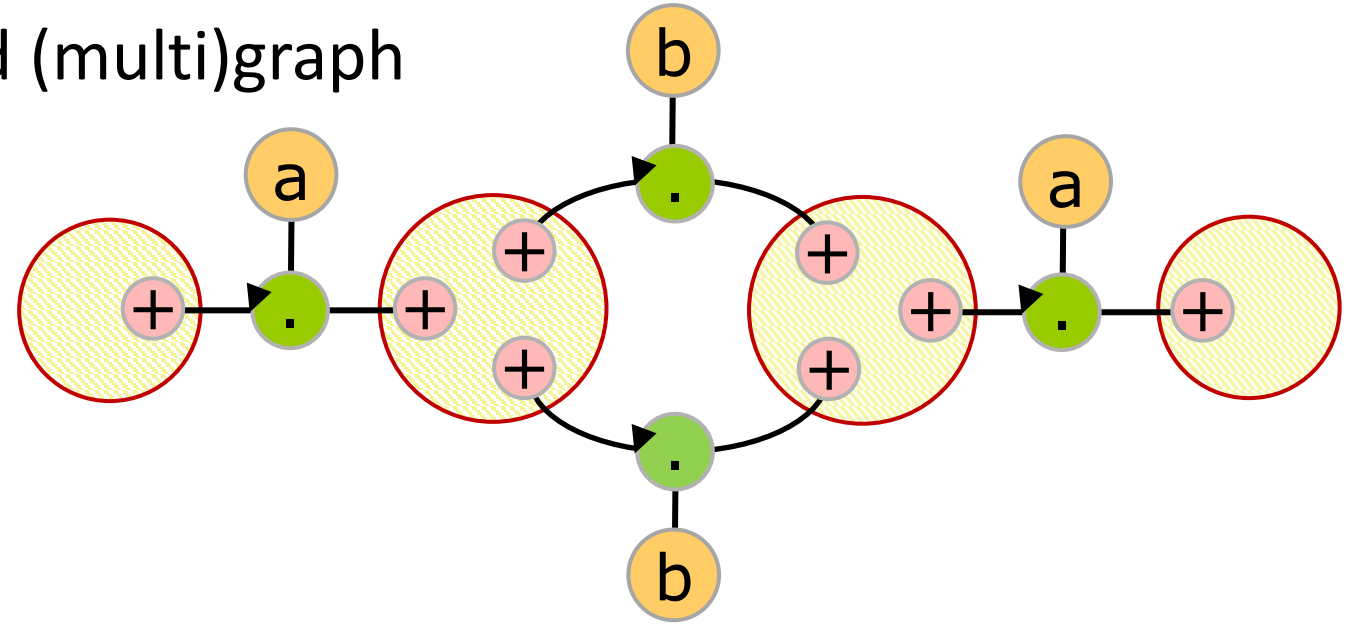
... are graphs with “less” edges (another important direction!)

How about “standard” graphs?

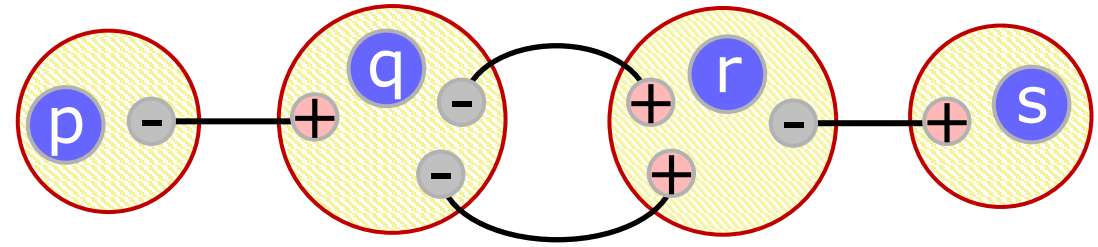
- Edge-labeled directed (multi)graph



König et al., LNCS 10800



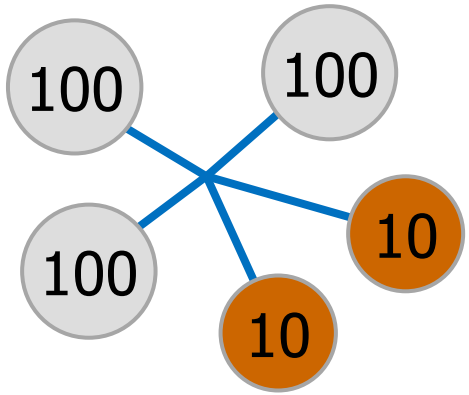
- Node-labeled directed (multi)graph
  (with several other alternatives depending on applications)



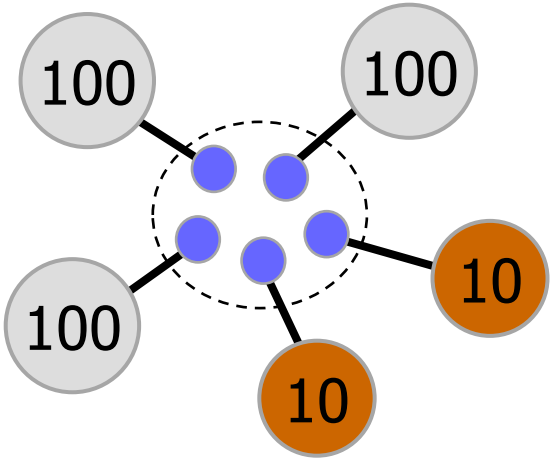
- Attributed graphs in a similar manner

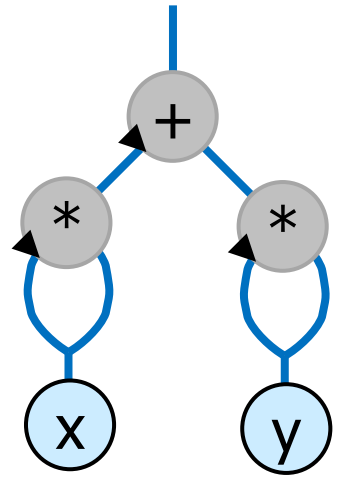
Hypergraph extension (HyperLMNtal)

- Hyperlinks (blue or curved) expressed differently from ordinary links

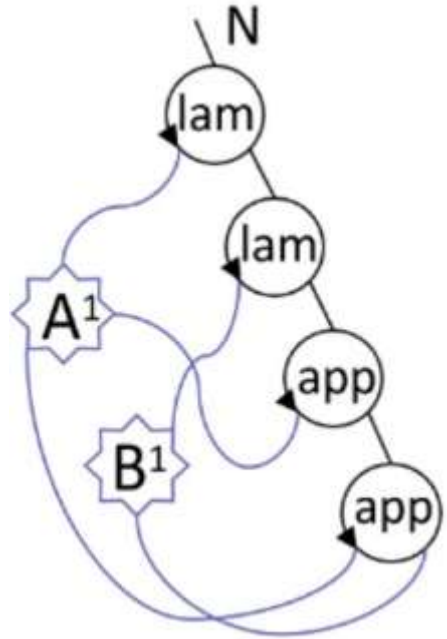


multiset





term graph



$\lambda f.\lambda x.f(fx)$

A hyperlink can be regarded as
a collection of *unary atoms with*
a private name (and an optional attribute)

Representing algorithms symmetrically

- Euclid's algorithm

```
m=20, n=8.
m=$x, n=$y :- $x>$y | m=$x-$y, n=$y.
m=$x, n=$y :- $x<$y | m=$x, n=$y-$x.
```

  - The algorithm uses no procedure/function names (can be considered as reaction between data)

- LMNtal allows us to give identical names to two or more data (useful for *symmetric* algorithms)

```
n=20, n=8.
n=$x, n=$y :- $x>$y | n=$x-$y, n=$y.
```

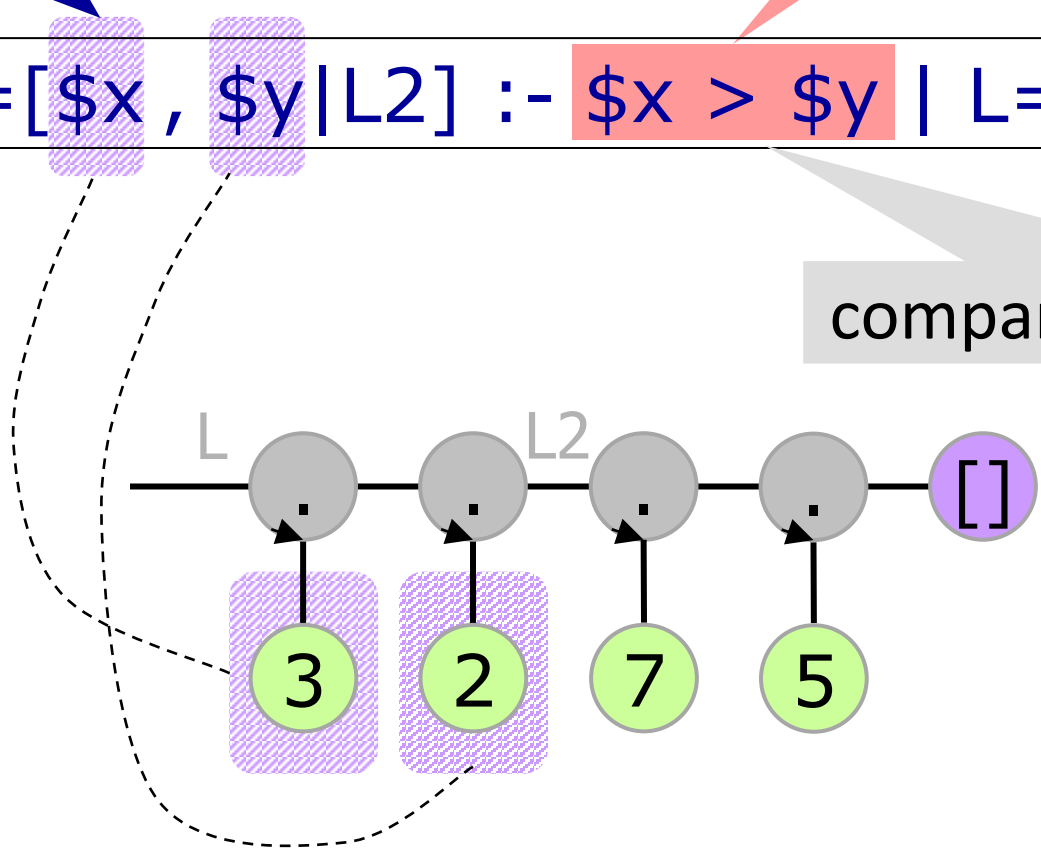
Demo: Bubblesort (without sequential control)

typed process context

guard

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

compare and swap if $x > $y



The LHS may match the middle of a list.

Structural congruence $\equiv$

To allow graph interpretation of terms

multisets

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

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

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

structural

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

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 exactly one of X and Y is a free link of P

Structural congruence in pictures

connectors

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

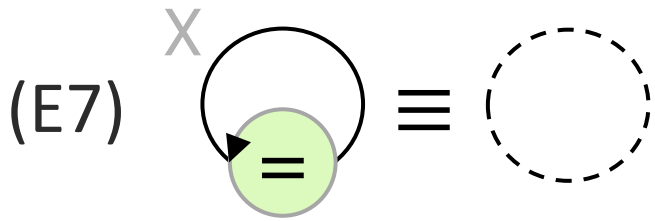
(E8) $X{=}Y \equiv Y{=}X$

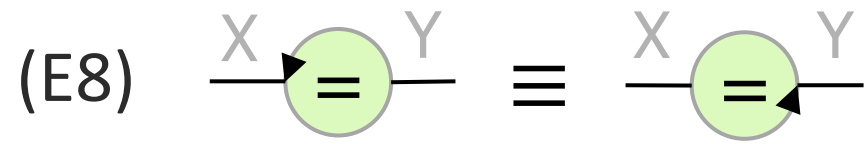
(E9) $X{=}Y,\ P \equiv P[Y/X]$

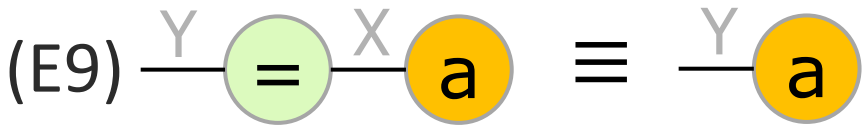
if P is an atom and X is a free link of P

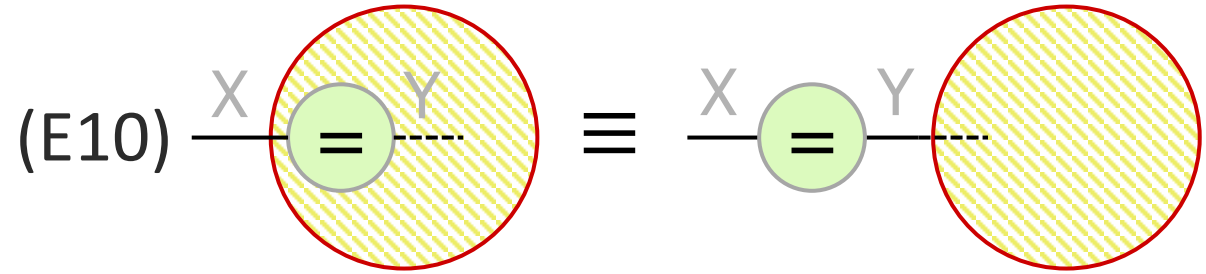
(E10) $\{X{=}Y,\ P\} \equiv \{P\},\ X{=}Y$

if exactly one of X and Y is a free link of P









Structural congruence: Notes

multisets

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

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

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

Renaming is admissible!

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

implied by "congruence"

(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) $X{=}Y \equiv Y{=}X$

Symmetry is admissible!

(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 exactly one of X and Y is 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) $$\frac{Q \equiv P \quad P \to P' \quad P' \equiv Q'}{Q \to Q'}$$

structural

connectors

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

θ is to instantiate contexts and bundles; no need to handle links

Reduction semantics

(R1) $\dfrac{P \to P'}{P, Q \to P', Q}$ (R2) $\dfrac{P \to P'}{\{P\} \to \{P'\}}$

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

structural

connectors

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

if X and Y are free links of $(X{=}$

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

if X and Y are free links of P

(R4)(R5) could be "downgraded" to standard library

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

θ is to instantiate contexts and bundles; no need to handle links

Reduction semantics: Notes

- Can `p(A,A)` be reduced using `p(X,Y) :- q(Y,X)` ?
  - The LHS of the rule can't be α-converted to `p(A,A)`
  - However, by structural congruence, `p(A,A)` can be reduced to `q(A,A)` (using fresh links `X, Y`) as:

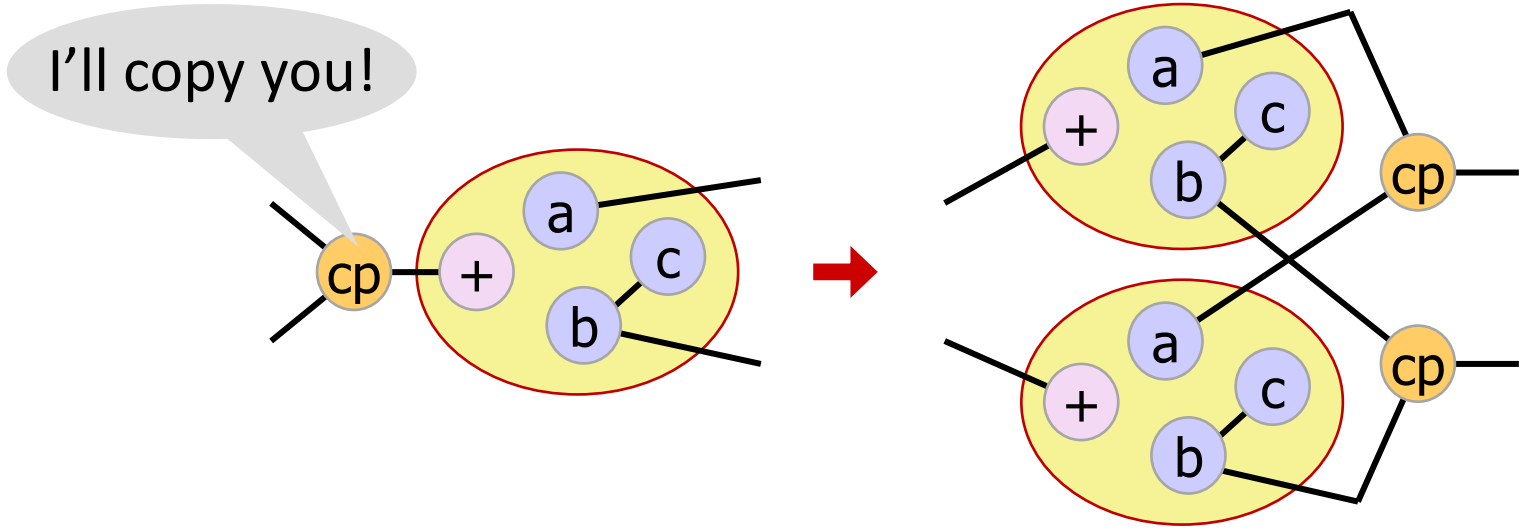
```
    p(A,A)
≡ p(X,Y), X=A, Y=A
→ q(Y,X), X=A, Y=A
≡ q(A,A)
```

Process contexts

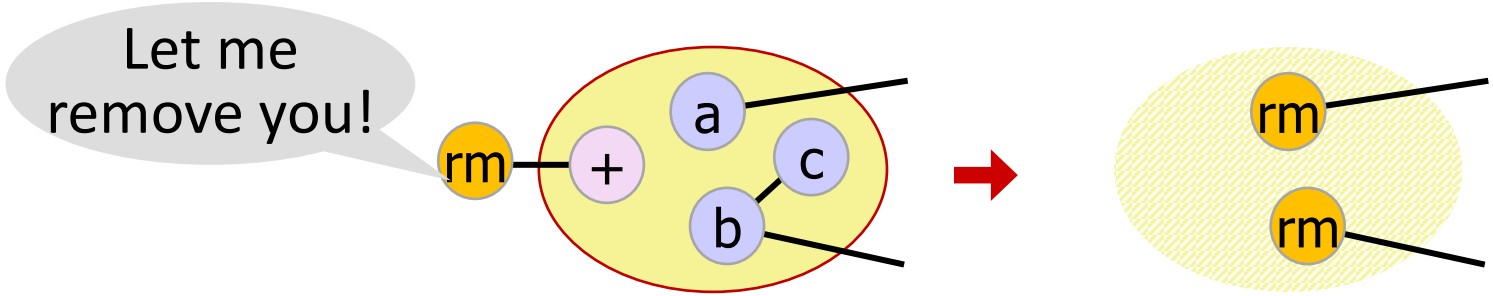
- *Process/graph-level variables (or wildcards)* for :
  - migration (across membranes)
  - cloning and deletion (without tedious graph traversal)
- Two families (according to hierarchy and connectivity):
  1. *Untyped* — to capture "the rest of the nodes" (= "context") of the enclosing membrane
  2. *Typed* — to capture a graph with a specific "shape" (= "type"), which is either
     a. pre-defined (int, unary, ground, …) or
     b. user-defined (by a grammar specified by "typedef")

Cloning and deletion using *untyped* contexts

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

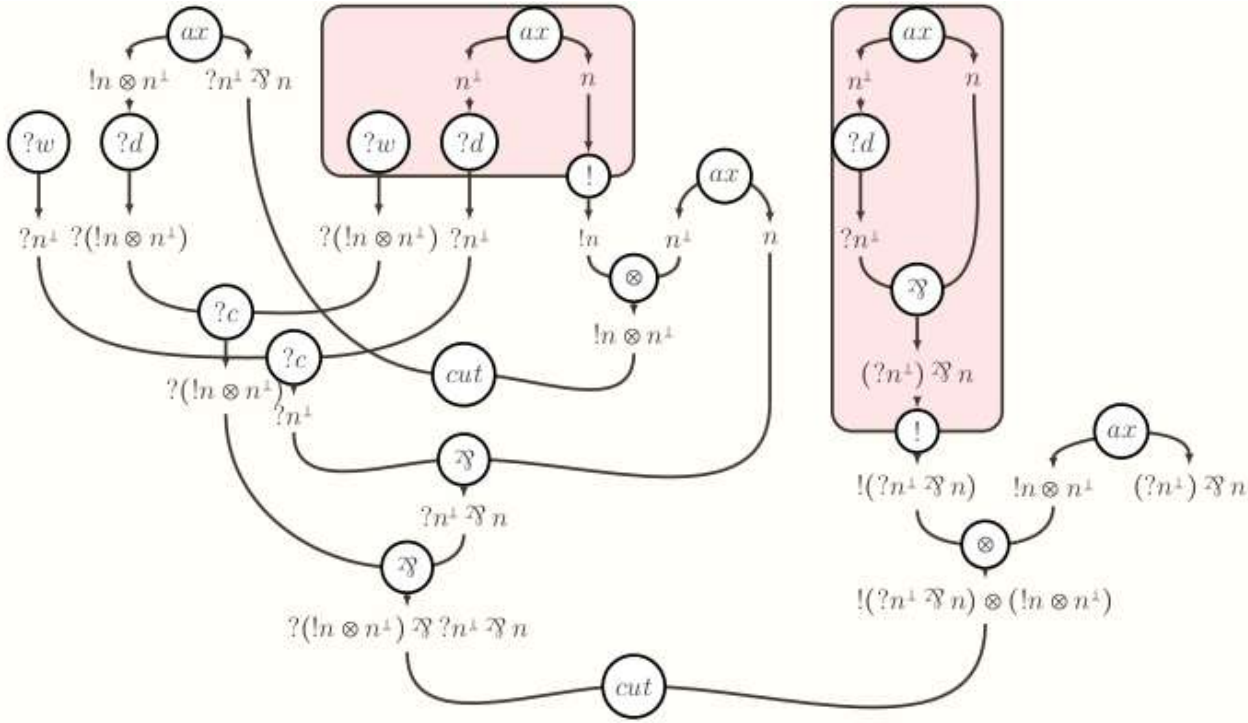


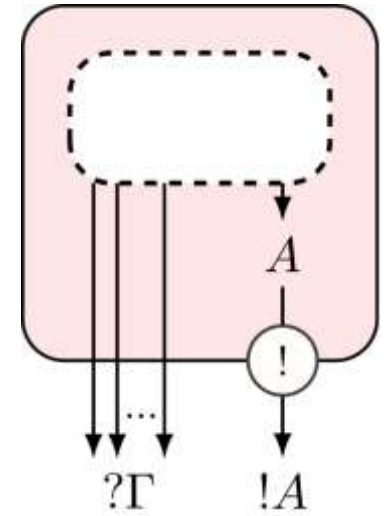
- `rm(S), {+S, $a[|*X]]} :- rm(*X)`



Affinity with Proof-Net cut elimination [PADL 2025]

- MELL (Multiplicative Exponential Linear Logic) (Girard)
  = Multiplicative Linear Logic (MLL) + Exponential operators !, ?
- Proof Net = graph representation of a sequent-calculus proof

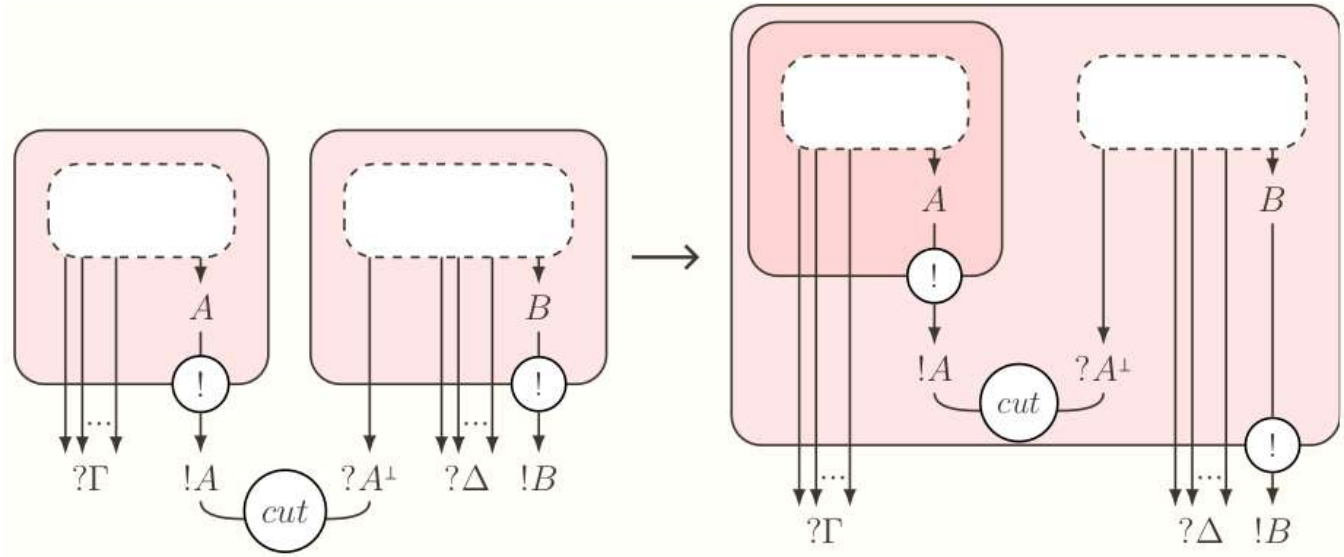


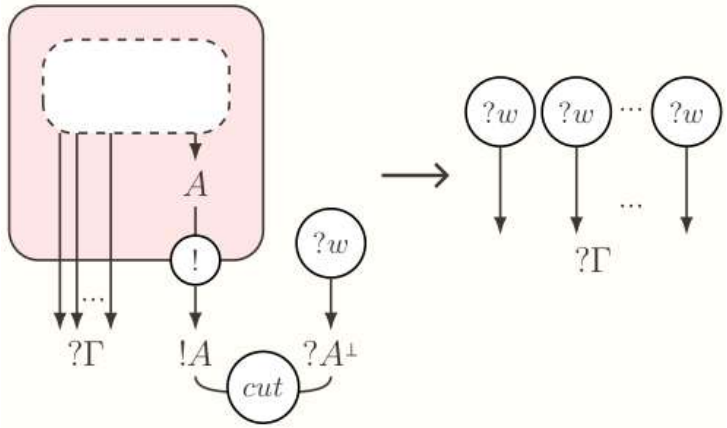


Promotion box for the !-rule to protect the inner items

Affinity with Proof-Net cut elimination [PADL 2025]

- Cut elimination translates to proof net reduction
- Examples:






```
cut_elimination_nested@@
{'!'(A,B),$p1[A|*X]},{$p2[C|*X]},cut{+(B),+(C)}
:- {{'!'(A,B),$p1[A|*X]},$p2[C|*X],cut{+(A),+(B)}}.
```

```
cut_elimination_weakening@@
{'!'(A,B),$p[A|*X]},'?w'(C), cut{+(B),+(C)}
:- nlmem.kill({trash(A),$p[A|*X]},'?w').
```

Typed process contexts

- *Graph-level variables (wildcards)* matching graphs with specific "shapes"
  - the shape is specified in a guard:

```
p(L), $n[L] :- int($n) | …
or    p($n) :- int($n) | …
```

type constraint telling that $n is a unary integer atom

  - considered as *rule schemata*

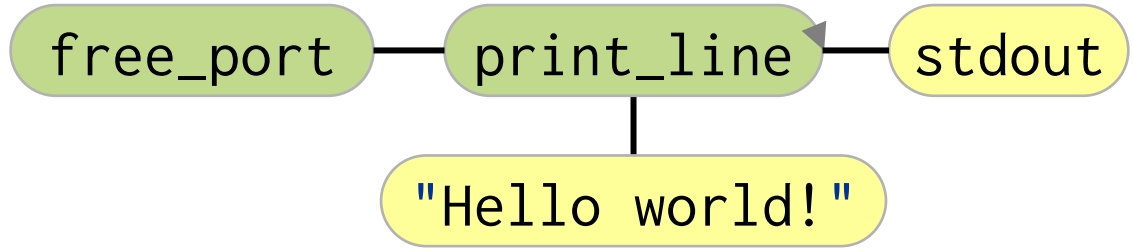
- Type hierarchy:
  - int, float, string ⊑ unary ⊑ ground
  - '<' ⊑ int × int , '=' ⊑ ground × ground
  - CSLMNtal supports typedef (CFG-based; cf. type graphs)

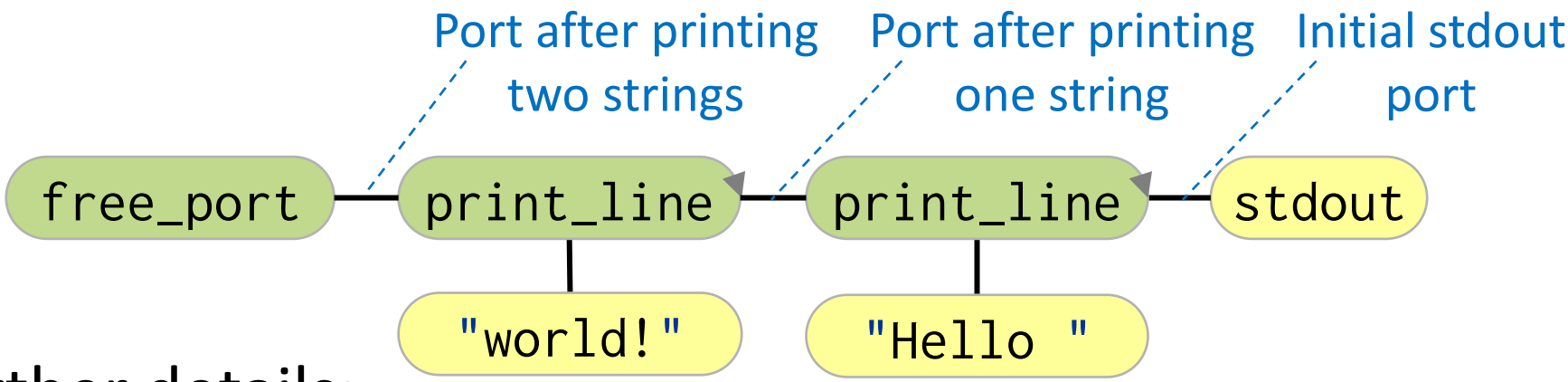
connected graph with specified 'roots'

Hello world! — I/O in declarative style

- Stream-based (or monadic) I/O

```
io.print_line(io.stdout,"Hello world!",io.free_port).
```



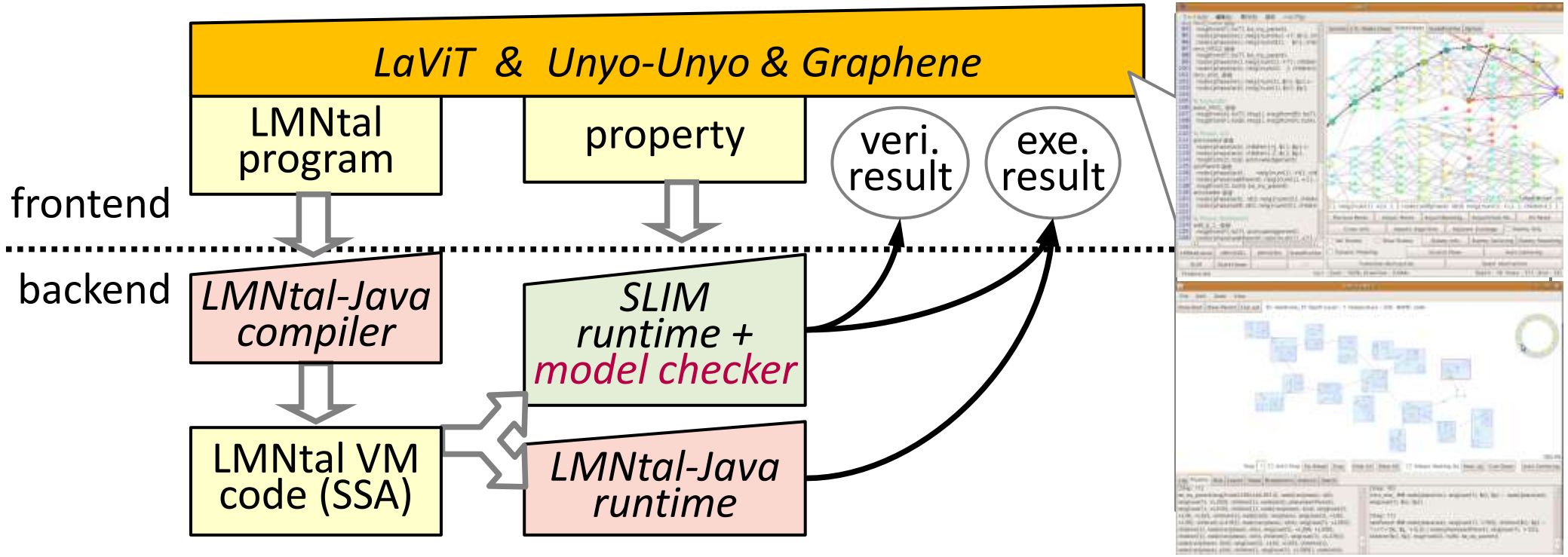


- Further details:

https://www.uedalab.jp/lmntal/index.php?Library%20Reference#io

Implementation overview

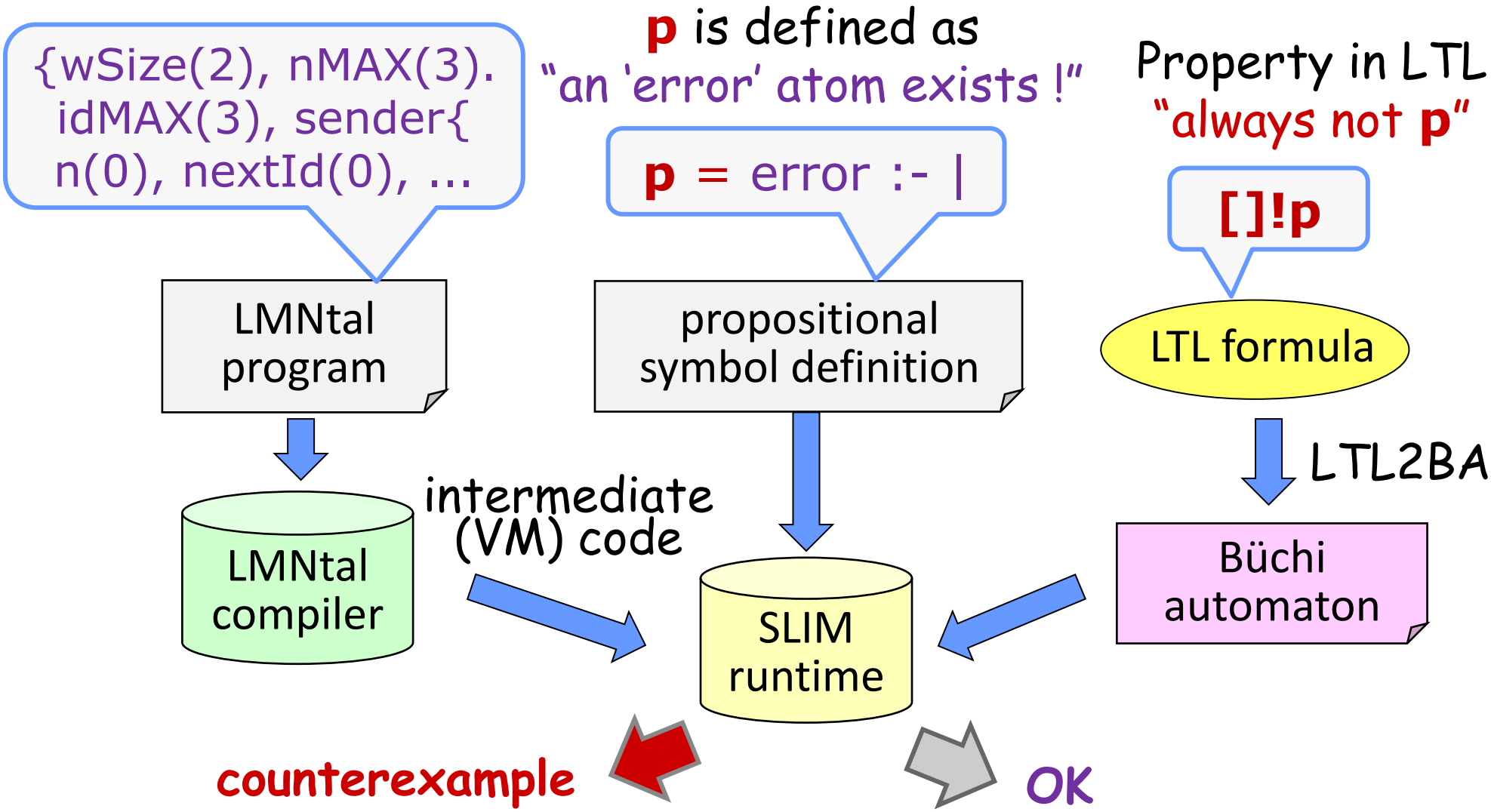
LMNtal-Java	2002–	Java	30kLOC	compiler + runtime w/FLI
Unyo-Unyo	2006–	Java	16kLOC	execution visualizer
SLIM	2007–	C++	34kLOC	faster runtime w/model checker
LaViT	2008–	Java	27kLOC	IDE w/state-space visualizer
Graphene	2014–	Scala	2kLOC	2nd-generation visualizer



Model checking in LMNtal: Motivations

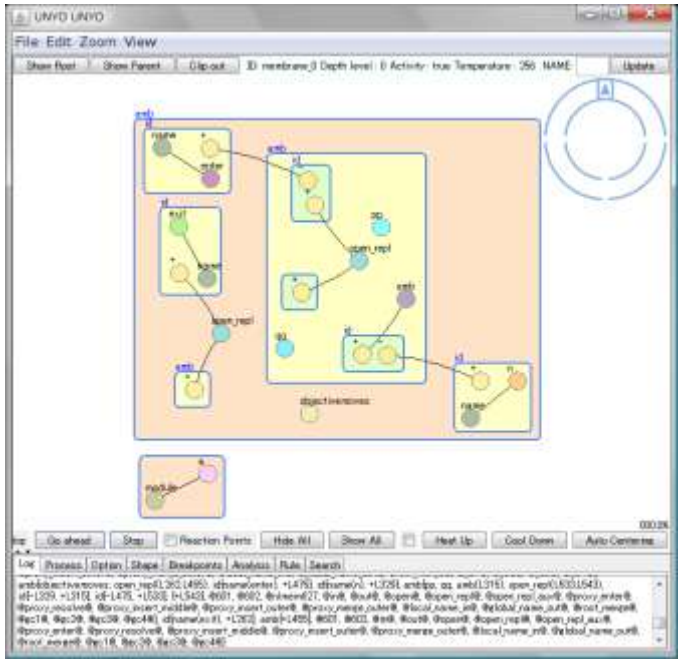
- LMNtal is good at modeling systems which computer-aided verification is concerned with, including
  - state transition systems (automata) and
  - concurrent systems.
- LMNtal is at the same time a full-fledged programming language allowing infinite states.
  - No gap between modeling and programming 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

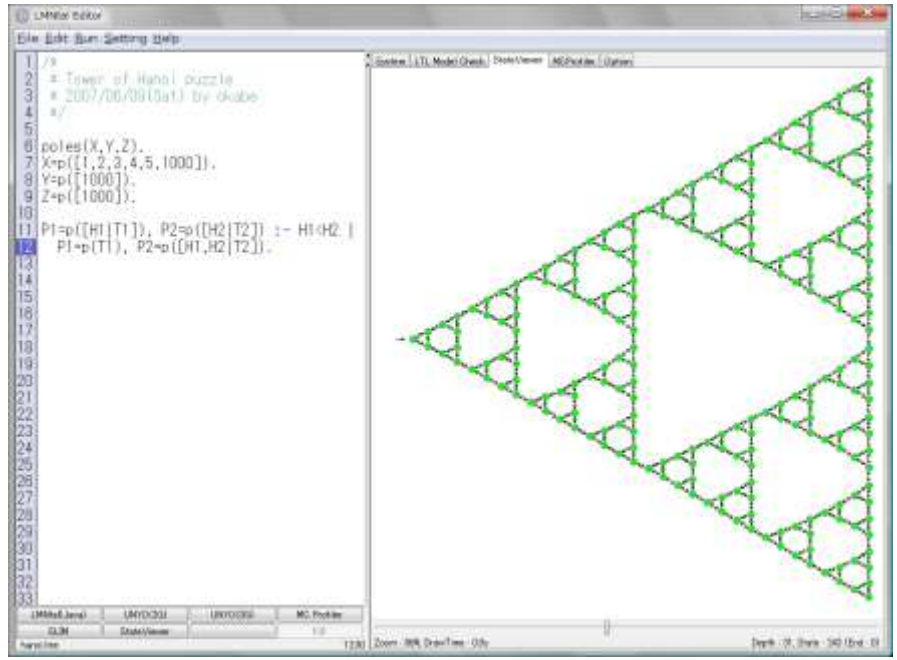


Model checking in LMNtal: Strengths

- 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*



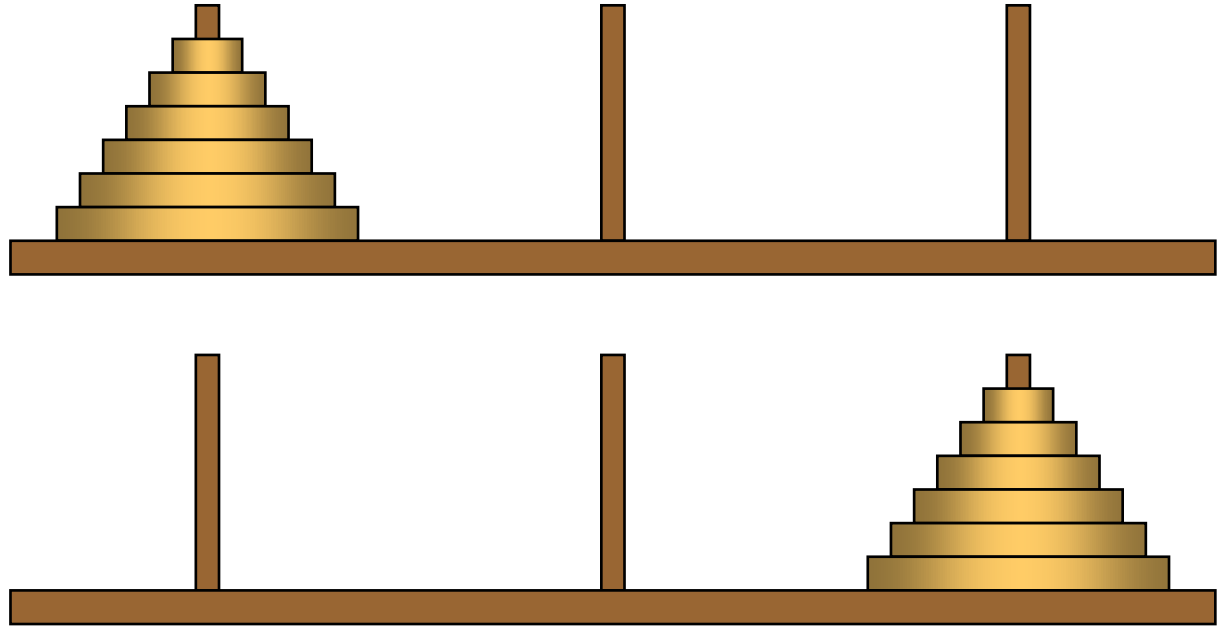
Unyo-Unyo Visualizer



StateViewer (Tower of Hanoi)

Demo: The tower of Hanoi

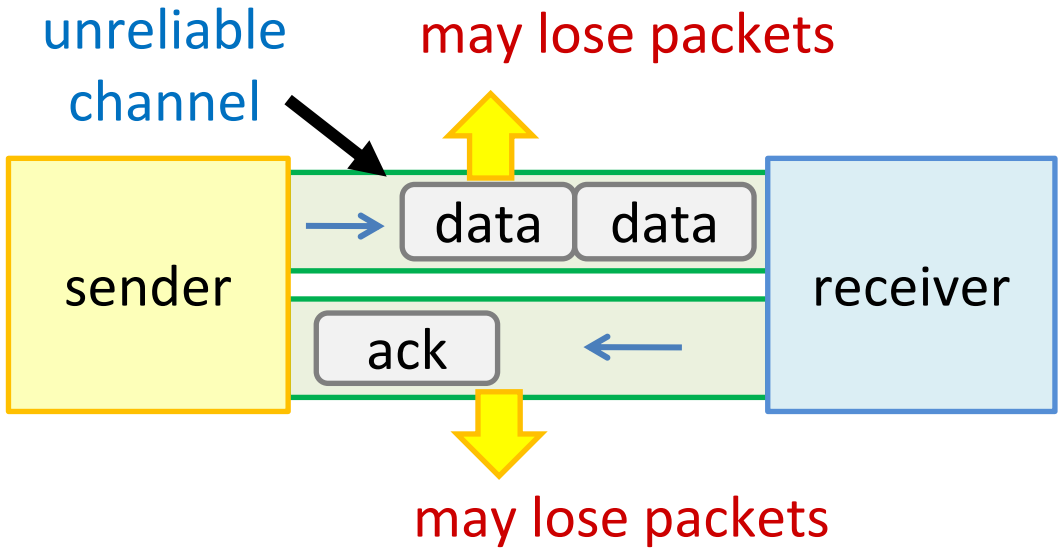
```
poles(p([1,2,3,4,5,6,99]),p([99]),p([99])).
```



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

Demo: Sliding window protocol (SWP)

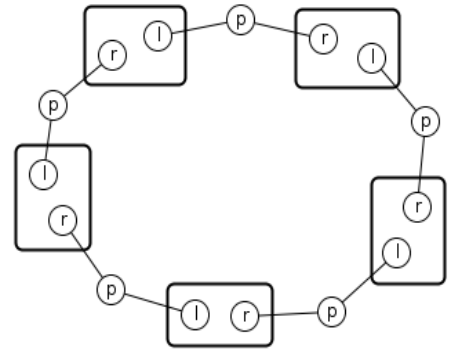
- SWP: transmission protocol used in TCP
  - Sends data packets (up to window size) without waiting for acknowledgment
    - Rollbacks if some item seems lost
  - Channels may lose data and acks



$\Box(send \Rightarrow \Diamond ack)$?

Coping with heavy structure and state explosion

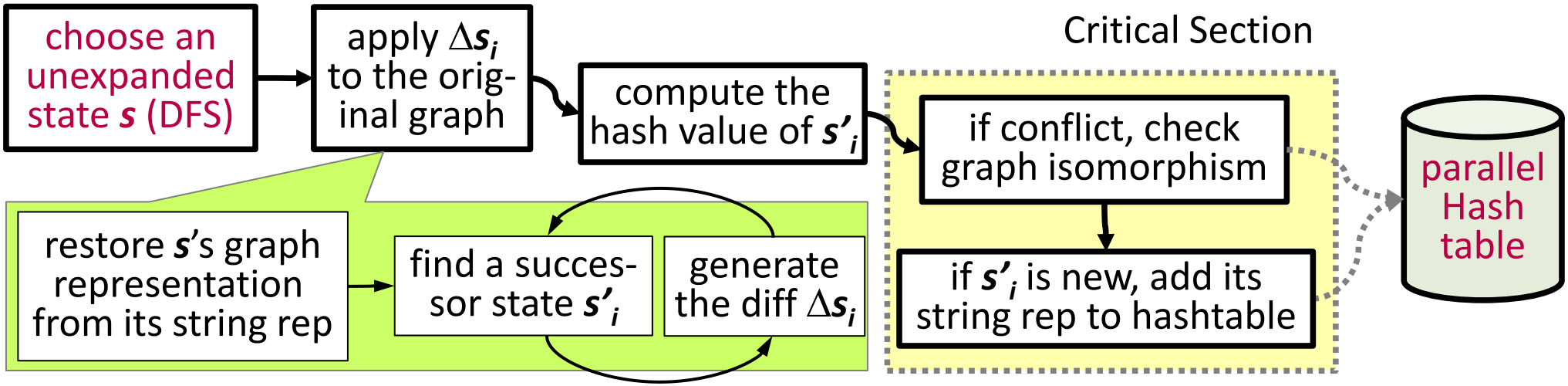
- Managing the state space of graphs requires both *space-efficient* graph representation and *time-efficient* isomorphism checking. They are supported by:
  - Hashing with parallel hash-table
  - State encoding (serialization)
    - Non-canonical encoding
    - Canonical encoding (labelling)
  - Tree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



Original	>2KB
Encoded	70B

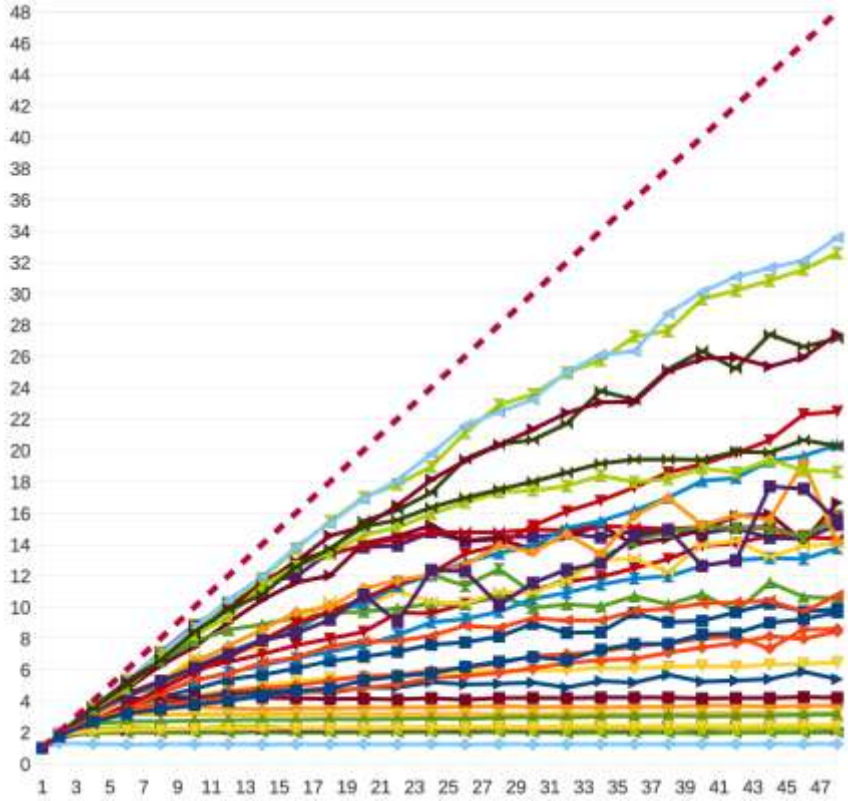
Parallelizing the LMNtal model checker

- 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

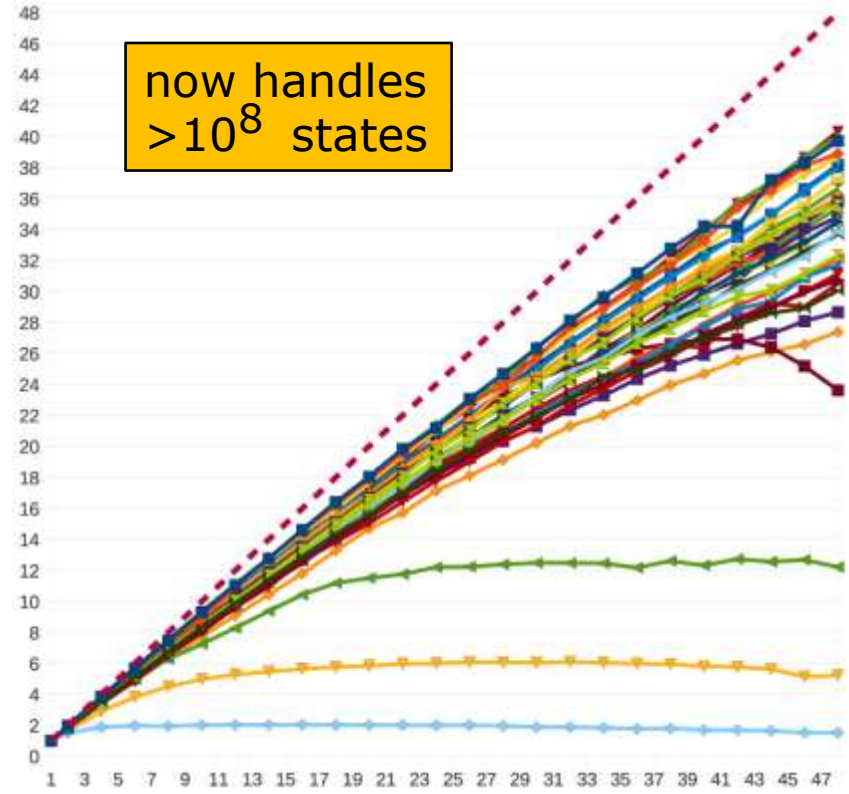


* M. Gocho, T. Hori and K. Ueda, *Computer Software* **28**(4), pp.137-157, 2011

Speedup of the LMNtal parallel model checker



stack slicing



now handles $>10^8$ states

stack slicing with work stealing

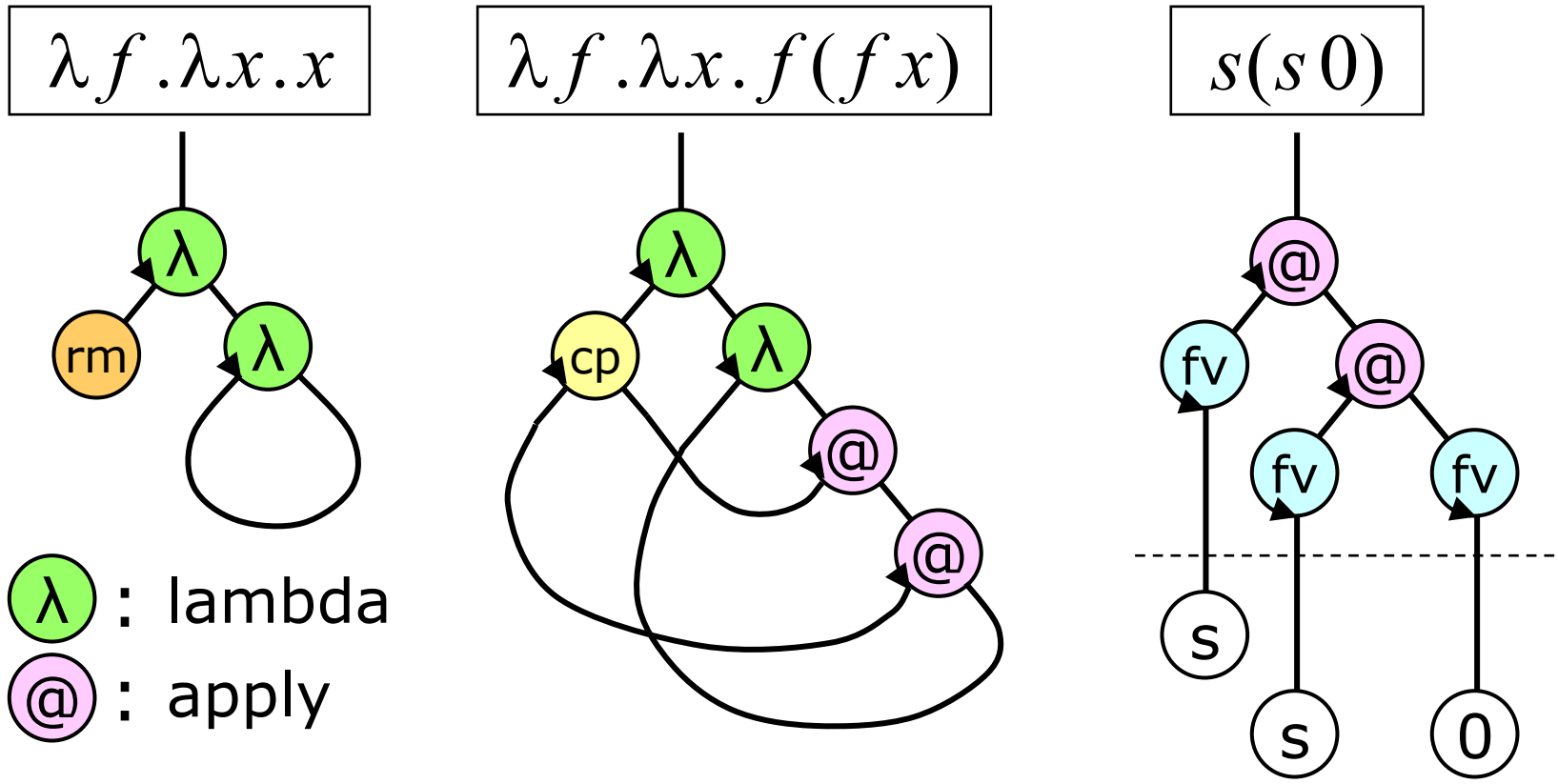
AMD Opteron (2.3GHz) 12-core x 4, 256GB of memory

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

Extensions (almost orthogonal to each other)

- HyperLMNtal (hypergraphs)
- Context (wildcard) extensions
  - 'ground' graphs (flexible handling of hyperlinks)
  - CSLMNtal (for context-sensitive rewriting)
- QLMNtal (quantifiers & negation) [LOPSTR 2024]
- Typing for ports and shapes [PPDP 2024, LNCS 8865, etc.]
- Rule extensions
  - zero-step (instantaneous) rules
  - one-shot rules
  - first-class rules (for metaprogramming)
- Foreign language interface

Lambda calculus, fine-grained (*a la* Interaction Nets)

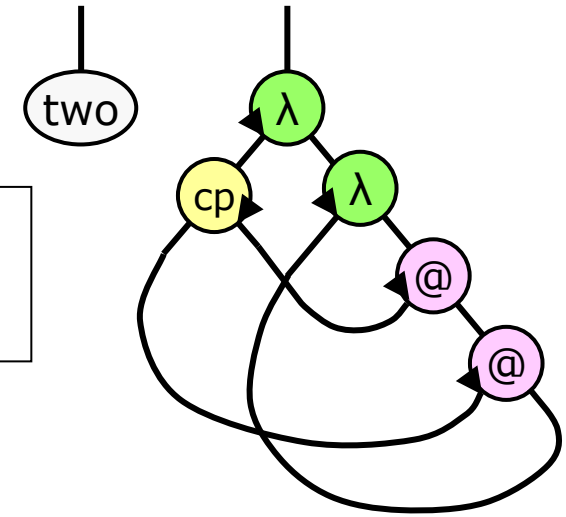


Unique up to cp+rm's equational theory
(= ACU : associativity & commutativity with unit)

Demo: Church numeral exponentiation [RTA 2008]

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

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



- 3^2 : $(((\lambda m.\lambda n.n\ m)\ 3)\ 2)$ – *or just* $(2\ \ 3)$

```
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(s)), fv(0)).
H = apply(fv(s), fv($i)) :- int($i) | H=fv($i +1).
```

Strong reduction by hierarchical token color management

Lambda calculus, coarse-grained [IEEE Access 2021]

- One-to-one translation of the textbook definition, where ‘ground’ for hypergraphs follows ordinary links, then copies/shares hyperlinks depending on how they occur

term structure

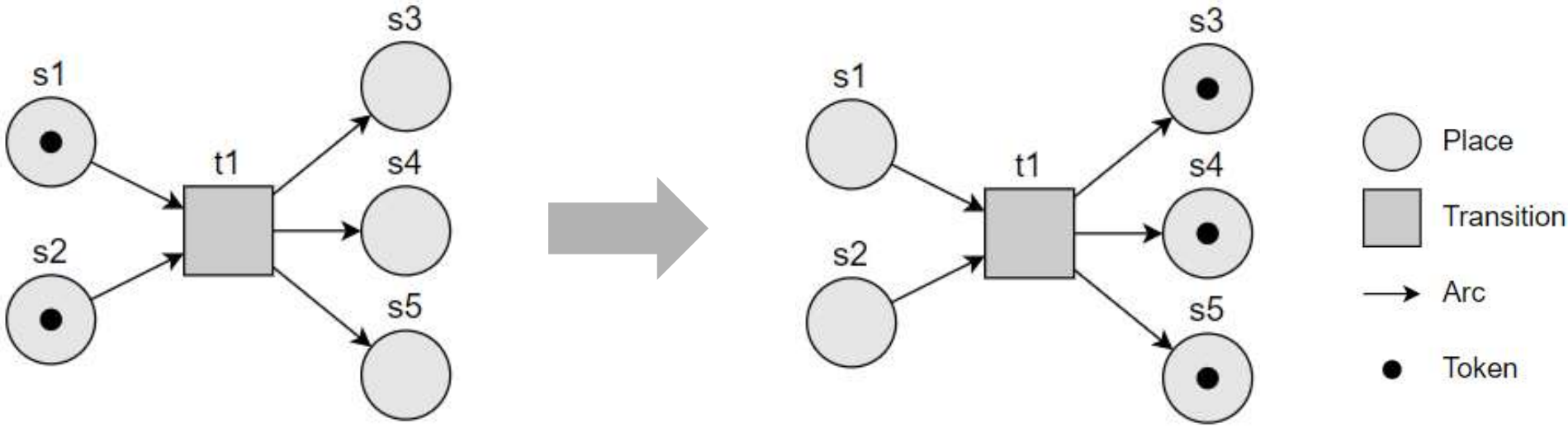
binding structure

```
beta@@ R=app(lam(X,A),B) :- R=subst(A,X,B).
var1@@ R=subst($x,$x,N) :- hlink($x) | R=N.
var2@@ R=subst($x,$y,$n) :-
         $x\=$y, ground($n,1) | R=$x.
abs@@  R=subst(lam($x,M),Y,N):-
         R=lam($x,subst(M,Y,N)).
app@@  R=subst(app(M1,M2),$x,$n) :-
         hlink($x), ground($n,1) |
         R=app(subst(M1,$x,$n),
               subst(M2,$x,$n)).
```

$(\lambda x.A)B \rightarrow A[x \mapsto B]$

Automatically capture-avoiding substitution

Petri Nets in QLMNtal [LOPSTR 2024]

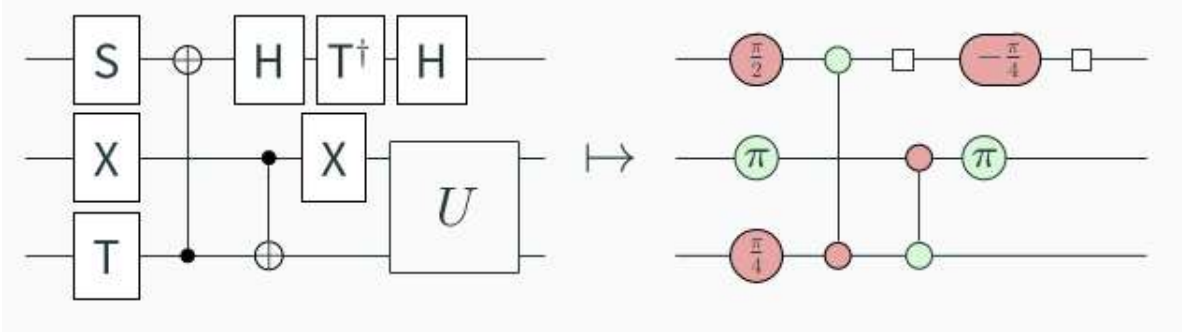


Labelled $\forall^{>0}$ quantifier

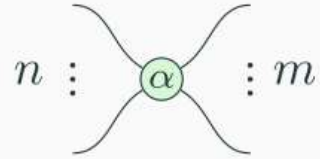
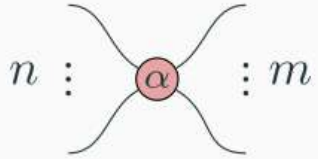
```
{token,-X1},{token,-X2}, {+Y1},{+Y2},{+Y3}.    //places
{+X1,+X2,-Y1,-Y2,-Y3}.                          //transition

M<+>{token,-X,$1}, {M<+> +X,N<+> -Y}, N<+>{+Y,$2} :-
M<+>{-X,$1}, {M<+> +X,N<+> -Y}, N<+>{token,+Y,$2}.
```

ZX-calculus in QLMNtal [PADL 2026]

- From quantum circuits to ZX-diagrams



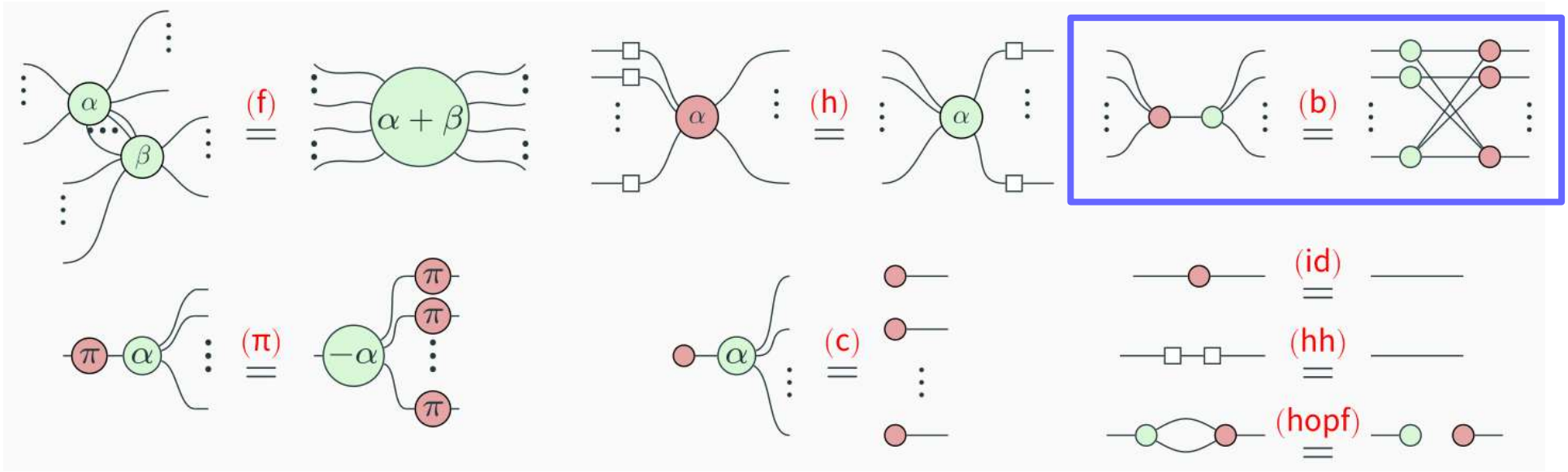
- Can be represented by just two types of nodes (X-spiders and Z-spiders) and wires (that can be bent and stretched)!

Z-spider	X-spider	Hadamard gate
		

- Requires only a small number of universal rewrite rules

ZX-calculus in QLMNtal [PADL 2026]

- ZX rewrite rules:



- Rules with "…" can be represented using possibly nested quantification; e.g., the "bialgebra" rule (b) becomes:

```
{M<*>+L1,+L2,e^i(0),c(1)}, {+L2,N<*>+L3,e^i(0),c(-1)} :-
M<*>{+L1,N<*>+L2,e^i(0),c(-1)}, N<*>{M<*>+L2,+L3,e^i(0),c(1)}
```

Experiences

- Generalized data structures with more *and less* edges
- Concurrency with controllable granularity
- Graph-based model checking (up to 10^9 states)
  - with many implementation techniques
- Unified framework of computation and verification
- Visualization for understanding (cf. verifying) systems

Implementation (available from GitHub) >99% done by students joining and graduating every year

Thank you for your attention!

Questions/suggestions welcome, e.g.,

"Can LMNtal express and execute X ?"
"Why don't you add feature Y ?"
"Could you help me encode my idea Z ?"

To try yourself, visit
http://www.uedalab.jp/lmntal/ and choose LaViT