

Unifying Framework and Tool for the High-performance Modeling and Verification of Hybrid Concurrent Systems

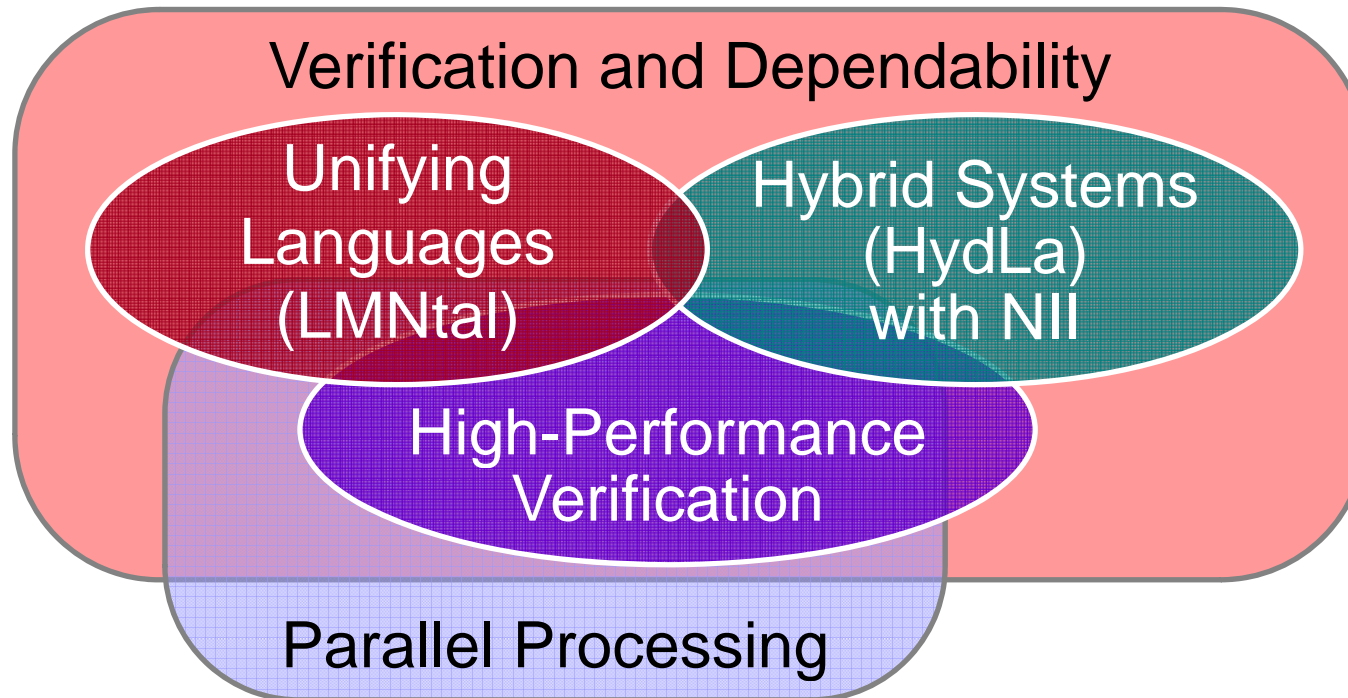
Kazunori Ueda

Dept. of Computer Science and Engineering,
Waseda University and
National Institute for Informatics

November 2009

Research Groups and Their Relationship

2



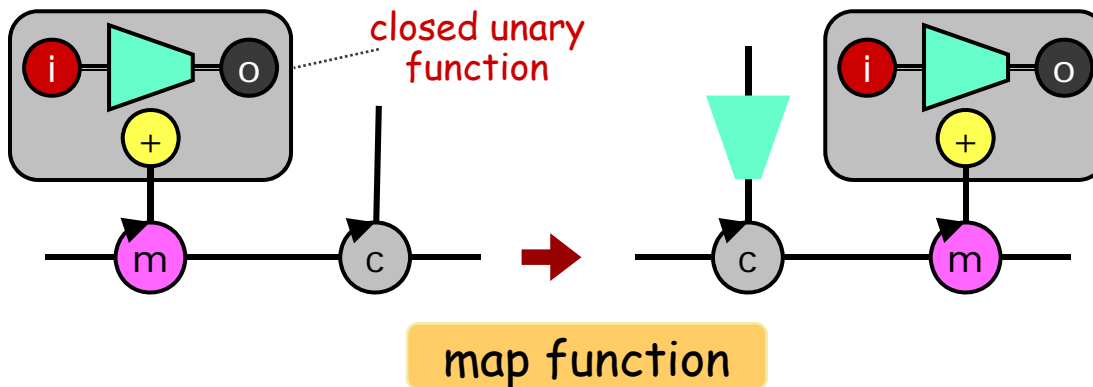
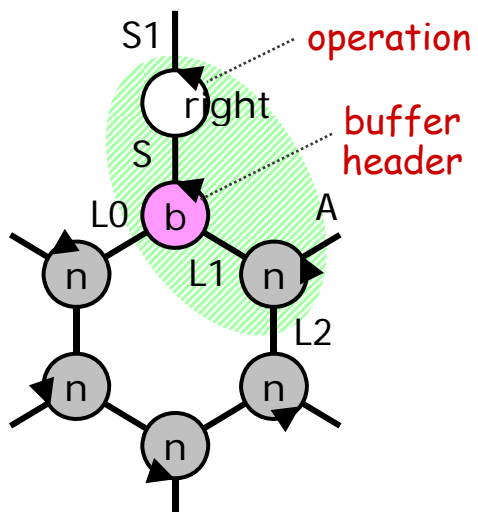
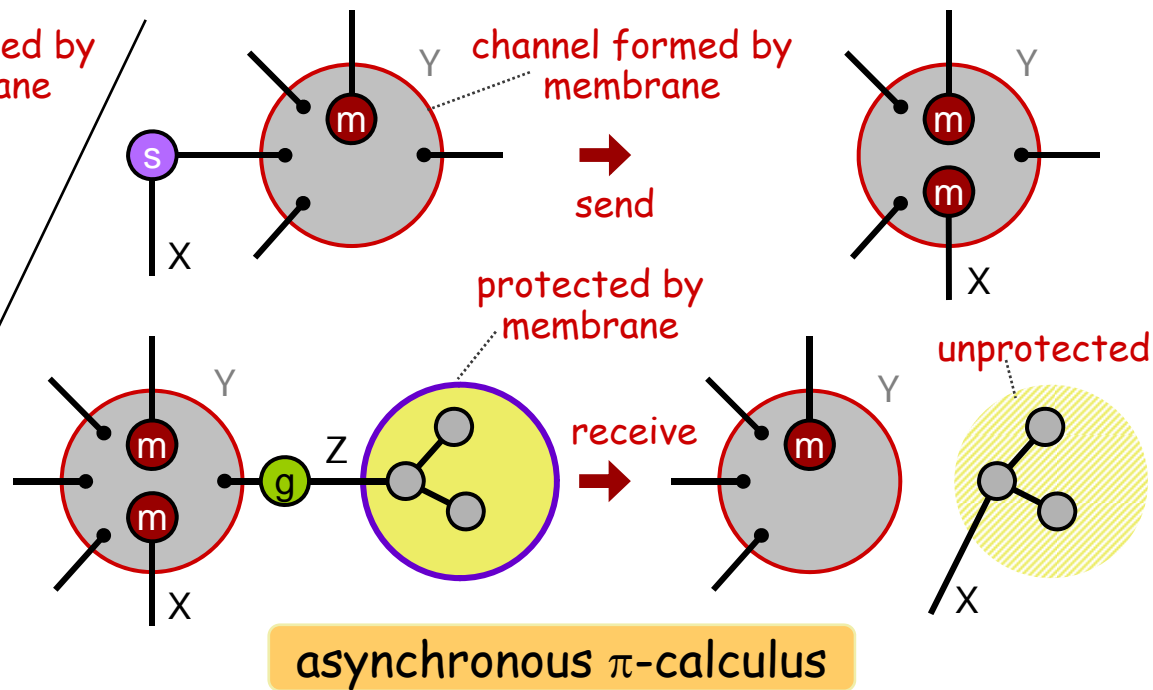
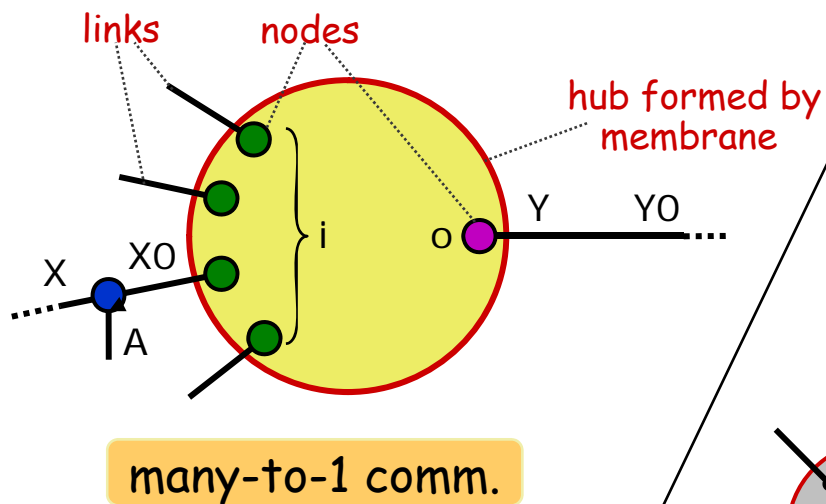
- Three interrelated groups
- Two cross-cutting concerns

Unifying Programming Language

- ◆ **Project LMNtal** (pronounce "elemental")
 - A concurrency model + language + system
 - Started in 2002, now working on the 4G impl.
 - >100,000 LOC involving many people over the years
 - Focused on verification (state-space search, LTL model checking) since 2007
 - Provides an IDE with visualization
- ◆ Ready to use; very low entry barrier
 - <http://www.ueda.info.waseda.ac.jp/lmntal/>
- ◆ Papers: [Ueda, RTA'08] [Ueda et al., ICTAC'09] [Ueda, TCS 410 (2009)]

- ◆ 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**
 - **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**

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



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

- ◆ (Coloured) 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**

Models and languages with **membranes + hierarchies**

- ◆ (Coloured) 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**

* : some versions
feature hierarchies

- ◆ **Seal calculus**
- ◆ **Kell calculus**
- ◆ **Brane calculi**
- ◆ **κ**

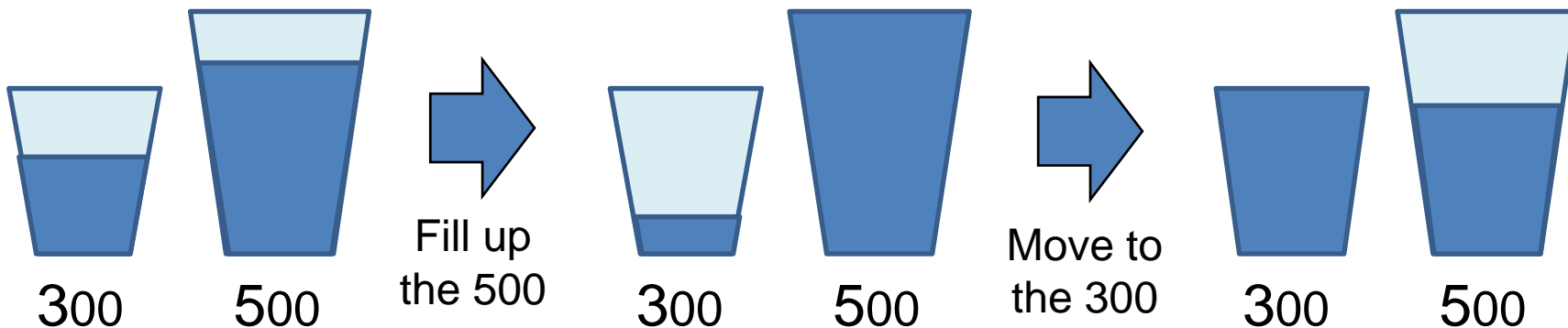
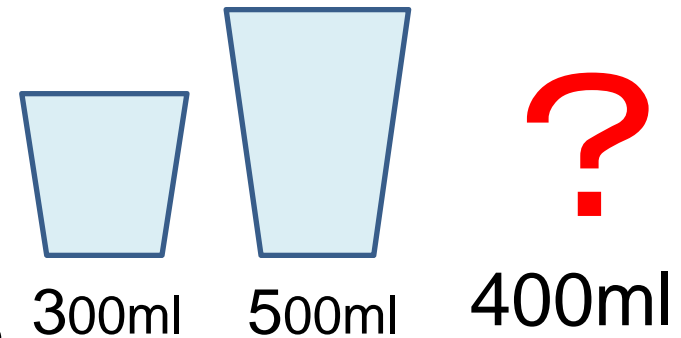
Demo: Water Jug Problem

- ◆ Typical AI search problem

- E.g. use a 300ml jug and a 500ml jug to get 400ml of water

- ◆ Operations:

- Empty a jug
- Fill up a jug with tap water
- Move water until it's emptied
- Move water until the other is filled



Towards Model Checking in LMNtal

- ◆ **Computer-aided verification** deals with systems for which LMNtal allows concise description:
 - state transition systems (automata)
 - multiset rewriting systems
 - concurrent systems
- ◆ LMNtal is at the same time a **full-fledged programming language**.
 - **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?

- ◆ The LMNtal IDE 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
- ◆ Challenge: implementing state management
 - graph isomorphism: a notoriously hard problem in general

- ◆ Applications so far
 - real-time scheduler
 - security / data transfer protocol
 - AI search
 - checking of the fine-grained, graph encoding of the untyped lambda calculus [Ueda, RTA'08]
 - etc.
- ◆ **Multiset rewriting** allows concise encoding of problems (e.g., n-queens) and state-space (symmetry) reduction (e.g., philosophers)
- ◆ **Visualization** turned out to be very useful for understanding systems

- ◆ High-performance parallel computing applied to computer-aided verification
 - **Parallel SAT** (propositional satisfiability) solver
 - First step: **c-sat** (cluster-based scalable solver) [Ohmura and Ueda, SAT '09]
 - Engine for verification and AI applications
 - ❖ Model checking, AI planning, constraint satisfaction, . . .
 - **Parallel model checkers**
 - Algorithms (task partitioning, etc.)
 - Systems (multi-core LMNtal underway)
- ◆ Few research groups worldwide

Question : How scalable is distributed-memory parallel SAT based on "modern" solvers?

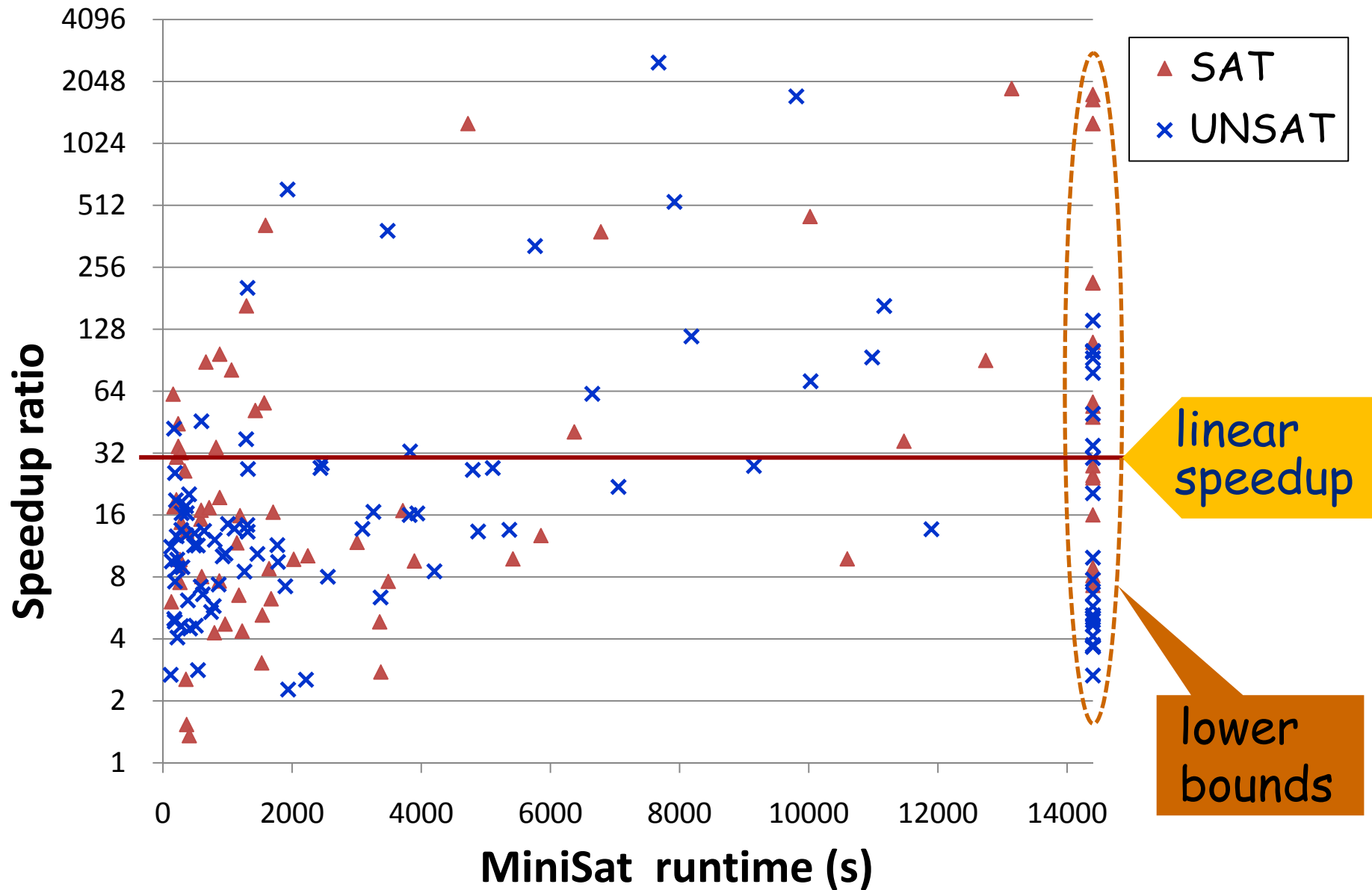
- SAT search is notoriously irregular.

Goal : comprehensive evaluation of **c-sat** on Beowulf clusters with 32-64 PEs (cores)

Result: **>23x** speedup (geometric mean) with **31 PEs** for 189 hard instances from SAT Races / Competition

- hundreds/thousands of machine hours
- **>31x** / **>19x** for **SAT** / **UNSAT** instances

Scalability (31 PEs, "hard" instances)



- ◆ Involves both continuous and discrete changes
 - billiard, air-conditioning, crossing gate, . . .
- ◆ Programming language aspects rather unexplored
- ◆ Designing and implementing **HydLa**, a constraint-based language for modeling hybrid systems
 - Key challenges are the semantics and validated computation under uncertainties
- ◆ Interdisciplinary field, ranging over logic, constraints, numerical computation and programming languages
- ◆ Relates also to biology and control engineering

- ◆ **Declarative** (\Leftrightarrow procedural)
 - employs popular math and logical notations
- ◆ **Constraint-based**
 - differential equations and other constraints define **functions over time**
 - able to handle **partial information** (numerical errors, uncertainties, etc.)
- ◆ Features **constraint hierarchies**
 - It's difficult to describe systems so that the constraints are **consistent and well-defined**.
 - cf. Frame problem

HydLa Example: Sawtooth

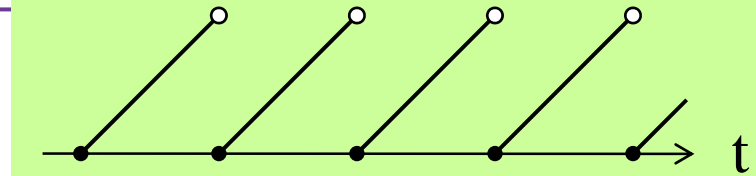
17

INIT $\Leftrightarrow 0 \leq f < 1$.

INCREASE $\Leftrightarrow \square (f' = 1)$.

DROP $\Leftrightarrow \square (f- = 1 \Rightarrow f = 0)$.

INIT, (INCREASE \ll DROP).



- ◆ Describes properties at time 0
- ◆ Time is implicit
 - $\square (f' = 1)$ means $\forall t \geq 0 (f'(t) = 1)$
- ◆ Intuitive meaning of constraint hierarchy is the maximal consistent set of the constraints, subject to hierarchy and atomicity

Example: Crossing gate (cf. Henzinger, LICS'96)

18

```
TRAIN(N, Ipos, Vel) {  
  train(N) = Ipos  $\wedge$   $\square$ (train(N)' = Vel) }
```

```
SENSOR(S, Pos, Sig) {  $\square$ (Sig = 0)  
  <<  $\forall N \in S$  ( $\square$ (train(N) = Pos  $\Rightarrow$  Sig = 1)) }
```

```
CONTROLLER {  $\square$ (raise' = 0)  
  << (raise = 1  
     $\wedge$   $\square$ (app = 1  $\Rightarrow$  raise5 = 0)  
     $\wedge$   $\square$ (exit = 1  $\Rightarrow$  raise5 = 1)) }
```

```
GATE {  $\square$ (g' = 0)  
  << (g = 90  
     $\wedge$   $\square$ (raise  
     $\wedge$   $\square$ (raise
```

```
TRAIN(1, 4000, -  
SENSOR({1,2}, 10  
CONTROLLER, GA
```

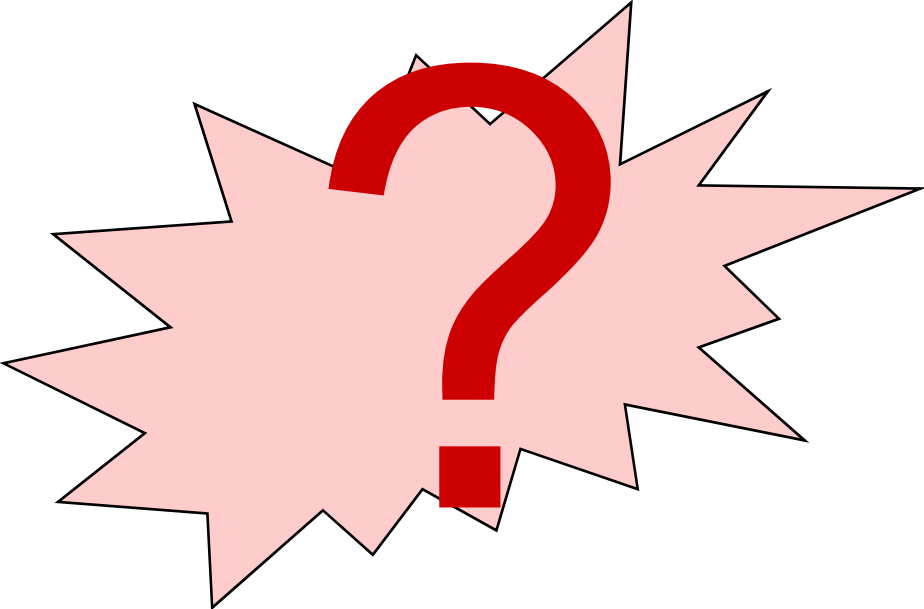
compositionality

communication

- ◆ Both symbolic and interval-based implementations underway.
- ◆ This is really non-obvious due to the interplay between uncertainties and conditionals [Ishii, Ueda et al., ADHS'09].

Computing paradigms must change . . .

Concurrency
Everywhere!

20th century	21st century
<p>von Neumann architecture + sequential computation</p>	<p>multi-core / clusters / Grid / distributed / embedded / molecular / ...</p>
<ul style="list-style-type: none"> ◆ Turing Machines (computability) ◆ RAM model (complexity) ◆ λ-calculus (programming languages) ◆ Floating point arithmetic (numerical analysis) 	

- ◆ Computing paradigm as a reliable **and accessible** infrastructure
 - to prepare for more challenging and sophisticated apps
- ◆ Unify
 - programming and modeling
 - computation and verification
 - symbolic and numerical processing

- ◆ Possible key principles: constraints (Univ. Nantes, LINA)
- ◆ Other possible common backgrounds:
 - concurrency, nondeterminism
 - hybrid systems
 - verified software
- ◆ Our (relative) strength:
 - language and system implementation
 - parallelization
- ◆ Need of collaboration:
 - reaching out to neighboring disciplines inside and outside CS (e.g., control and biology)
 - exploring unforeseen applications

Thanks for your attention!