Model Checking

Javier Esparza and Stephan Merz

Lab. for Foundations of Computer Science, University of Edinburgh

Institut für Informatik, Universität München

Program

9:00–10:00 Basics

- A bit of history
- A case study: the Needham-Schroeder protocol
- Linear and branching time temporal logics

10:00–10:30 Model-checking LTL I

- The automata-theoretic approach

10:30–11:00 Coffee Break

11:00–11:30 Model-checking LTL II

- On-the-fly model checking
- Partial-order techniques

11:30–12:30 Model-checking CTL

- Basic algorithms
- Binary Decision Diagrams

12:30–14:00 Lunch

14:00–15:30 Abstraction

- Basics
- Predicate Abstraction

15:30–16:00 Coffee Break

16:00–17:30 Infinite state spaces

- Sources of infinity
- Symbolic search
- Accelerations and widenings

Basics

A bit of history

A case study: the Needham-Schroeder protocol

Linear and branching time temporal logics

A bit of history

Goal: automatic verification of systems

Prerequisites: formal semantics and specification language

- In the beginning there were Input-Output Systems . . .

  Total correctness $=$ partial correctness $+$ termination

  Formal semantics: input-output relation

  Specification language: first-order logic.

- Late 60s: Reactive systems emerge . . .

  Reactive systems do not "compute anything"

  Termination may not be desirable (deadlock!)

  Total correctness: safety $+$ progress $+$ fairness . . .

  Formal semantics: Kripke structures, transition systems ($\sim$ automata)

  Specification language: Temporal logic

Temporal logic

- Middle Ages: analysis of modal and temporal inferences in natural language.

  Since yesterday she said she'd come tomorrow, she'll come today.

- Beginning of the 20th century: Temporal logic is formalised

  Primitives: always, sometime, until, since . . .

  Prior: *Past, present, and future*. Oxford University Press, 1967

- 1977: Pnueli suggests to use temporal logic as specification language

  Temporal formulas are interpreted on Kripke structures

  A. Pnueli: *The Temporal Logic of Programs*. FOCS '77

"System	satisfies	property"
	formalised as	
Kripke structure	is model of	temporal formula

Automatising the verification problem

Given a reactive system *S* and a temporal formula ϕ, give an algorithm to decide if the system satisfies the formula.

- Late 70s, early 80s: reduction to the validity problem

  1. Give a proof system for checking validity in the logic (e.g. axiomatization)
  2. Extract from *S* a set of formulas *F*
  3. Prove that $F \rightarrow \phi$ is valid using the proof system

  Did not work: step 3 too expensive

- Early 80s: reduction to the model checking problem

  1. Construct and store the Kripke structure $\mathcal{K}$ of *S* $\rightarrow$ restriction to finite-state systems
  2. Check if $\mathcal{K}$ is a model of ϕ directly through the definition

  Clarke and Emerson: *Design and synthesis of synchronisation skeletons using branching time temporal logic*. LNCS 131, 1981
  Quielle and Sifakis: *Specification and verification of concurrent systems in CESAR*. 5th International Symposium on Programming, 1981

Making the approach work

State explosion problem: the number of reachable states grows exponentially with the size of the system

- Late 80s, 90s: Attacks on the problem

  Compress. Represent sets of states succinctly: Binary decision diagrams, unfoldings.

  Reduce. Do not generate irrelevant states: Stubborn sets, sleep sets, ample sets.

  Abstract. Aggregate equivalent states: Verification diagrams, process equivalences.

- 90s, 00s: Industrial applications

  Considerable success in hardware verification (e.g. Pentium arithmetic verified)

  Groups in all big companies: IBM, Intel, Lucent, Microsoft, Motorola, Siemens . . .

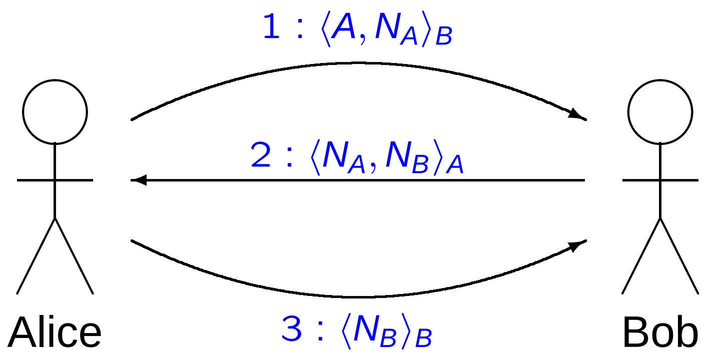
  Many commercial and non-commercial tools: FormalCheck, PEP, SMV, SPIN . . .

  Exciting industrial and academic jobs!

- 90s, 00s: Extensions: Infinite state systems, software model-checking

Case study: Needham-Schroeder protocol

Establish joint secret (e.g. pair of keys) over insecure medium



- secret represented by pair $\langle N_A, N_B \rangle$ of "nonces"
- messages can be intercepted
- assume secure encryption and uncompromised keys

Is the protocol secure?

Protocol analysis by model checking

Representation as finite-state system

Finite number of agents	Alice, Bob, Intruder
Finite-state model of agents	limit honest agents to single protocol run
	one (pre-computed) nonce per agent
	describe capabilities of intruder with limited memory
Simple network model	shared communication channels
Simulate encryption	pattern matching instead of computation

Protocol description in $B(PN)^2$ (Basic Petri Net Programming Notation)

Input language for the PEP tool

`http://theoretica.informatik.uni-oldenburg.de/ pep/`

$B(PN)^2$ model of honest agents

Model for Alice

```
begin
    nondeterministically choose partner
    < PartnerKey'=KeyB OR PartnerKey'=KeyI >;
    send initial message, modelled as a triple (key,d1,d2)
    < msg1!=PartnerKey AND msg2!=Alice AND msg3!=NonceA >;
    expect matching reply from partner
    < KeyA=msg1? AND NonceA=msg2? AND PartnerANonce'=msg3? >;
    send final message
    < msg1!=PartnerAKey AND msg2!=PartnerANonce AND msg3!=0 >;
    declare success
    < StatusA'=1 >
end
```

Similar model for Bob

B(PN)^2 model of intruder (1)

```
begin
   do
      receive or intercept message, decrypt if possible
      < IntKey'=msg1! AND IntD1'=msg2! AND IntD2'=msg3? >;
      do < IntKey=KeyI AND
          (IntD1=NonceA OR IntD1=NonceA) AND KnowNA'=1 >; exit
      [] < IntKey=KeyI AND
          (IntD1=NonceB OR IntD1=NonceB) AND KnowNB'=1 >; exit
      od; repeat
   [] replay intercepted message
      < msg1!=IntKey AND msg2!=IntD1 AND msg3!=IntD2 >; repeat
```

B(PN)^2 model of intruder (2)

```
  [] do compose/fake initial message
        choose identity and nonce
        < Sender'=Alice OR Sender'=Bob OR Sender'=Intruder >;
        < (KnowNA=1 AND Nonce'=NonceA) OR
          (KnowNB=1 AND Nonce'=NonceB) OR
           Nonce'=NonceI >;
        < msg1!=KeyB AND msg2!=Sender AND msg3!=Nonce >; exit
     [] fake reply if NonceA is known
        choose nonce
        < (KnowNA=1 AND Nonce'=NonceA) OR
          (KnowNB=1 AND Nonce'=NonceB) OR
           Nonce'=NonceI >;
        < KnowNA=1 AND msg1!=KeyA AND msg2!=NonceA AND msg3!=Nonce >; exit
     [] fake acknowledgement if NonceB is known
        < KnowNB=1 AND msg1!=KeyB AND msg2!=NonceB AND msg3!=0 >; exit
     od; repeat
  od
end
```

Protocol analysis using PEP/The Model Checking Kit

Input

B(PN)^2 model of protocol

Property expressed as temporal logic formula

$$\mathbf{G}\left(\mathit{StatusA} = 1 \wedge \mathit{StatusB} = 1 \;\Rightarrow\right.$$
$$\left.(\mathit{PartnerAKey} = \mathit{KeyB} \Leftrightarrow \mathit{PartnerBKey} = \mathit{KeyA})\right)$$

Verification (Sun Ultra 60, 295 MHz, 1.5 GB)

Program automatically translated into Petri net:
130 places, 5461 transitions, 8279 reachable markings

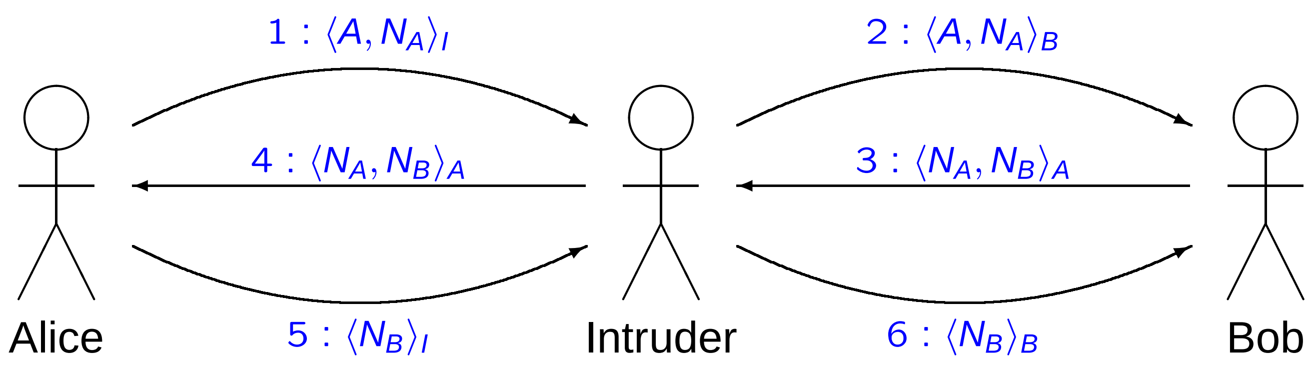
“Compressed” state space computed in about 2 minutes

Shortest run violating the property computed in about 2 seconds

Protocol bug

Alice (correctly) believes to talk with Intruder

Bob (incorrectly) believes to talk with Alice



Bug went undetected for 17 years

Three steps to model checking

1. Model abstraction of system under investigation

- reduce number of processes
- limit computational resources
- increase non-determinism
- coarser grain of atomicity

2. Validate model

- simulation ensures existence of certain executions
- check “obvious” properties

3. Run model checker for properties of interest

“true”	property holds of model, and perhaps of system
“false”	counterexample guides debugging of model and/or system
timeout	review model, tune parameters of model checker

Kripke structures

Basic model of computation $\mathcal{K} = (S, I, \delta, AP, L)$

S	system states (control, variables, channels)
$I \subseteq S$	initial states
$\delta \subseteq S \times S$	transition relation
AP	atomic propositions over states
$L\colon S \to 2^{AP}$ (labels)	labelling function

All states assumed to have at least one successor

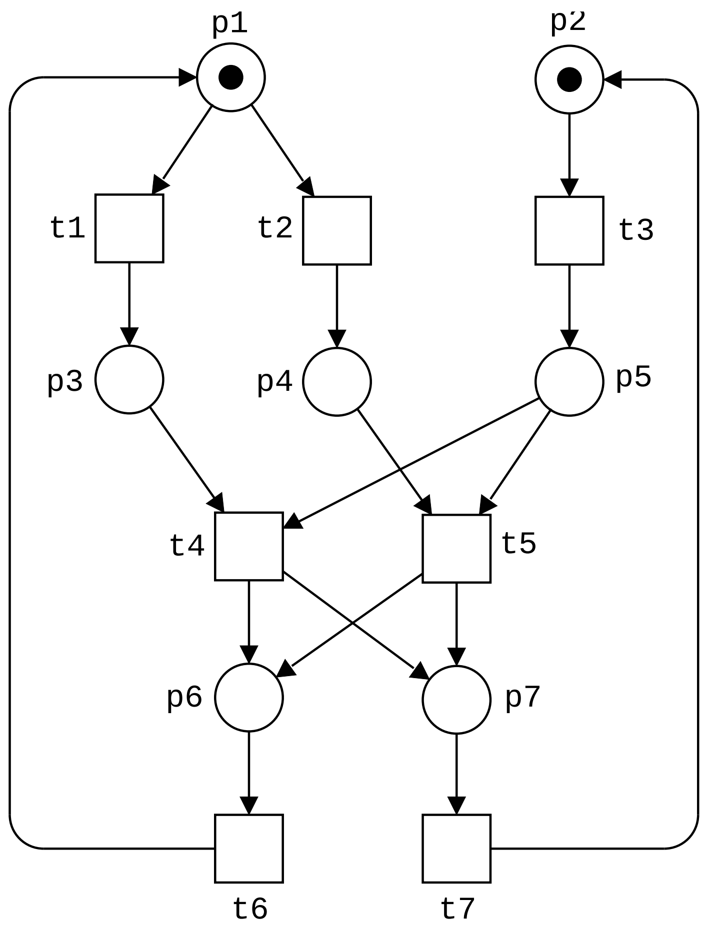
$\mathcal{K}$ described in modelling language (e.g., $B(PN)^2$, Petri nets, process algebra)

Size of $\mathcal{K}$ usually exponential in size of description

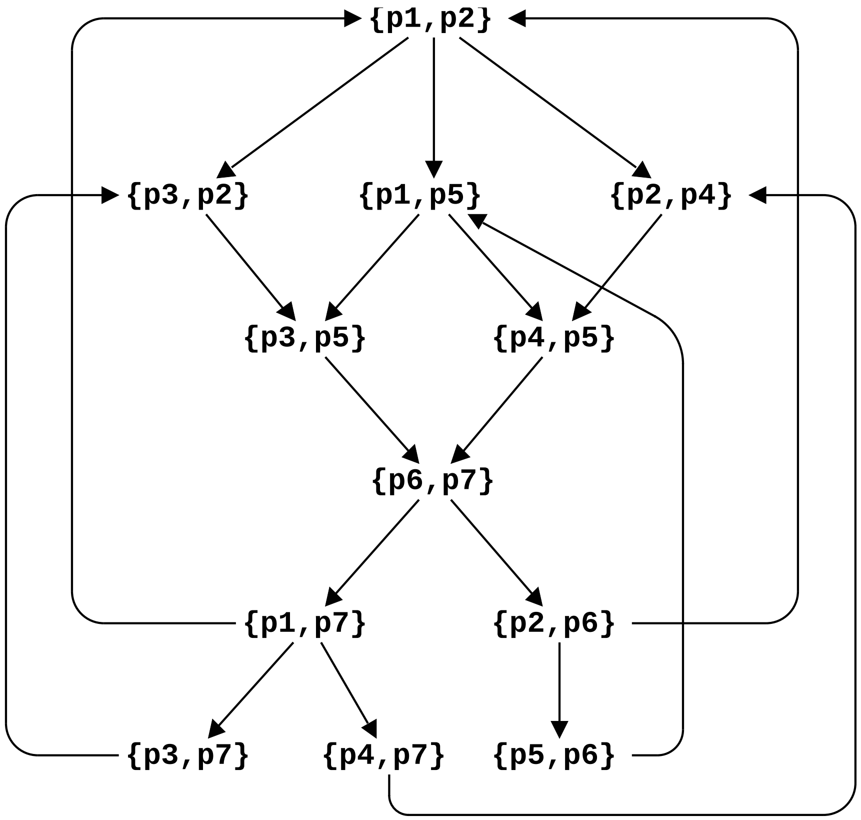
Petri net view

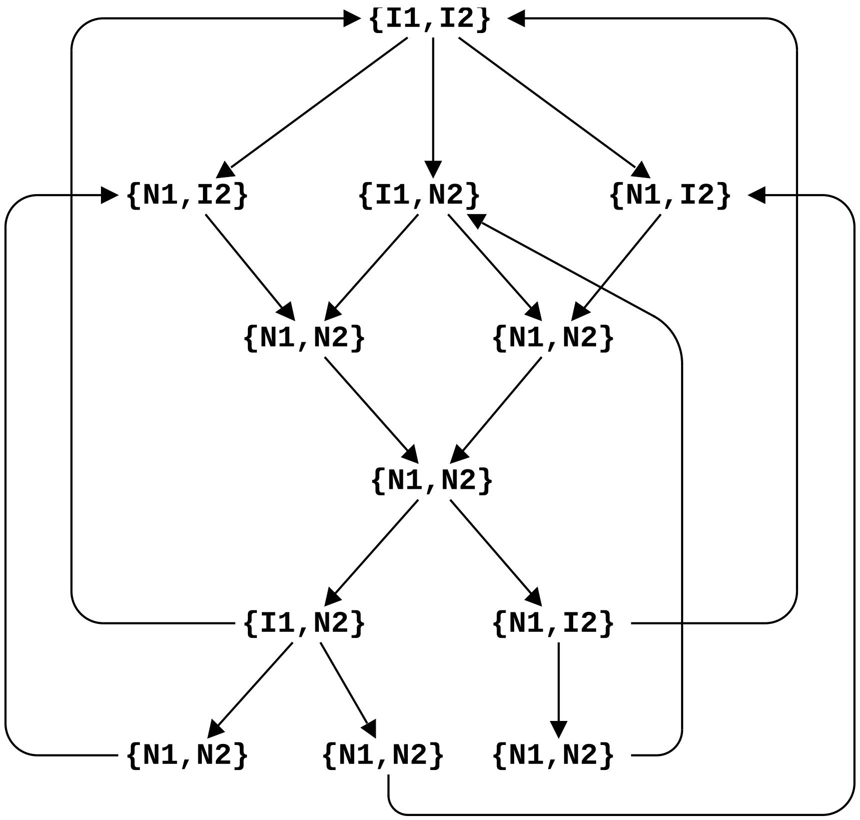
S	reachable markings
AP	set of places
$L(M)$	set of places marked at M

Example: Petri net



Example: Kripke structures





Computations of Kripke structures

Computations of $\mathcal{K} = (S, I, \delta, AP, L)$

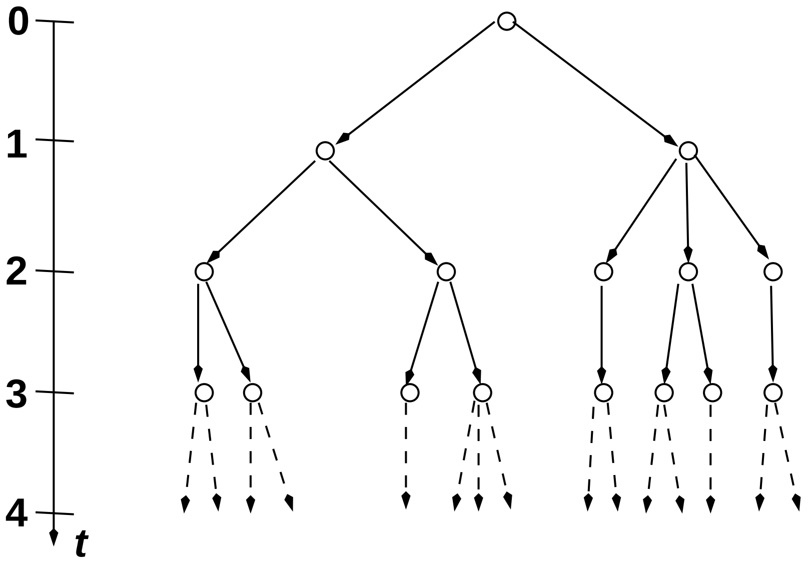
infinite sequences $L(s_0)L(s_1)\ldots \in S^\omega$ satisfying $s_0 \in I$ and $(s_i, s_{i+1}) \in \delta$

Petri net view

infinite sequences of markings $M_0M_1\ldots$ starting at an initial marking and obeying the firing rule

Computation tree

represents all computations of $\mathcal{K}$



nodes	system states
edges	transitions
paths	computations
branching	non-determinism
	(e.g., interleaving)

Linear-time temporal logic (LTL)

Express time-dependent properties of system runs

Evaluated over infinite sequences of labels (computations or not)

type	formula	$\rho \models \varphi$ iff ...
atomic	$p \in AP$	p holds of ρ_0
boolean	$\neg\varphi$	$\rho \not\models \varphi$
	$\varphi \vee \psi$	$\rho \models \varphi$ or $\rho \models \psi$
temporal	$\mathbf{X}\,\varphi$	$\rho\vert_1 \models \varphi$
	$\mathbf{F}\,\varphi$	$\rho\vert_i \models \varphi$ for some $i \in \mathbb{N}$
	$\mathbf{G}\,\varphi$	$\rho\vert_i \models \varphi$ for all $i \in \mathbb{N}$
	φ **until** ψ, $\varphi\,\mathbf{U}\,\psi$	there is $i \in \mathbb{N}$ such that $\rho\vert_i \models \psi$ and $\rho\vert_j \models \varphi$ for all $0 \leq j < i$
	φ **unless** ψ, $\varphi\,\mathbf{W}\,\psi$	$\rho \models \varphi$ **until** ψ or $\rho \models \mathbf{G}\,\varphi$

System validity: $\mathcal{K} \models \varphi$ iff $\sigma \models \varphi$ for all computations of $\mathcal{K}$

LTL: examples

Invariants	$\mathbf{G}\, P$
$\mathbf{G}\,\neg(\mathit{crit}_1 \wedge \mathit{crit}_2)$	mutual exclusion
$\mathbf{G}(\mathit{preset}_1 \vee \ldots \vee \mathit{preset}_n)$	deadlock freedom
Response, recurrence	$\mathbf{G}(P \Rightarrow \mathbf{F}\, Q)$
$\mathbf{G}(\mathit{try}_1 \Rightarrow \mathbf{F}\, \mathit{crit}_1)$	eventual access to critical section
$\mathbf{G}\,\mathbf{F}\,\neg\mathit{crit}_1$	no starvation in critical section
Reactivity, Streett	$\mathbf{G}\,\mathbf{F}\, P \Rightarrow \mathbf{G}\,\mathbf{F}\, Q$
$\mathbf{G}\,\mathbf{F}(\mathit{try}_1 \wedge \neg\mathit{crit}_2) \Rightarrow \mathbf{G}\,\mathbf{F}\,\mathit{crit}_1$	strong fairness
Precedence	$\mathbf{G}(P_1$ **unless** $\ldots$ **unless** $P_n)$
$\mathbf{G}(\mathit{try}_1 \wedge \mathit{try}_2 \Rightarrow \neg\mathit{crit}_2\ \mathbf{W}\ \mathit{crit}_2\ \mathbf{W}\ \neg\mathit{crit}_2\ \mathbf{W}\ \mathit{crit}_1)$	1-bounded overtaking

Branching-time temporal logic

Include assertions about branching behavior

combine temporal modalities and quantification over paths

Example: CTL   Computation Tree Logic

	Q	T	
		X	**successor (next)**
for some path	**E**	**F**	**sometime in the future (eventually)**
for all paths	**A**	**G**	**always in the future (globally)**
		U, W	**until, unless**

evaluated at subtree $\mathcal{K}, s \models \varphi$

system validity $\mathcal{K} \models \varphi$ iff $\mathcal{K}, s \models \varphi$ for all $s \in I$

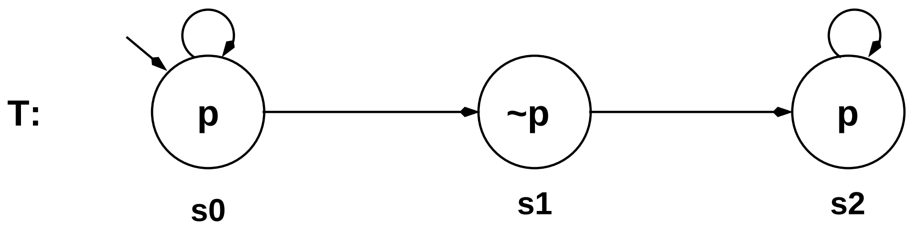
Possibility properties

$\mathbf{AG}\,\mathbf{EF}$ *init*   home state, resettability

Linear vs. branching time

Incomparable expressiveness of LTL and CTL

- LTL cannot express possibility properties
- CTL cannot express $\mathbf{F}\,\mathbf{G}\,p$



$\mathcal{K} \models \mathbf{F}\,\mathbf{G}\,p$ $\qquad$ $\mathcal{K} \not\models \mathbf{AF}\,\mathbf{AG}\,p$

- implications on complexity of model checking

Choose your logic depending on problem requirements

More expressive logics: CTL*, μ-calculus

Model-checking LTL I

The automata-theoretic approach

Büchi automata

Finite automata operating on ω-words $\quad \mathcal{B} = (Q, I, \delta, F)$

$$\left.\begin{array}{ll} Q & \text{finite set of states} \\ I \subseteq Q & \text{initial states} \\ \delta \subseteq Q \times \Sigma \times Q & \text{transition relation} \\ F \subseteq Q & \text{accepting states} \end{array}\right\} \quad \text{same structure as finite automaton}$$

Run of $\mathcal{B}$ on ω-word $a_0 a_1 \ldots \in \Sigma^\omega$

sequence $\quad q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \cdots$

initialization $\quad q_0 \in I$

consecution $\quad (q_i, a_i, q_{i+1}) \in \delta$ for all $i \in \mathbb{N}$

accepting $\quad q_i \in F$ for infinitely many $i \in \mathbb{N}$

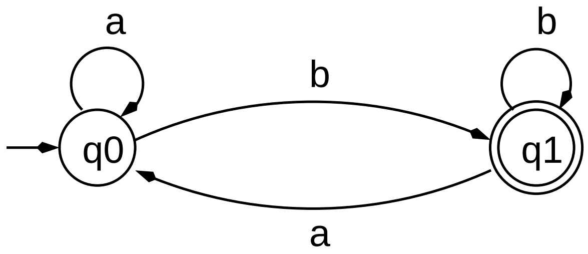
ω-language defined by $\mathcal{B}$

$\mathcal{L}(\mathcal{B}) = \{w \in \Sigma^\omega : \mathcal{B}$ has some accepting run on $w\}$

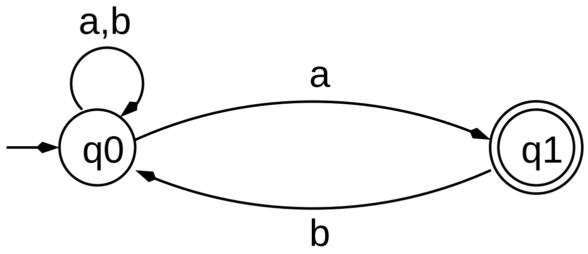
ω-regular languages $\quad$ class of (ω-)languages definable by Büchi automata

Büchi automata: examples

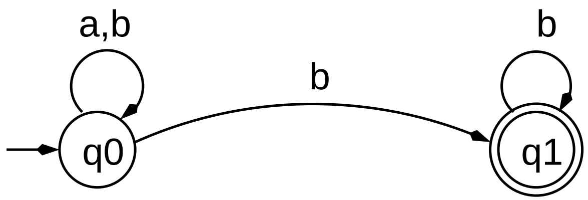
infinitely often ‘b’



infinitely often ‘ab’



eventually only ‘b’



not definable by deterministic Büchi automaton

Büchi automata: basic properties

Decidability of emptiness problem

$\mathcal{L}(\mathcal{B}) \neq \emptyset$ iff exist $q_0 \in I, q \in F$ such that $q_0 \stackrel{\Sigma^*}{\Longrightarrow} q \stackrel{\Sigma^+}{\Longrightarrow} q$

complexity linear in $|Q|$ (NLOGSPACE)

Closure properties

- union standard NFA construction
- intersection “marked” product
- complement difficult construction $O(2^{n \log n})$ states
- projection $\Sigma \to \Sigma'$

Other kinds of ω-automata

Generalized Büchi automata $\mathcal{B} = (Q, I, \delta, \{F_1, \ldots, F_n\})$

- run accepting iff infinitely many $q_i \in F_k$, for all k
- can be coded as a Büchi automaton with additional counter $\pmod n$
- intersection definable via product automaton

Muller automata $\mathcal{M} = (Q, I, \delta, \mathcal{F})$

- run accepting iff set of states attained infinitely often $\in \mathcal{F}$
- special case: Streett automata, can be exponentially more succinct than Büchi automata

Alternating automata

- transition relation $\delta \subseteq Q \times \Sigma \times 2^Q$
- several states can be simultaneously active
- unifying framework for encoding linear-time and branching-time logics

From LTL to (generalized) Büchi automata

Basic insight

- Let $\mathcal{L}(\varphi)$ be the set of sequences of labels satisfying ϕ
- Construct automaton $\mathcal{B}_\varphi$ that accepts precisely $\mathcal{L}(\varphi)$

  Alphabet of $\mathcal{B}_\varphi$ is 2^{AP}

Idea of construction

states	sets of "subformulas" of φ promised to be true
initial states	states containing φ
transition relation	ensures satisfaction of non-temporal formulas in source state
	replaces temporal formulas in source by others in target
	temporal formulas decomposed according to recursion laws
accepting states	defined from "eventualities" $\mathbf{F}\,\varphi$ or φ **until** ψ

$$\begin{aligned} \mathbf{G}\,\varphi &\equiv \varphi \wedge \mathbf{X}\,\mathbf{G}\,\varphi \\ \mathbf{F}\,\varphi &\equiv \varphi \vee \mathbf{X}\,\mathbf{F}\,\varphi \\ \varphi \textbf{ until } \psi &\equiv \psi \vee (\varphi \wedge \mathbf{X}(\varphi \textbf{ until } \psi)) \end{aligned}$$

Example: $\mathbf{G}(p \Rightarrow \mathbf{F}\, q)$

Subformulas

$$\{(p \Rightarrow \mathbf{F}\, q) \wedge \mathbf{X}\, \mathbf{G}(p \Rightarrow \mathbf{F}\, q), p \Rightarrow \mathbf{F}\, q, \mathbf{X}\, \mathbf{G}(p \Rightarrow \mathbf{F}\, q), \neg p, q, \mathbf{X}\, \mathbf{F}\, q\}$$

Example of state

$$\{(p \Rightarrow \mathbf{F}\, q) \wedge \mathbf{X}\, \mathbf{G}(p \Rightarrow \mathbf{F}\, q), p \Rightarrow \mathbf{F}\, q, \mathbf{X}\, \mathbf{G}(p \Rightarrow \mathbf{F}\, q), \mathbf{X}\, \mathbf{F}\, q\}$$

Promises $p \Rightarrow \mathbf{F}\, q$ by promising p, $\neg q$, and $\mathbf{F}\, q$

Promises $\mathbf{F}\, q$ by promising $\mathbf{X}\, \mathbf{F}\, q$

Transitions labelled by $\{p\}$

Target states must promise $\mathbf{G}(p \Rightarrow \mathbf{F}\, q)$ and $\mathbf{F}\, q$

Example of state

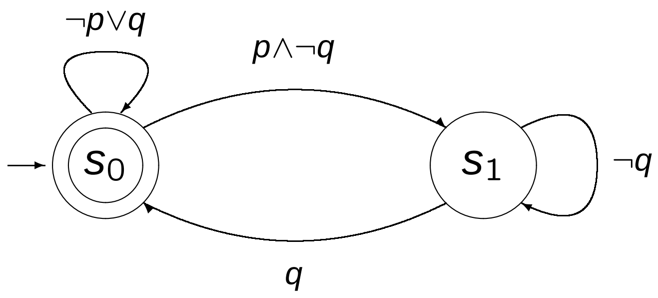
$$\{(p \Rightarrow \mathbf{F}\, q) \wedge \mathbf{X}\, \mathbf{G}(p \Rightarrow \mathbf{F}\, q), p \Rightarrow \mathbf{F}\, q, \mathbf{X}\, \mathbf{G}(p \Rightarrow \mathbf{F}\, q), \neg p\}$$

Promises $p \Rightarrow \mathbf{F}\, q$ by promising $\neg p$ (and $\neg q$)

Transitions labelled by $\{\neg p\}$

Target states must promise $\mathbf{G}(p \Rightarrow \mathbf{F}\, q)$

Result for the example (improved construction)



Complexity

- worst case: $\mathcal{B}_\varphi$ exponential in length of φ
- improved constructions try to avoid exponential blow-up

Application LTL decision procedure

- φ satisfiable iff $\mathcal{L}(\mathcal{B}_\varphi) \neq \emptyset$
- exponential complexity (PSPACE)

Model Checking

Problem Given $\mathcal{K}$ and φ, decide whether $\mathcal{K} \models \varphi$

Automata-theoretic solution

Consider $\mathcal{K}$ as ω-automaton with all states final

Define $\mathcal{L}(\mathcal{K})$ = set of computations of $\mathcal{K}$

$$\boxed{\begin{array}{c} \mathcal{K} \models \varphi \\ \text{iff} \\ \mathcal{L}(\mathcal{K}) \subseteq \mathcal{L}(\varphi) \\ \text{iff} \\ \mathcal{L}(\mathcal{K}) \cap \mathcal{L}(\neg\varphi) = \emptyset \\ \text{iff} \\ \mathcal{L}(\mathcal{K} \times \mathcal{B}_{\neg\varphi}) = \emptyset \end{array}}$$

Complexity $O(|\mathcal{K}| \cdot |\mathcal{B}_{\neg\varphi}|) = O(|\mathcal{K}| \cdot 2^{|\varphi|})$

State explosion

$\mathcal{K} \times \mathcal{B}_{\neg\varphi}$ is too big to be computed effectively

Problems start around 10^6 states

Solutions

- Reduce: ignore irrelevant portions of $\mathcal{K} \times \mathcal{B}_{\neg\varphi}$
- Compress: construct compact representation of $\mathcal{K} \times \mathcal{B}_{\neg\varphi}$
- Abstract: see section on abstraction

Model-checking LTL II

On-the-fly model checking

Partial-order techniques

On-the-fly LTL model checking

Basic insight

- Construct only reachable states of $\mathcal{K} \times \mathcal{B}_{\neg\varphi}$
- Stop if a word in $\mathcal{L}(\mathcal{K} \times \mathcal{B}_{\neg\varphi})$ (acceptance cycle) is found

Setup

- Consider pairs (s, q) of states of $\mathcal{K}$ and $\mathcal{B}_{\neg\varphi}$

  initial pairs both components initial

  successors joint execution of $\mathcal{K}$ and $\mathcal{B}_{\neg\varphi}$

  accepting pairs second component accepting for $\mathcal{B}_{\neg\varphi}$

"On-the-fly" search for acceptance cycles [Courcoubetis et al 1992]

- depth-first search for accepting pair reachable from itself
- interleave state generation and search for cycle
- stack of pairs whose successors need to be explored (contains counterexample)
- hashtable of pairs already seen (in current search mode)

On-the-fly LTL model checking

```
dfs(boolean search_cycle) {
   p = top(stack);
   foreach (q in successors(p)) {
      if (search_cycle and (q == seed))
         report acceptance cycle and exit;
      if ((q, search_cycle) not in visited) {
         enter (q, search_cycle) into visited;
         push q onto stack;
         dfs(search_cycle);
         if (not search_cycle and (q is accepting)) {
            seed = q; dfs(true);
   }  }  }
   pop(stack);
}
// initialization
visited = emptyset(); stack = emptystack(); seed = null;
foreach initial pair p {
   push p onto stack;
   enter (q, false) into visited;
   dfs(false)
}
```

Partial-order reduction (Petri net view)

Transitions t, u are independent if $({}^\bullet t \cup t^\bullet) \cap ({}^\bullet u \cup u^\bullet) = \emptyset$

Examples

- assignments to different variables of values that do not depend on the other variable
- sending and receiving on a channel that is neither empty nor full

Idea: avoid exploring independent transitions . . .

. . . is correct if the property cannot distinguish their order and
every transition is eventually considered

. . . may lead to exponential reduction in part of system explored

Practical issues

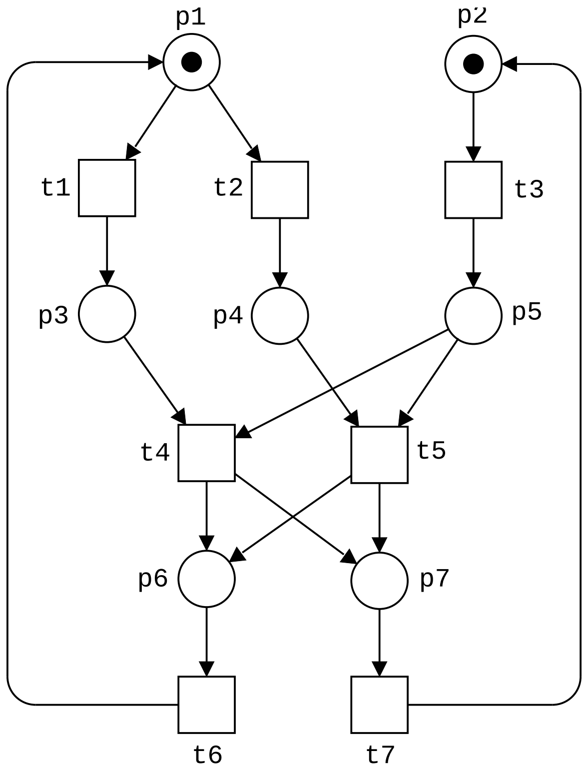
Select at each new state an appropriate subset of the enabled transitions

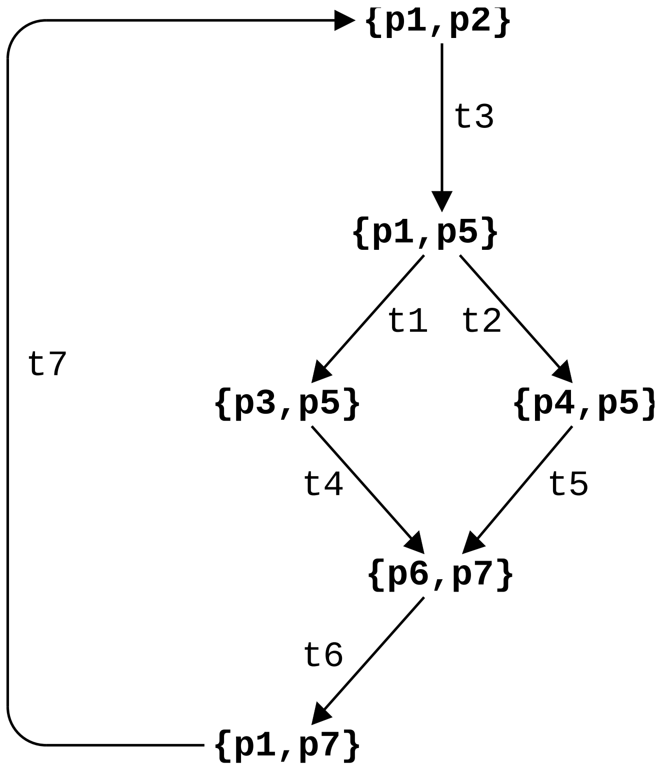
Selecting a smallest subset is NP-complete

Linear and quadratic suboptimal algorithms

Different techniques: stubborn sets, sleep sets, ample sets

Examples





Deadlock freedom can be decided by exploring only six states

Needham-Schroeder: property checked by PROD after examining 942 states (out of 8279)

Partial-order compression (Petri net view)

Based on “true concurrency” theory

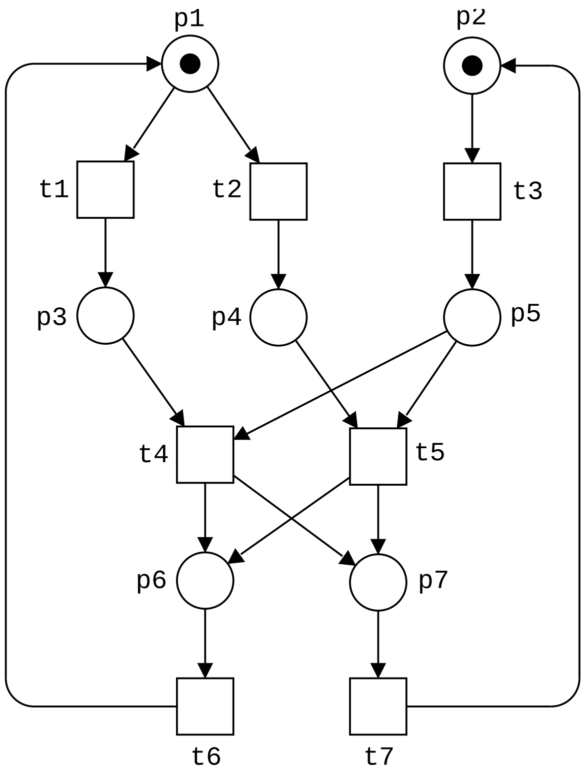
Unfolding of a Petri net

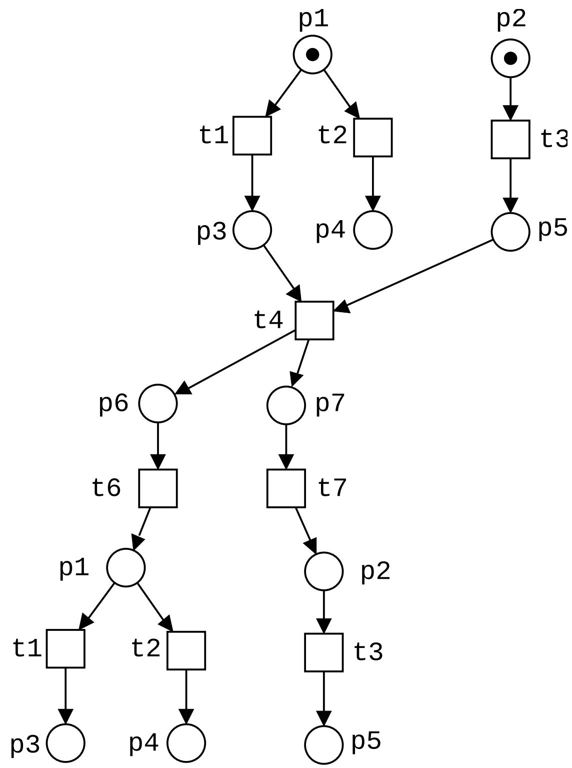
- Obtained through “unrolling”
- Acyclic, possibly infinite net
- Equivalent to the original net for all sensible equivalence notions

Checking procedure for a property φ

- Generate a Petri net $N \times \mathcal{B}_{\neg\varphi}$ with “final places”
- Generate a finite prefix of the unfolding of $N \times \mathcal{B}_{\neg\varphi}$ to decide if $\mathcal{L}(N \times \mathcal{B}_{\neg\varphi}) = \emptyset$
- Prefix can be exponentially more compact than $\mathcal{K} \times \mathcal{B}_{\neg\varphi}$

Examples





Needham-Schroeder: prefix with 3871 events

Model Checking CTL

Basic Algorithms

Binary Decision Diagrams

Computation Tree Logic (CTL)

Branching structure and temporal modalities

type	formula φ	$\mathcal{K}, s_0 \models \varphi$ iff ...
atomic	$p \in AP$	p holds of s_0
propositional	$\neg\varphi$	$\mathcal{K}, s_0 \not\models \varphi$
	$\varphi \vee \psi$	$\mathcal{K}, s_0 \models \varphi$ or $\mathcal{K}, s_0 \models \psi$
temporal	$\mathbf{EX}\,\varphi$	exists path $s_0 s_1 \ldots$ s.t. $\mathcal{K}, s_1 \models \varphi$
	$\mathbf{AF}\,\varphi$	for all paths $s_0 s_1 \ldots$ exists $i \in \mathbb{N}$ s.t. $\mathcal{K}, s_i \models \varphi$
	$\varphi\ \mathbf{EU}\ \psi$	exists path $s_0 s_1 \ldots$ and $i \in \mathbb{N}$ s.t. $\mathcal{K}, s_i \models \psi$ and $\mathcal{K}, s_j \models \varphi$ for all $0 \leq j < i$
	$\mathbf{AX}\,\varphi$, $\mathbf{EF}\,\varphi$, ...	similar

invariants $\mathbf{AG}\,\neg(crit_1 \wedge crit_2)$

home state, resettability $\mathbf{AG}\,\mathbf{EF}\,reset$

CTL model checking

Idea: label states with formulas they satisfy

Recall system validity:

$$\mathcal{K} \models \varphi \quad \text{iff} \quad \mathcal{K}, s \models \varphi \quad \text{for all } s \in I$$

$$\text{iff} \quad I \subseteq [\![\varphi]\!]_{\mathcal{K}}$$

where $[\![\varphi]\!]_{\mathcal{K}} \;=_{\text{def}}\; \{s \in S \mid \mathcal{K}, s \models \varphi\}$

Model checking requires:

– algorithm to compute $[\![\varphi]\!]_{\mathcal{K}}$

– data structures to represent and manipulate sets of states

Bottom-up calculation of $[\![\varphi]\!]_{\mathcal{K}}$

Observation: all CTL formulas definable from $\mathbf{EX}$, $\mathbf{EG}$, and $\mathbf{EU}$, e.g.

$$\mathbf{AX}\,\varphi \;\equiv\; \neg\,\mathbf{EX}\,\neg\varphi \qquad\qquad \mathbf{EF}\,\varphi \;\equiv\; \textbf{true}\;\mathbf{EU}\;\varphi$$

$$\mathbf{AG}\,\varphi \;\equiv\; \neg\,\mathbf{EF}\,\neg\varphi \qquad\qquad \mathbf{AF}\,\varphi \;\equiv\; \neg\,\mathbf{EG}\,\neg\varphi$$

simple cases: reformulation of CTL semantics

$$[\![p]\!]_{\mathcal{K}} = \{s \in S \mid p \in L(s)\} \quad \text{for } p \in AP$$

$$[\![\neg\psi]\!]_{\mathcal{K}} = S \setminus [\![\psi]\!]_{\mathcal{K}}$$

$$[\![\psi_1 \vee \psi_2]\!]_{\mathcal{K}} = [\![\psi_1]\!]_{\mathcal{K}} \cup [\![\psi_2]\!]_{\mathcal{K}}$$

$$[\![\mathbf{EX}\,\psi]\!]_{\mathcal{K}} = \delta^{-1}([\![\psi]\!]_{\mathcal{K}}) \;=_{\text{def}}\; \{\, s \in S \mid t \in [\![\psi]\!]_{\mathcal{K}} \text{ for some } t \text{ s.t. } (s,t) \in \delta \,\}$$

missing cases: $[\![\mathbf{EG}\,\varphi]\!]_{\mathcal{K}}$, $[\![\varphi\;\mathbf{EU}\;\psi]\!]_{\mathcal{K}}$

Calculation of $[\![\mathbf{EG}\,\varphi]\!]_{\mathcal{K}}$

Observe recursion law

$$\mathbf{EG}\,\varphi \quad \equiv \quad \varphi \wedge \mathbf{EX}\,\mathbf{EG}\,\varphi$$

In fact:

$[\![\mathbf{EG}\,\varphi]\!]_{\mathcal{K}}$ is the greatest "solution" of $X = [\![\varphi]\!]_{\mathcal{K}} \cap \delta^{-1}(X)$ in $(2^S, \subseteq)$

Proof.

- Recursion law implies that $[\![\mathbf{EG}\,\varphi]\!]_{\mathcal{K}}$ is a solution.
- Assume $M = [\![\varphi]\!]_{\mathcal{K}} \cap \delta^{-1}(M)$ for $M \subseteq S$, show $M \subseteq [\![\mathbf{EG}\,\varphi]\!]_{\mathcal{K}}$. Assume $s_0 \in M$.

  1. $s_0 \in [\![\varphi]\!]_{\mathcal{K}}$ implies $\mathcal{K}, s_0 \models \varphi$.
  2. $s_0 \in \delta^{-1}(M)$ implies there is $s_1 \in M$ s.t. $(s_0, s_1) \in \delta$.

  Inductively obtain path $s_0, s_1, \ldots$ of states satisfying φ.

  This proves $\mathcal{K}, s_0 \models \mathbf{EG}\,\varphi$ and thus $s_0 \in [\![\mathbf{EG}\,\varphi]\!]_{\mathcal{K}}$.

Calculation of fixed point

Kleene's fixed point theorem implies:

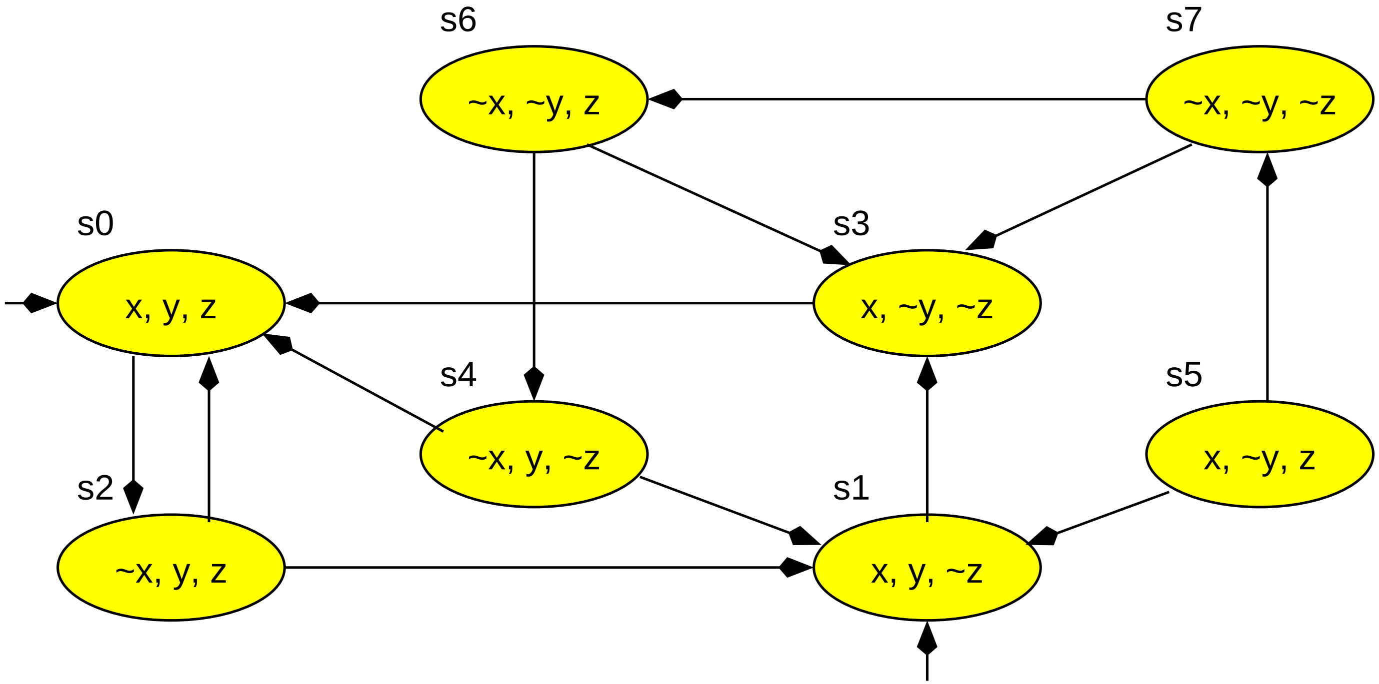
$[\![\mathbf{EG}\,\varphi]\!]_{\mathcal{K}}$ can be computed as the limit of

$$S,\ \ \pi(S),\ \ \pi(\pi(S)),\ \ \ldots \qquad \text{for} \quad \pi : \begin{cases} 2^S \to 2^S \\ X \mapsto [\![\varphi]\!]_{\mathcal{K}} \cap \delta^{-1}(X) \end{cases}$$

Convergence: obvious, because S is finite

Computation of greatest fixed point (1)

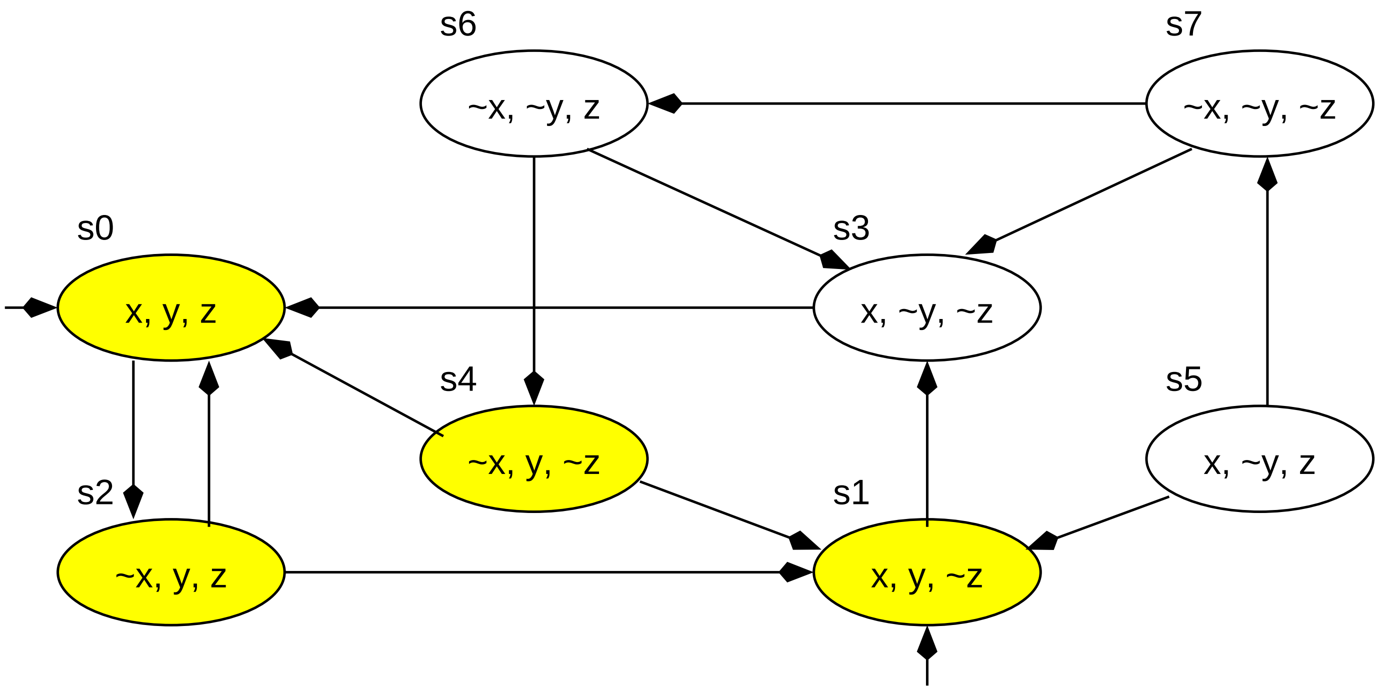
Compute $[\![\mathbf{EG}\, y]\!]$



$\pi^0(S) \;=\; S$

Computation of greatest fixed point (2)

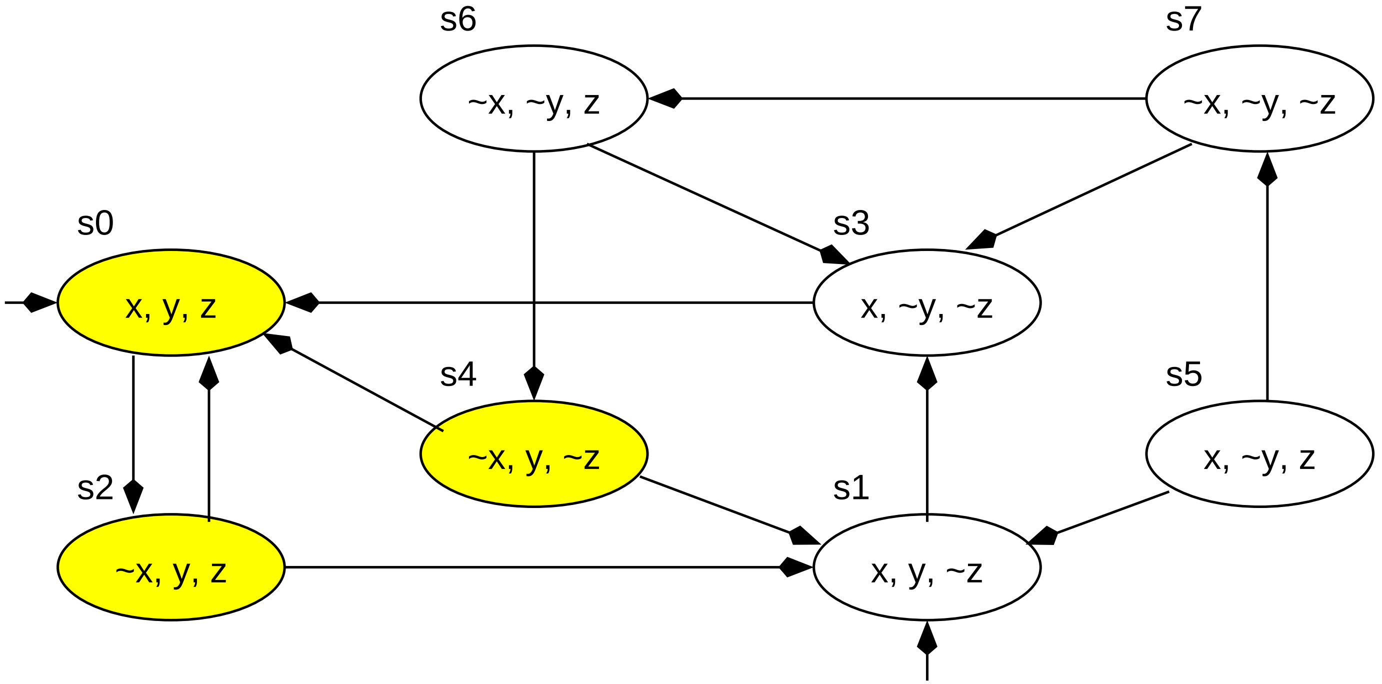
Compute $[\![\mathbf{EG}\, y]\!]$



$$\pi^1(S) \;=\; [\![y]\!]_{\mathcal{K}} \cap \delta^{-1}(S)$$

Computation of greatest fixed point (3)

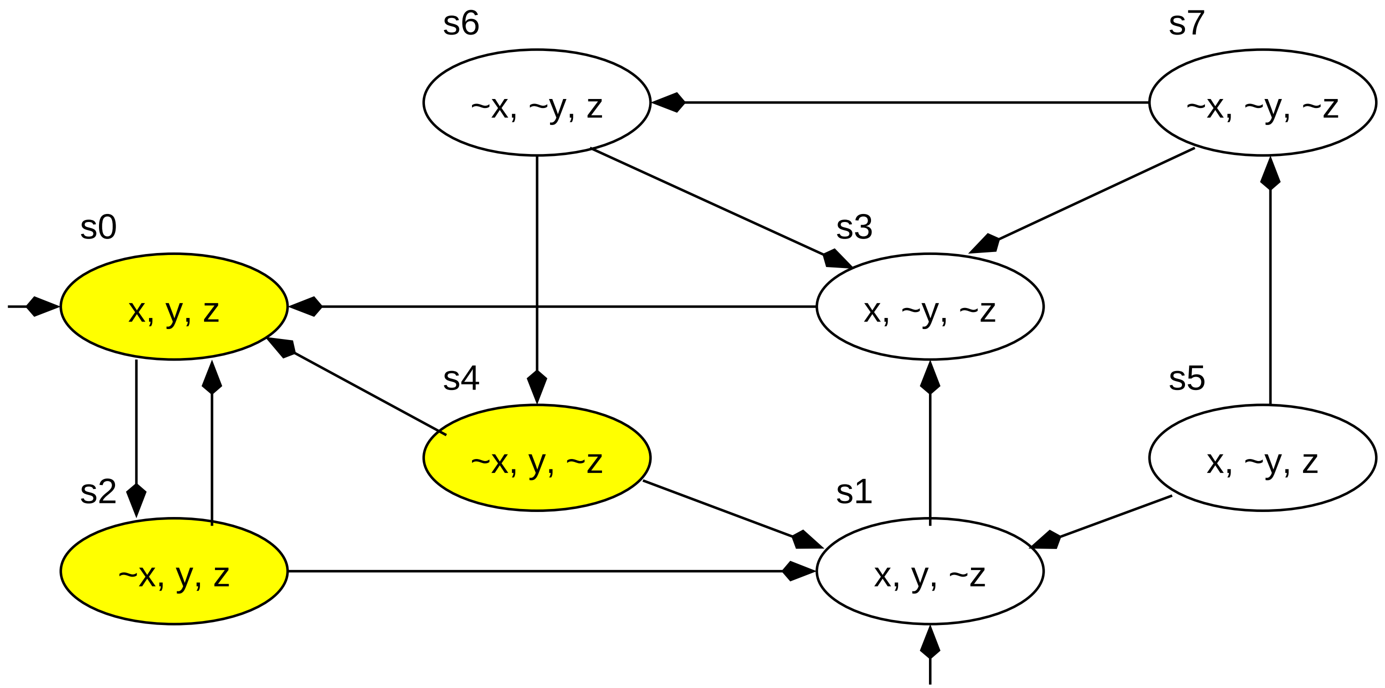
Compute $[\![\mathbf{EG}\, y]\!]$



$$\pi^2(S) \;=\; [\![y]\!]_{\mathcal{K}} \cap \delta^{-1}(\pi^1(S))$$

Computation of greatest fixed point (4)

Compute $[\![\mathbf{EG}\, y]\!]$



$\pi^3(S) \;=\; [\![y]\!]_{\mathcal{K}} \cap \delta^{-1}(\pi^2(S)) \;=\; \pi^2(S)$: $\quad [\![\mathbf{EG}\, y]\!]_{\mathcal{K}} = \{s_0, s_2, s_4\}$

Calculation of $[\![\varphi\ \mathbf{EU}\ \psi]\!]_{\mathcal{K}}$

Similarly:

$$\varphi\ \mathbf{EU}\ \psi \quad \equiv \quad \psi \vee (\varphi \wedge \mathbf{EX}(\varphi\ \mathbf{EU}\ \psi))$$

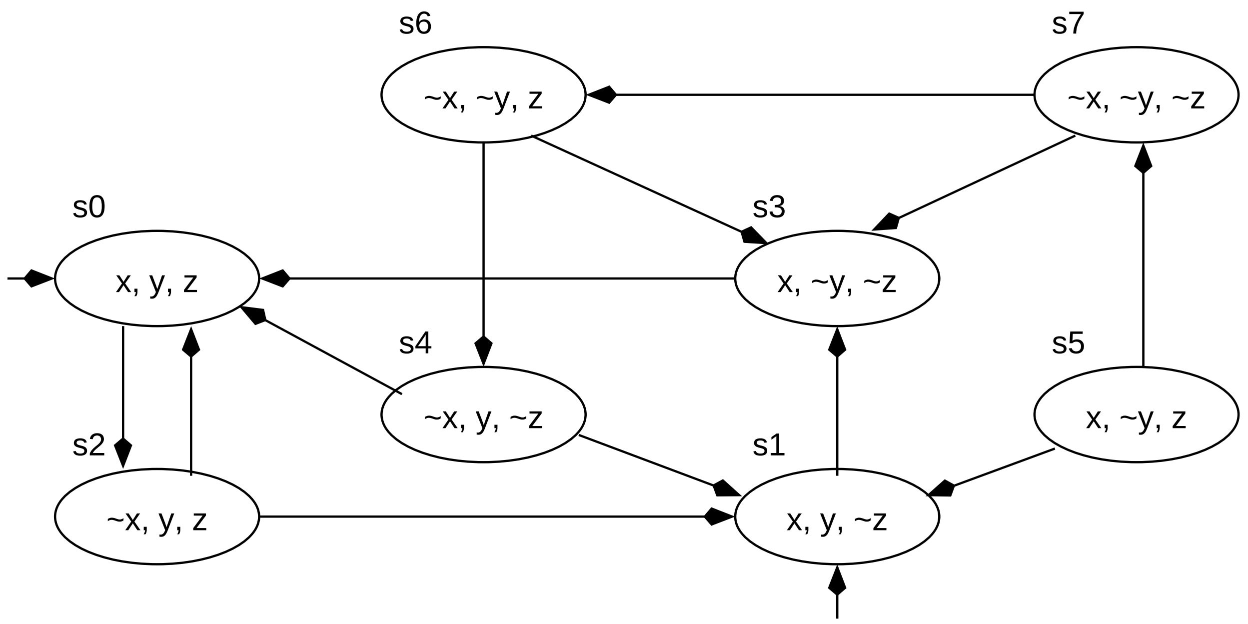
$[\![\varphi\ \mathbf{EU}\ \psi]\!]_{\mathcal{K}}$ is the smallest solution of $X \;=\; [\![\psi]\!]_{\mathcal{K}} \cup ([\![\varphi]\!]_{\mathcal{K}} \cap \delta^{-1}(X))$

Computation: calculate the limit of

$$\emptyset,\ \ \pi(\emptyset),\ \ \pi(\pi(\emptyset)),\ \ \ldots \qquad \text{for} \quad \pi : \begin{cases} 2^S \rightarrow 2^S \\ X \mapsto [\![\psi]\!]_{\mathcal{K}} \cup ([\![\varphi]\!]_{\mathcal{K}} \cap \delta^{-1}(X)) \end{cases}$$

Computation of least fixed point (1)

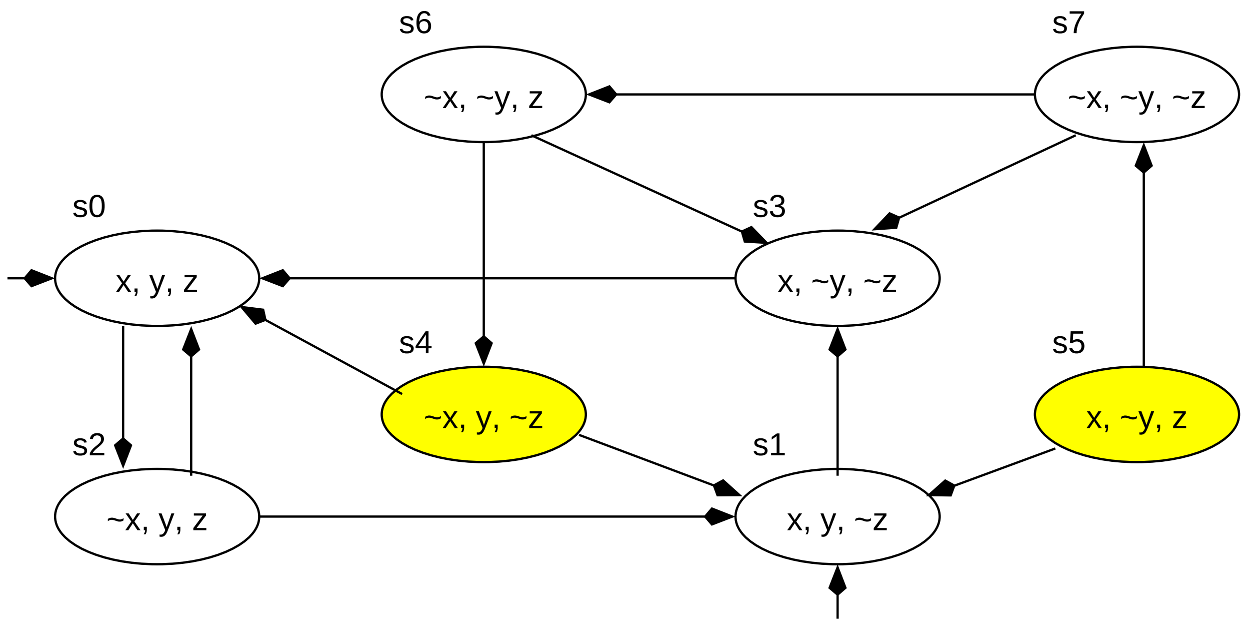
Compute $[\![\mathbf{EF}((x = z) \wedge (x \neq y))]\!]_{\mathcal{K}}$



$$\pi^0(\emptyset) \;=\; \emptyset$$

Computation of least fixed point (2)

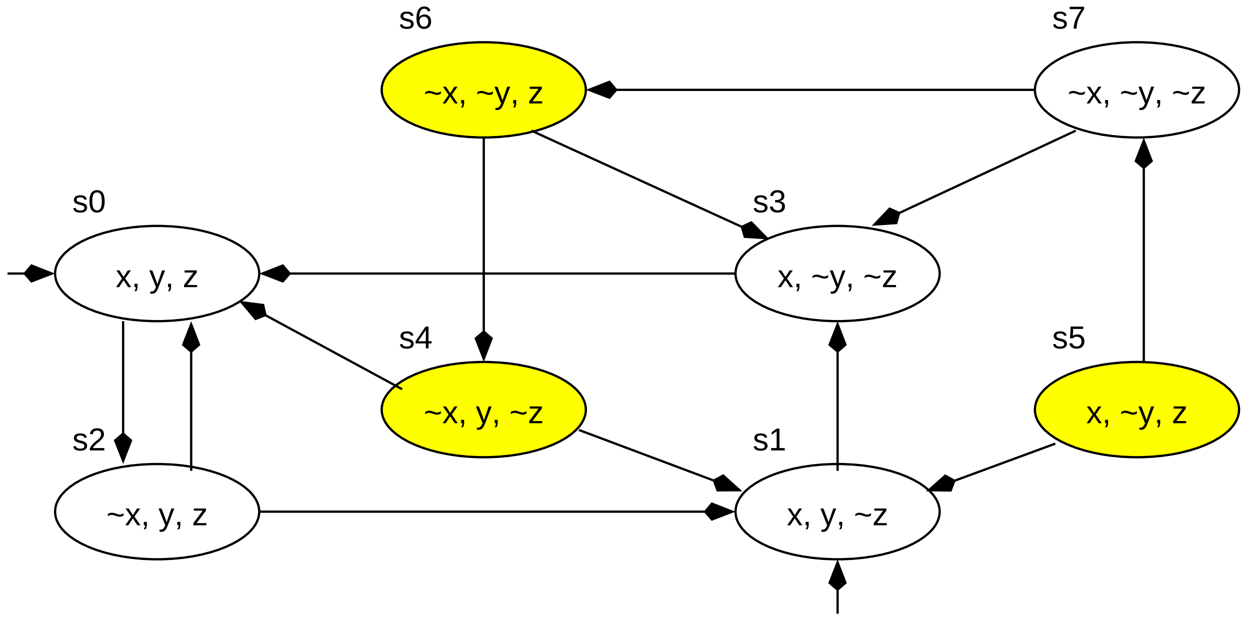
Compute $[\![\mathbf{EF}((x = z) \wedge (x \neq y))]\!]$



$$\pi^1(\emptyset) \quad = \quad [\![(x = z) \wedge (x \neq y)]\!]_{\mathcal{K}} \cup \delta^{-1}(\emptyset)$$

Computation of least fixed point (3)

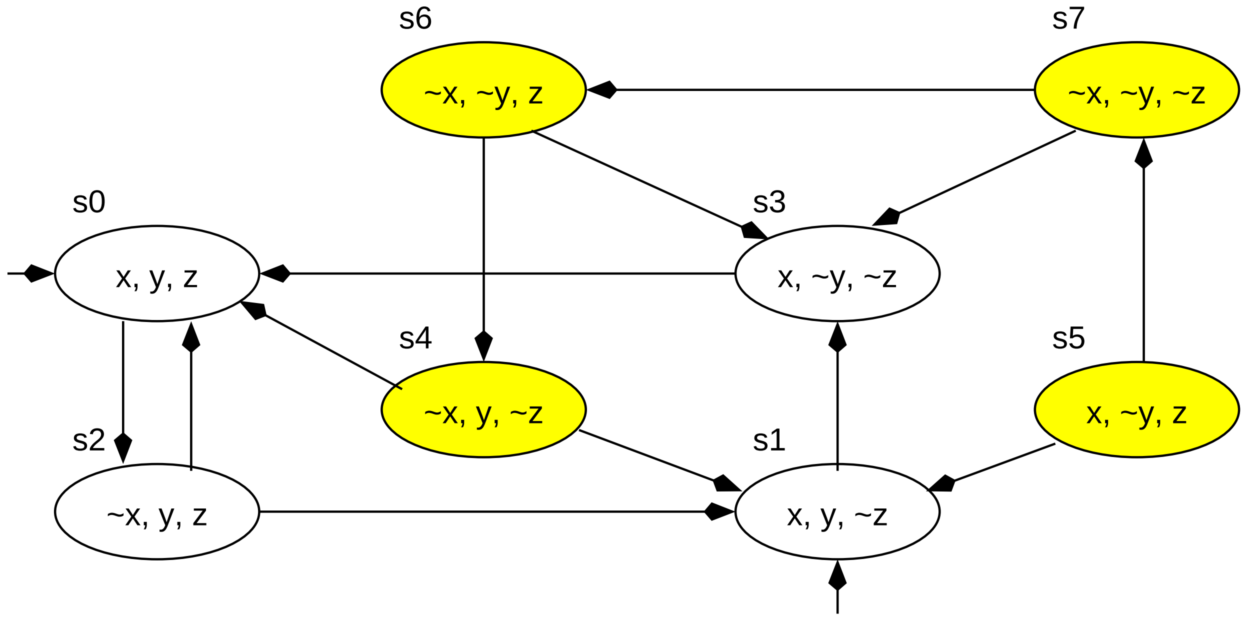
Compute $[\![\mathbf{EF}((x = z) \wedge (x \neq y))]\!]_{\mathcal{K}}$



$$\pi^2(\emptyset) \;=\; [\![(x = z) \wedge (x \neq y)]\!]_{\mathcal{K}} \cup \delta^{-1}(\pi^1(\emptyset))$$

Computation of least fixed point (4)

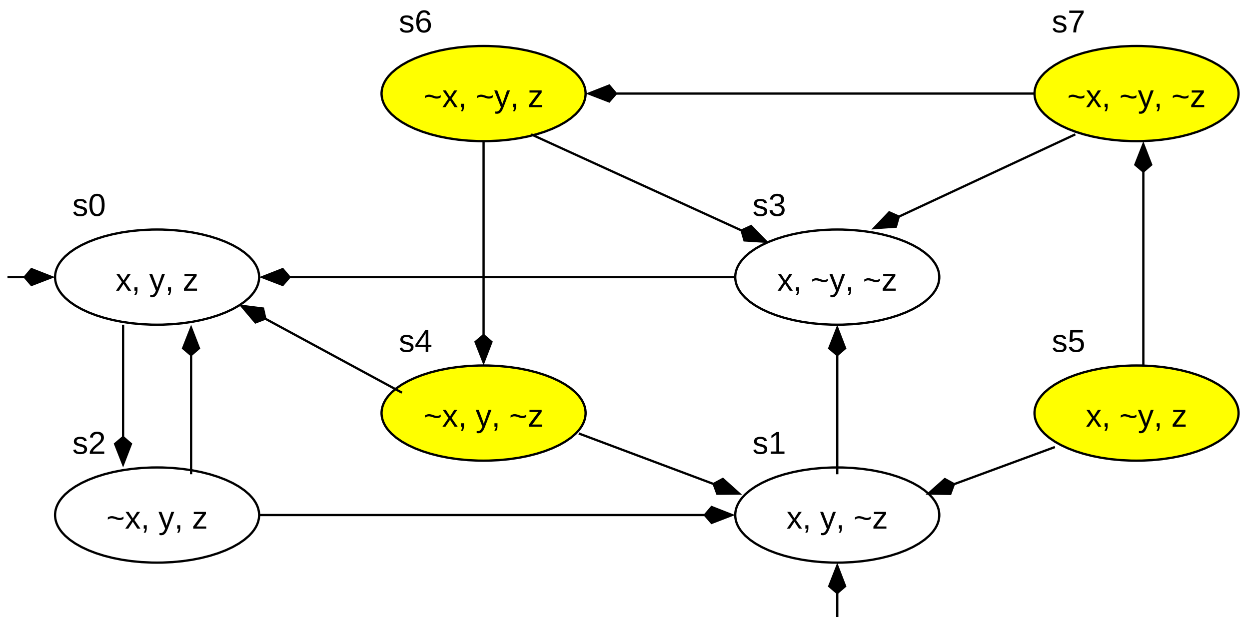
Compute $[\![\mathbf{EF}((x = z) \land (x \neq y))]\!]_{\mathcal{K}}$



$$\pi^3(\emptyset) \quad = \quad [\![(x = z) \land (x \neq y)]\!]_{\mathcal{K}} \cup \delta^{-1}(\pi^2(\emptyset))$$

Computation of least fixed point (5)

Compute $[\![\mathbf{EF}((x = z) \wedge (x \neq y))]\!]_{\mathcal{K}}$



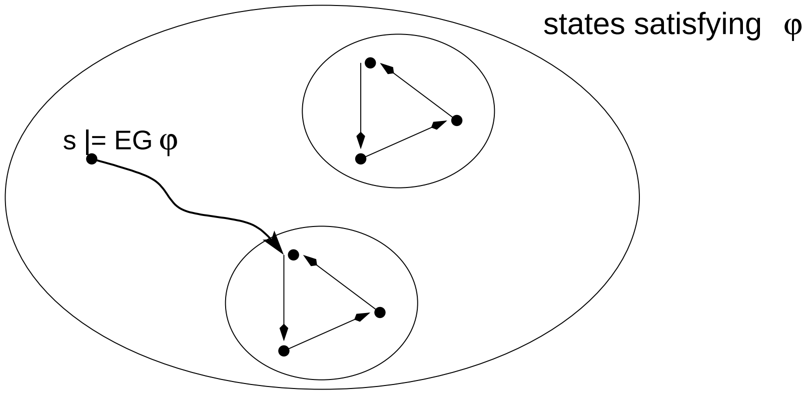
$$\pi^4(\emptyset) \;=\; [\![(x = z) \wedge (x \neq y)]\!]_{\mathcal{K}} \cup \delta^{-1}(\pi^3(\emptyset)) \;=\; \pi^3(\emptyset):$$

$$[\![\mathbf{EF}((x = z) \wedge (x \neq y))]\!]_{\mathcal{K}} = \{s_4, s_5, s_6, s_7\}$$

Complexity issues

Complexity of fixed point algorithm: $O(|\varphi| \cdot |S| \cdot (|S| + |\delta|))$

Improved algorithm [Clarke, Emerson, Sistla 1986]



– Computation of $[\![\mathbf{EG}\,\varphi]\!]_{\mathcal{K}}$

1. restrict $\mathcal{K}$ to states satisfying φ
2. compute SCCs of restricted graph
3. find states from which some SCC is reachable, using backward search

– Computation of $[\![\varphi\ \mathbf{EU}\ \psi]\!]_{\mathcal{K}}$ can similarly be reduced to backward search

Complexity: $O(|\varphi| \cdot (|S| + |\delta|))$ linear in size of model and formula

Fairness constraints

Recall limited expressiveness of CTL: fairness conditions not expressible

Instead: modify semantics and model checking algorithm

FairCTL: exclude "unfair" paths, e.g.

$\mathcal{K}, s_0 \models \mathbf{EG}_\mathsf{f}\, \varphi$ iff there exists fair path $s_0, s_1, \ldots$ s.t. $\mathcal{K}, s_i \models \varphi$ for all i

$\mathcal{K}, s_0 \models \mathbf{AG}_\mathsf{f}\, \varphi$ iff $\mathcal{K}, s_i \models \varphi$ holds for all fair paths $s_0, s_1, \ldots$ and all i

Fairness conditions specified by additional constraints

SMV: indicate CTL formulas that must hold infinitely often along a fair path

Key property: suffix closure

path $s_0, s_1, s_2, \ldots$ is fair iff $s_n, s_{n+1}, s_{n+2}, \ldots$ is fair (for all n)

Model checking FairCTL

Observe: $\mathbf{EG_f\,true}$ holds at s iff there is some fair path from s

Suffix closure ensures

$$\begin{aligned} \mathbf{EX_f}\,\varphi &\equiv \mathbf{EX}(\varphi \wedge \mathbf{EG_f\,true}) \\ \varphi\,\mathbf{EU_f}\,\psi &\equiv \varphi\,\mathbf{EU}\,(\psi \wedge \mathbf{EG_f\,true}) \end{aligned}$$

Therefore: need only modify algorithm to compute $[\![\mathbf{EG_f}\,\varphi]\!]_{\mathcal{K}}$

assume k SMV-style fairness constraints: $\psi_1 \wedge \ldots \wedge \psi_k$

1. restrict $\mathcal{K}$ to states satisfying φ
2. compute SCCs of restricted graph
3. remove SCCs that do not contain a state satisfying ψ_j, for some j
4. $[\![\mathbf{EG_f}\,\varphi]\!]_{\mathcal{K}}$ consists of states from which some (fair) SCC is reachable

Complexity: $O(|\varphi| \cdot (|S| + |\delta|) \cdot k)$ still linear in the size of the model

Symbolic CTL model checking

Compress: data structures for model checking algorithm

compact representation of sets $[\![\varphi]\!]_{\mathcal{K}} \subseteq S$ and relation $\delta \subseteq S \times S$

Operations required

- Boolean operations on sets — union, intersection, complement
- inverse image operation — $\delta^{-1}(M)$
- comparison — detect termination of fixed point computation

BDDs (binary decision diagrams) [Bryant 1986]

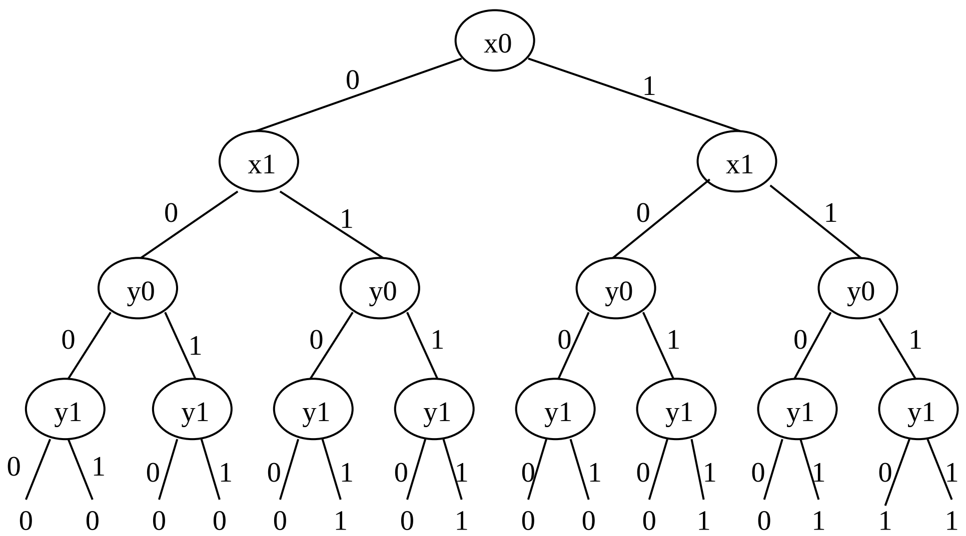
- widely used data structure for boolean functions
- compact, canonical dag representation of binary decision trees
- can represent large sets of regular structure

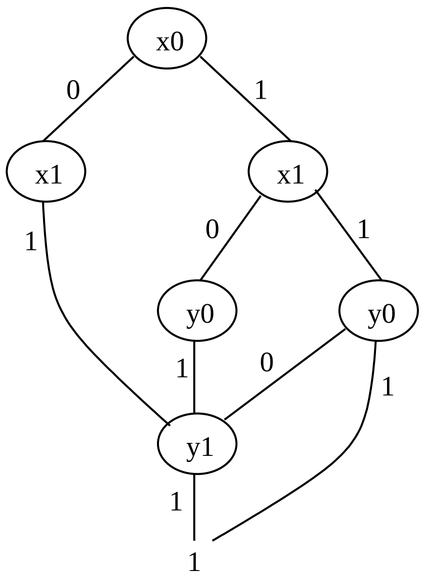
Compact set representations

Assume states are valuations of Boolean variables x_0, x_1, y_0, y_1

Example: set of states such that sum $x_1x_0 \oplus y_1y_0$ produces carry

- explicit enumeration $\{\overline{x_0}x_1\overline{y_0}y_1, \overline{x_0}x_1y_0y_1, x_0\overline{x_1}y_0y_1, x_0x_1\overline{y_0}y_1, x_0x_1y_0\overline{y_1}, x_0x_1y_0y_1\}$
- decision tree set elements correspond to paths leading to 1
- BDD dag obtained by removing redundant nodes and combining equal subtrees



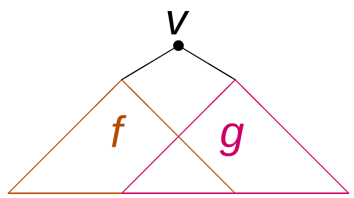


BDD implementation

Constructors

– constant BDDs **true**, **false** $\boxed{1}$ $\boxed{0}$

– inner nodes $BDD(v, f, g)$



Observe global invariants:

– along any path, variables occur in same order (if at all)

– subdags of inner node are always distinct

– avoid reallocation of equivalent BDD nodes (use hash table)

Therefore:

– BDD uniquely determined by Boolean function

– equivalence checking reduces to testing pointer equality

Boolean operations for BDDs

basic operation $\quad ite(f, g, h) \;=\; (f \wedge g) \vee (\neg f \wedge h)$ $\quad$ "if _ then _ else _"

all Boolean connectives definable from *ite* and constants

recursive computation

$$ite(\mathbf{true}, g, h) \;=\; g \qquad ite(\mathbf{false}, g, h) = h$$

Else: let v be "smallest" variable in f, g, h

$$\begin{aligned} ite(f, g, h) \;&=\; v \;\wedge\; ite(f|_{v=\mathbf{true}}, g|_{v=\mathbf{true}}, h|_{v=\mathbf{true}}) \\ &\qquad\qquad \vee \\ &\quad\; \neg v \;\wedge\; ite(f|_{v=\mathbf{false}}, g|_{v=\mathbf{false}}, h|_{v=\mathbf{false}}) \\ &=\; \begin{cases} ite(f|_{v=\mathbf{true}}, \ldots) & \text{if } ite(f|_{v=\mathbf{true}}, \ldots) = ite(f|_{v=\mathbf{false}}, \ldots) \\ BDD(v, ite(f|_{v=\mathbf{true}}, \ldots), ite(f|_{v=\mathbf{false}}, \ldots)) & \text{otherwise} \end{cases} \end{aligned}$$

Cofactor $\quad f|_{v=\mathbf{true}}, f|_{v=\mathbf{false}}$ $\quad$ for v at most head variable of f

equals left or right sub-dag of f if v is head variable, otherwise equals f

Complexity: $\quad O(|f| \cdot |g| \cdot |h|)$ $\quad$ if recomputation is avoided by hashing

BDD implementation: quantifiers

projection $(\exists x : \varphi) \; = \; (\varphi|_{x=\mathbf{true}} \vee \varphi|_{x=\mathbf{false}})$

quantification over head variable

$$
\begin{aligned}
& \exists x : BDD(x, f, g) \\
= \; & \exists x : (x \wedge f) \vee (\neg x \wedge g) && \text{[Def. BDD]} \\
= \; & (true \wedge f) \vee (\neg true \wedge g) \vee (false \wedge f) \vee (\neg false \wedge g) && \text{[note: } x \text{ does not occur in } f, g\text{]} \\
= \; & f \vee g
\end{aligned}
$$

general case: quantification over several variables

$$
\exists \mathbf{x} : BDD(y, f, g) \; = \; \begin{cases} BDD(y, \; \exists \mathbf{x} : f, \; \exists \mathbf{x} : g) & \text{if } y \notin \mathbf{x} \\ (\exists \mathbf{x} : f) \vee (\exists \mathbf{x} : g) & \text{otherwise} \end{cases}
$$

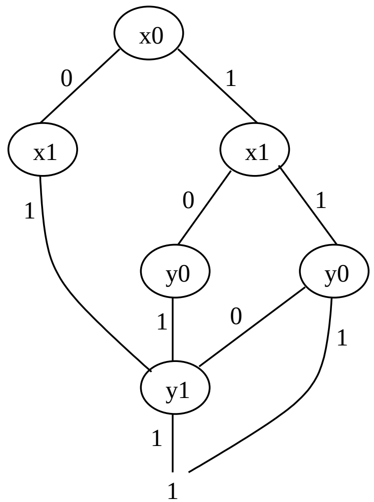
universal quantification: similar

Complexity: worst case exponential, but usually works well in practice

BDDs: variable ordering

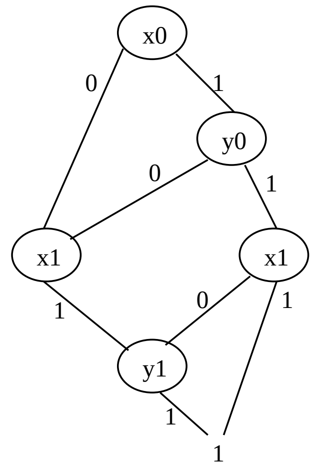
Variable ordering can drastically affect BDD sizes

example:



exponential growth in *n*

vs.



linear growth in *n*

determining optimal variable ordering is NP-hard

Heuristics

- manual ordering cluster dependent variables
- automatic strategies based on steepest-ascent or similar techniques
- some structures (e.g. multipliers, queues) do not admit compact BDD representation

Symbolic CTL model checking: implementation

Symbolic representation

– state space S	vector of (Boolean) state variables $\mathbf{x}$
– initial states I	BDD over $\mathbf{x}$
– transition relation δ	BDD over $\mathbf{x}, \mathbf{x}'$, perhaps split conjunctively
– sets $[\![\varphi]\!]_{\mathcal{K}}$	BDDs over $\mathbf{x}$

Operations

– set operations	Boolean operations on BDDs
– pre-image	$\delta^{-1}(M) \;=\; \exists \mathbf{x}' : \delta \wedge M'$
– set comparison	pointer comparison

Complexity

can be exponential in size of BDD representing δ

Results

- systems with huge potential state spaces (10^{120} states) have been analysed
- particularly successful for synchronous hardware with short data paths

Abstraction techniques

Basics

Predicate Abstraction

Extensions for liveness

State explosion problem

Exponential increase of reachable states with system size

Partial solutions

- reduce partial-order, symmetry: explore only relevant part of state space
- compress unfoldings, BDDs: efficient data structures

But: 10^{100} potential states are generated by just 300 bits

What about larger systems?

- hardware register files, execution pipelines
- software usually unbounded state size

Ad hoc approach

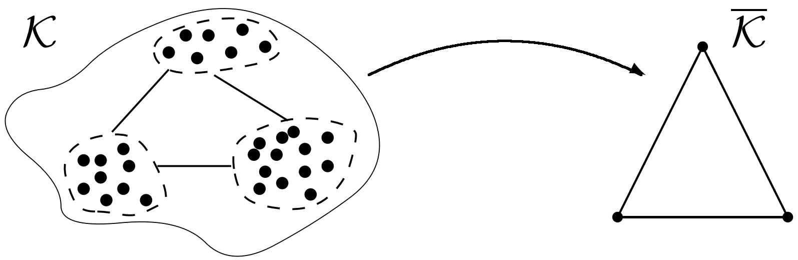
analyse small instances 2 cache lines, 3 potential data values, etc.

How do you make sure that you'll catch the bug?

Abstraction

Idea

- compute “abstract system” $\overline{\mathcal{K}}$ (finite, small)
- infer properties of $\mathcal{K}$ from properties of $\overline{\mathcal{K}}$



$$\mathcal{K} \models \varphi \qquad \Longleftarrow \qquad \overline{\mathcal{K}} \models \overline{\varphi}$$

Issues

– how to obtain and present abstract model?

– full automation or user interaction?

– what if $\overline{\mathcal{K}} \not\models \overline{\varphi}$ (“false negatives”) ?

Predicate abstraction: abstraction determined by predicates over concrete state space

– predicates of interest indicated by the user

– subsumes other abstraction techniques

– intuitive presentation of abstract model

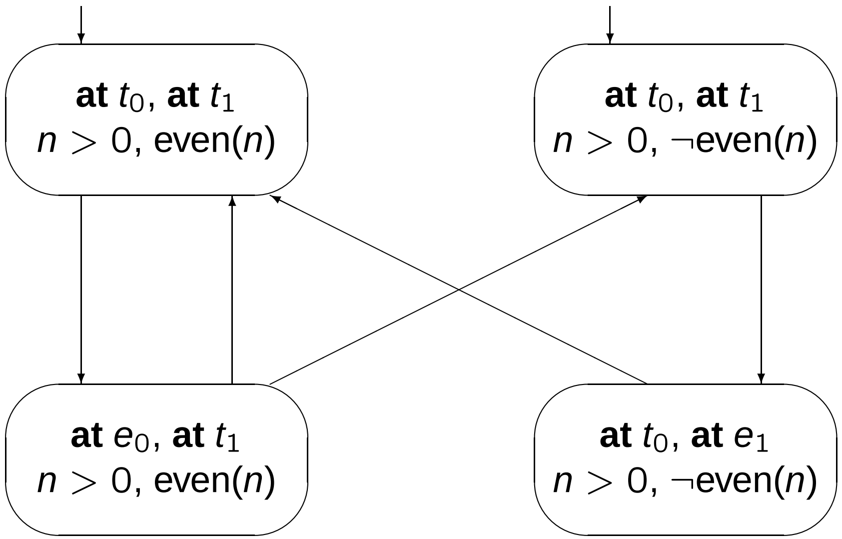
Example: dining mathematicians

mutual exclusion for two processes (synchronization via integer variable n)

int $n > 0$

loop		**loop**
t_0 : **await** even(n);	$\|$	t_1 : **await** $\neg$even(n);
e_0 : $n := n$ div 2		e_1 : $n := 3 * n + 1$
endloop		**endloop**

abstract representation: control state, parity



$\mathbf{G}(n > 0)$

$\mathbf{G}\,\neg(\mathbf{at}\ e_0 \wedge \mathbf{at}\ e_1)$

$\mathbf{G}\,\mathbf{F}\,\mathbf{at}\ e_0$

$\mathbf{G}\,\mathbf{F}\,\mathbf{at}\ e_1$ can not be verified

Predicate diagrams

Fix set AP of atomic propositions

$\overline{AP}$ denotes set of propositions in AP and their negations

Presentation of abstraction as transition system $\overline{\mathcal{A}} = (\overline{S}, \overline{I}, \overline{\delta})$

finite set $\overline{S} \subseteq 2^{\overline{AP}}$ of nodes (let $\overline{s} \in \overline{S}$ also denote conjunction of literals)

Verification conditions for correctness of abstraction

– initialization: initial nodes of $\overline{\mathcal{A}}$ cover initial states of $\mathcal{K}$

$$\bigvee_{\overline{s} \in \overline{I}} \overline{s} \;\Rightarrow\; \bigvee_{s \in I} L(s)$$

– consecution: transitions of $\overline{\mathcal{A}}$ cover possible transitions of $\mathcal{K}$

$(\overline{s}, \overline{t}) \in \overline{\delta}$ if $L(s) \Rightarrow \overline{s}$ and $L(t) \Rightarrow \overline{t}$ for some $(s, t) \in \delta$

Note: extra initial states or transitions preserves correctness

Preservation of properties

Correctness of abstraction implies:

- all computations of $\mathcal{K}$ represented as computations of $\overline{\mathcal{A}}$
- properties of $\mathcal{K}$ can be inferred from those of $\overline{\mathcal{A}}$

  $\overline{\mathcal{A}} \models \varphi \quad \Longrightarrow \quad \mathcal{K} \models \varphi$ for all LTL (actually, ACTL*) formulas φ over *AP*

- $\overline{\mathcal{A}} \models \varphi$ established by model checking: consider atomic propositions as Boolean variables

$\overline{\mathcal{A}}$ may contain additional computations

- $\overline{\mathcal{A}} \not\models \varphi$ need not imply $\mathcal{K} \not\models \varphi$
- counter example often suggests how to improve the abstraction
- spurious loops invalidate liveness properties (cf. “dining mathematicians”)

Strengthening abstractions

- split nodes extend set *AP* of atomic propositions
- break cycles represent information for liveness properties

Generating predicate diagrams (1)

Correct abstraction by elimination

- assume $\mathcal{K}$ being given by initial condition *Init* and transition relation *Next*
- start with full graph over 2^{AP}
- remove node $\overline{s}$ from $\overline{I}$ if $\models \mathit{Init} \Rightarrow \neg\overline{s}$
- remove edge $(\overline{s}, \overline{t})$ from $\overline{\delta}$ if $\models \overline{s} \wedge \mathit{Next} \Rightarrow \neg\overline{t}'$

Implementation: use theorem prover

- try to prove implications using automatic tactic with limited resources
- many “local” goals instead of “global” property
- unproven implications: approximation, perhaps good enough
- drawback: $2^{|AP|}$ states, $2^{2|AP|}$ proof attempts

Optimized implementation in PVS

H. Saïdi and N. Shankar: Abstract and model check while you prove. [CAV’99, LNCS 1633]

Generating predicate diagrams (2)

Compute abstraction by symbolic evaluation

- reduce: generate only reachable abstract states
- compilation approach: borrow from abstract interpretation

Formally: Galois connection

$$\text{sets of states} \; \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} \; \text{Boolean algebra of predicates}$$

Implementation

- rewrite $\overline{s} \wedge \mathit{Next}$ into disjunction $\overline{t_1}' \vee \ldots \vee \overline{t_n}'$ of successor states
- sample rules for “dining mathematicians”

$$\begin{array}{lll} \mathit{even}(x), \mathit{even}(y) \Rightarrow \mathit{even}(x+y) & \mathit{even}(x), \neg\mathit{even}(y) \Rightarrow \neg\mathit{even}(x+y) & \\ x \in \mathit{Nat}, x > 0, \mathit{even}(x) \Rightarrow x \text{ div } 2 > 0 & \mathit{even}(0) & \neg\mathit{even}(1) \end{array}$$

Example: bakery algorithm

Lamport's mutual-exclusion protocol (2 processes, "atomic" version)

int $t_1 = 0$, $t_2 = 0$ (* "queueing tickets" *)

loop		**loop**	
l_1 : "noncritical section";		m_1 : "noncritical section";	
l_2 : $t_1 := t_2 + 1$;		m_2 : $t_2 := t_1 + 1$;	
l_3 : **await** $t_2 = 0 \vee t_1 \leq t_2$;	$\|$	m_3 : **await** $t_1 = 0 \vee \neg(t_1 \leq t_2)$;	
l_4 : "critical section";		m_4 : "critical section";	
l_5 : $t_1 := 0$		m_5 : $t_2 := 0$	
endloop		**endloop**	

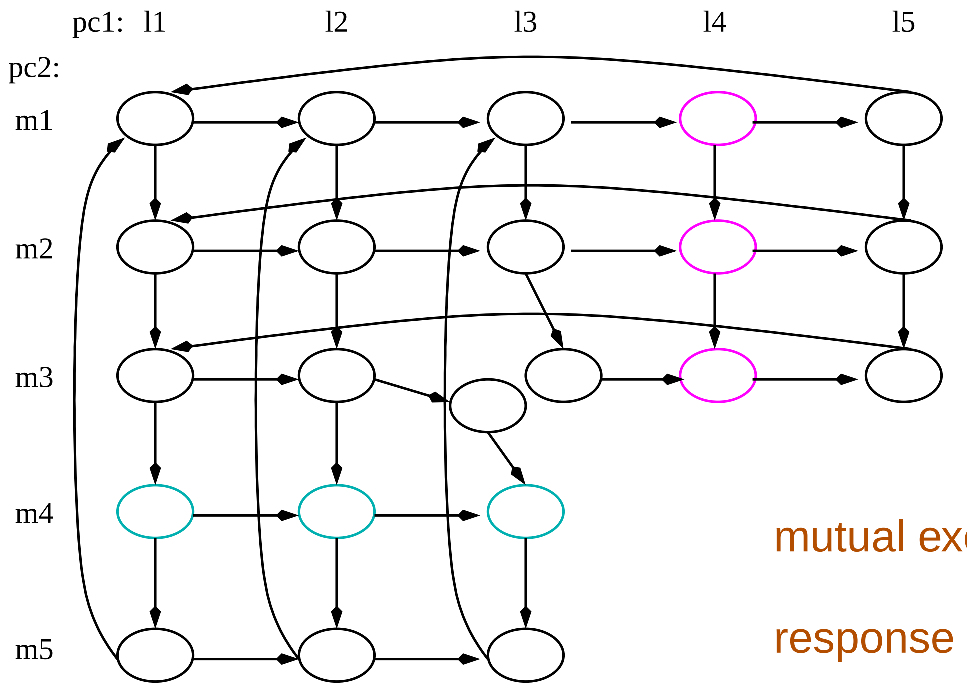
Note: ticket values can grow arbitrarily large

Predicates of interest

- control state
- $t_1 = 0$, $t_2 = 0$, $t_1 \leq t_2$

Bakery: predicate diagram

Symbolic evaluation produces the following diagram (only control state indicated)



mutual exclusion $\mathbf{G}\,\neg(\mathbf{at}\ l_4 \wedge \mathbf{at}\ m_4)$

response $\mathbf{G}(\mathbf{at}\ l_3 \Rightarrow \mathbf{F}\,\mathbf{at}\ l_4)$

precedence

$\mathbf{G}(\mathbf{at}\ l_3 \Rightarrow \neg\mathbf{at}\ m_4\ \mathbf{W}\ \mathbf{at}\ m_4\ \mathbf{W}\ \neg\mathbf{at}\ m_4\ \mathbf{W}\ \mathbf{at}\ l_4)$

all properties verified from single diagram

Predicates on-the-fly

Symbolic evaluation can fail due to insufficient information

Bakery example: computing successors of

$$\overline{n} \quad =_{\text{def}} \quad \{\mathbf{at}\ l_3, \mathbf{at}\ m_3, t_1 \neq 0, t_2 \neq 0\}$$

fails because guard $g \equiv t_1 \leq t_2$ cannot be evaluated

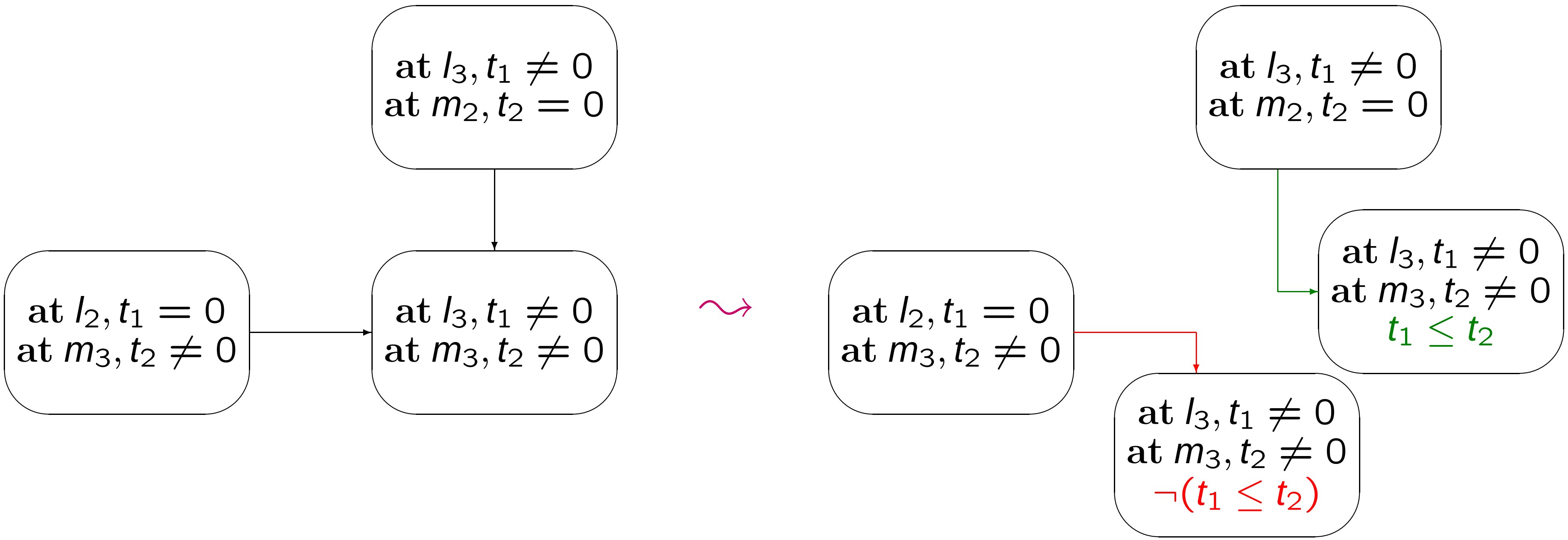
Solution: reconsider predecessors of $\overline{n}$

- for every predecessor $\overline{m}$ in the diagram, try to establish

$$\overline{m} \wedge \mathit{Next} \wedge \overline{n}' \quad \Rightarrow \quad \left\{ \begin{array}{c} g \\ \neg g \end{array} \right\}$$

- add $(\neg)g$ to the node label of $\overline{n}$ as appropriate
- possibly split node $\overline{n}$

Predicates on-the-fly: Bakery example



Predicate $t_1 \leq t_2$ need not be supplied by the user

inferred predicates added precisely where necessary

Strengthening for liveness

Boolean abstractions often cannot prove liveness properties

- predicate diagram usually contains cycles that do not correspond to “concrete” computations
- “dining mathematicians” example: liveness for process 1 could not be verified

Standard techniques to establish liveness properties

– fairness conditions	action taken infinitely often if sufficiently often enabled
– well-founded orderings	exclude cycles that correspond to infinite descent

These need to be represented in the abstraction!

Representing fairness conditions

Annotate (some) transitions in $\overline{\delta}$ with actions $A \in Act$

- formally, transitions are now triples $\overline{\delta} \subseteq \overline{S} \times Act \times \overline{S}$
- assume actions are described by characteristic predicate over $(\mathbf{x}, \mathbf{x}')$

Correctness conditions $(\overline{s}, A, \overline{t}) \in \overline{\delta}$ implies:

- enabledness: action A is enabled at $\overline{s}$

  $$\overline{s} \Rightarrow \exists \mathbf{x}' : A$$

- effect: represent all possible A-successors

  $$\overline{s} \wedge A \Rightarrow \bigvee_{(\overline{s}, A, \overline{t}) \in \overline{\delta}} \overline{t}'$$

Model checking under fairness assumptions

Instrument abstract transition system $\overline{\mathcal{A}}$

add Boolean variables en_A and $taken_A$ for every action $A \in Act$:

- enabledness en_A true at states that have outgoing edge $(\overline{s}, A, \overline{t}) \in \overline{\delta}$
- execution $taken_A$ true when previous transition may have been caused by A

Weaken property to prove

Deduce $\mathcal{K} \models \varphi$ from

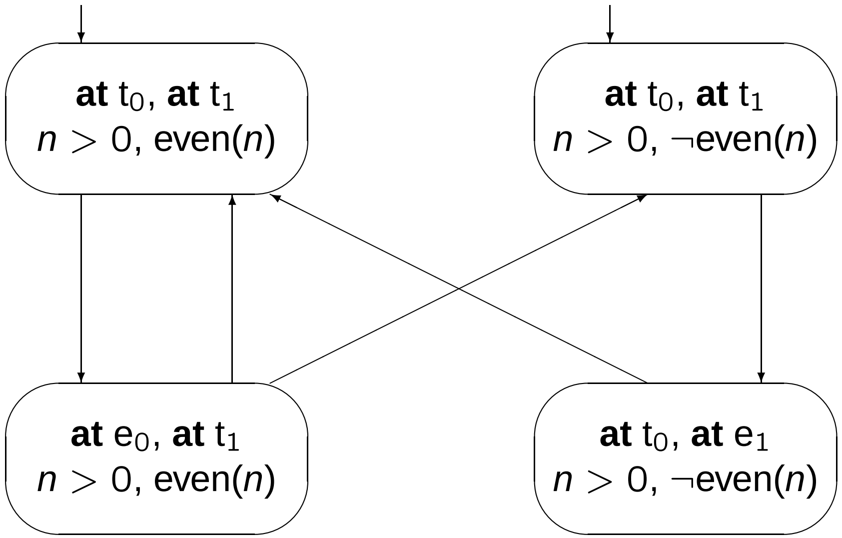
$$\overline{\mathcal{A}} \models \bigwedge_{A \in Act} \left\{ \begin{array}{c} WF(A) \\ SF(A) \end{array} \right\} \Rightarrow \varphi$$

for actions $A \in Act$ with weak (resp., strong) fairness assumption where

$$\begin{aligned} WF(A) &=_{\text{def}} \mathbf{F}\,\mathbf{G}\, en_A \Rightarrow \mathbf{G}\,\mathbf{F}\, taken_A \\ SF(A) &=_{\text{def}} \mathbf{G}\,\mathbf{F}\, en_A \Rightarrow \mathbf{G}\,\mathbf{F}\, taken_A \end{aligned}$$

Representing well-founded orderings

Reconsider “dining mathematicians”



int $n > 0$
loop
t_0 : **await** even(n);
e_0 : $n := n$ div 2
endloop

$\|$ . . .

No computation of “concrete” system cycles between left-hand nodes

n stays positive and even . . .

. . . but is infinitely often divided by 2

Note: every finite-state abstraction must contain similar cycle!

Ordering annotations

Represent descent w.r.t. well-founded ordering in $\overline{\mathcal{A}}$

– let t be (concrete-level) term and $\prec$ be well-founded ordering on domain of t

– label edge $(\overline{m}, A, \overline{n}) \in \overline{\delta}$ by $(t, \prec)$ (resp., $(t, \preceq)$) if

$$\overline{m} \wedge A \wedge \overline{n}' \quad \Rightarrow \quad \left\{ \begin{array}{c} t' \prec t \\ t' \preceq t \end{array} \right\}$$

Use edge annotations in model checking

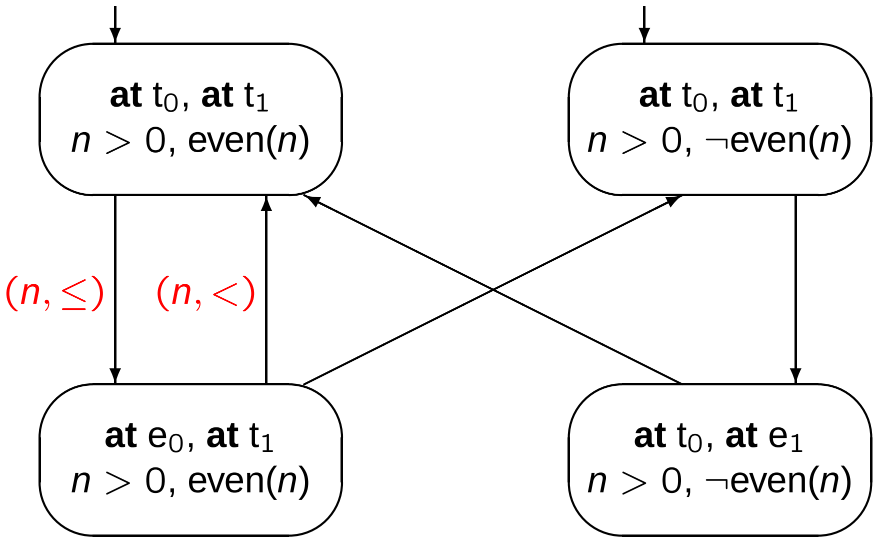
– exclude computations of $\overline{\mathcal{A}}$ that correspond to infinite descent of t in $\mathcal{K}$:

Deduce $\mathcal{K} \models \varphi$ from $\overline{\mathcal{A}} \models (\mathbf{G}\,\mathbf{F}\,“t' \prec t” \Rightarrow \mathbf{G}\,\mathbf{F}\,\neg“t' \preceq t”) \quad \Rightarrow \quad \varphi$

– $“t' \prec t”$ represented by auxiliary Boolean variables

Dining mathematicians completed

Diagram annotated with ordering information



$\mathbf{G}(n > 0)$

$\mathbf{G}\,\neg(\textbf{at } e_0 \wedge \textbf{at } e_1)$

$\mathbf{G}\,\mathbf{F}\,\textbf{at } e_0$

$\mathbf{G}\,\mathbf{F}\,\textbf{at } e_1$ can ~~not~~ be verified
also

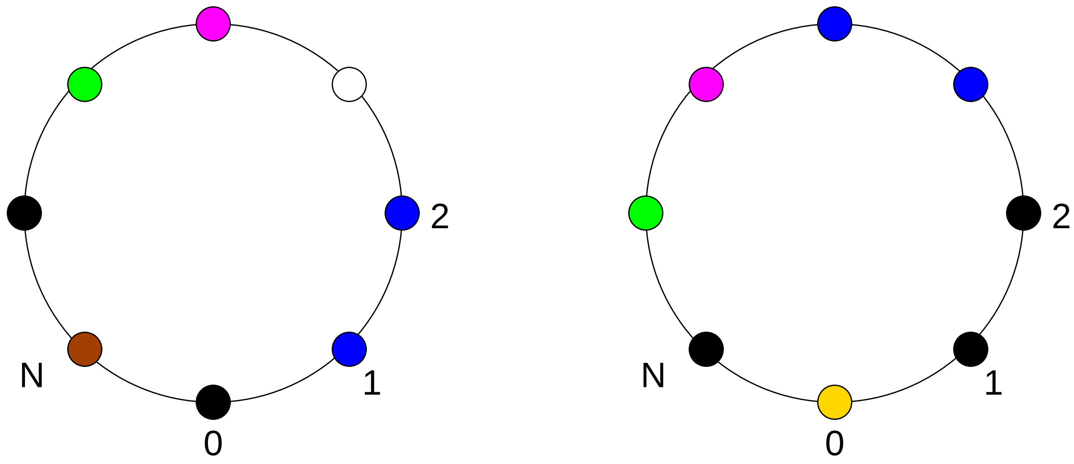
Justification

– $\mathrm{at}\ t_0 \wedge \mathrm{even}(n) \wedge \mathit{Next} \Rightarrow n' = n$

– $\mathrm{at}\ e_0 \wedge \mathrm{even}(n) \wedge n > 0 \wedge \mathit{Next} \Rightarrow n' = n \text{ div } 2$

Case study: self-stabilizing protocol

Dijkstra's algorithm for self-stabilization



var v : **array** $[0 \mathinner{..} N]$ **of** $[0 \mathinner{..} M]$

$$
\begin{array}{lll}
 & v[0] = v[N] & \longrightarrow \quad v[0] := (v[0]+1) \bmod (M+1) \\
\square_{i<N} & v[i+1] \neq v[i] & \longrightarrow \quad v[i+1] := v[i]
\end{array}
$$

Stable configurations: precisely one process can move

Protocol formalized in Isabelle

```
types
  proc     =   nat
  vals     =   nat
  state    =   proc => vals
consts
  N, M           :: nat
rules
  N_pos          "0 < N"
  M_atleast_N    "N ≤ M"
constdefs
  Proc      :: proc set                                    valid process IDs
            Proc ≡ {i . i ≤ N}
  Vals      :: vals set                                    valid register values
            Vals ≡ {i . i ≤ M}
  act       :: [proc, state, state] => bool                action definition
            act i v w ≡
              (case i of
                 0     => (v 0 = v N) ∧ (w 0 = (Suc (v 0)) mod (Suc M))
               | Suc j => (v j ≠ v (Suc j)) ∧ (w (Suc j) = v j))
              ∧ (∀ k. k ≠ i ⇒ (w k = v k))
  enab      :: [proc, state] => bool                       enabledness predicate
            enab i v ≡ ∃ w. act i v w
```

Idea of correctness

1. Once the ring has stabilized, it will remain stable.

2. Assume process 0's value $v[0]$ is different from all other register values.

   – Process 0 cannot move until $v[0]$ appears in register N.
   – $v[0]$ will eventually spread along the ring.
   – When $v[0]$ has reached register N, the configuration is stable.

3. Some value $k \leq M$ does not occur in any register $v[i+1]$.

   – Observe $N \leq M$ and use pigeonhole principle.
   – This value cannot be introduced into the ring except by an action of process 0.
   – Because every move disables that process, process 0 must make infinitely many moves.
   – Therefore its register will eventually contain k, and we have reached case 2.

Concepts for correctness formalized

prefix: initial ring segment with same register values

```
prefix   :: state => proc set
prefix v ≡ { i. i∈Proc ∧ (∀ j. j ≤ i ⇒ v j = v 0) }
```

othVals: values of registers outside the prefix

```
othVals   :: state => vals set
othVals v ≡ v ‘‘ (Proc \ (prefix v))
```

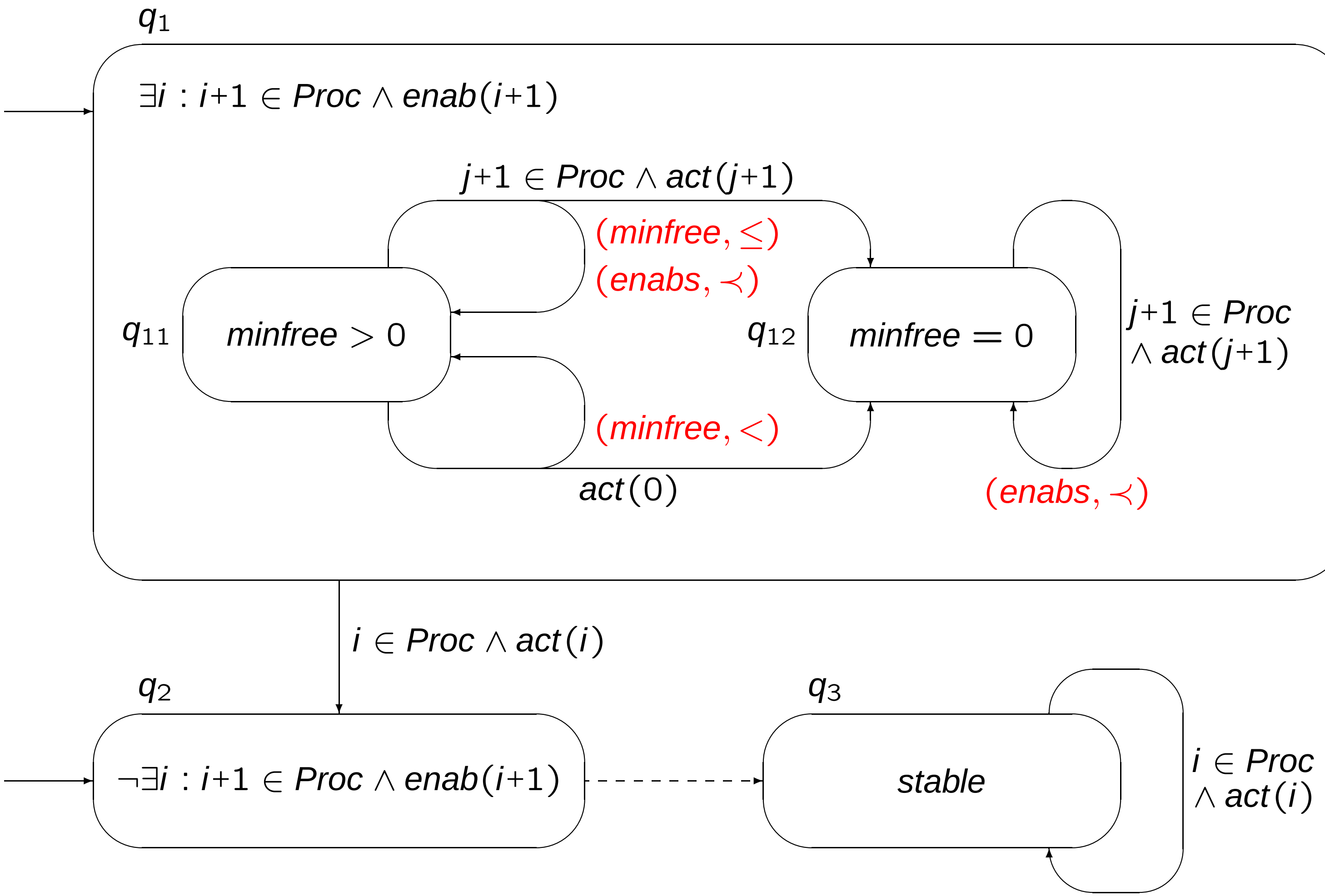
minfree: distance to least value not contained in othVals

```
minfree  :: state => vals
minfree v ≡ LEAST n. n ∈ Vals
                   ∧ (v 0 + n) mod (Suc M) ∉ othVals v
```

enabs: bit vector indicating enabledness of nonzero processes

ordered lexicographically

Predicate diagram for Dijkstra's protocol



$\mathbf{F}\,\mathbf{G}\,stable$ verifiable by model checking

Summary

Semi-automatic construction of abstraction followed by model checking

Combination of model checking, theorem proving, and abstract interpretation

Challenge: integrate tools (SAL project at SRI, Stanford, Berkeley, Grenoble)

Identify useful abstractions that can be generated automatically:

- parameterized systems (Manna, Sipma ’99; Baukus, Lakhnech, Stahl ’00)
- see also part of tutorial on infinite state spaces

Infinite State Spaces

Sources of infinity

Symbolic search: forward and backward

Accelerations and widenings

Sources of infinity

Data manipulation: unbounded counters, integer variables, lists . . .

Control structures: procedures $\longrightarrow$ stack, process creation $\longrightarrow$ bag

Asynchronous communication: unbounded FIFO queues

Parameters: number of processes, of input gates, of buffers, . . .

Real-time: discrete or dense domains

A bit of history

- Late 80s, early 90s: First theoretical papers

  Decidability/Undecidability results for Place/Transition Petri nets

  Efficient model-checking algorithms for context-free processes

  Region construction for timed automata

- 90s: Research program

  1. Decidability analysis
  2. Design of algorithms or semi-algorithms
  3. Design of implementations
  4. Tools
  5. Applications

- Late 90s, 00s: General techniques emerge

  Automata-theoretic approach to model-checking

  Symbolic reachability

  Accelerations and widenings

Parametrized protocols

Defined for *n* processes.

Correctness: the desired properties hold for every *n*

Processes modelled as communicating finite automata

For each value of *n* the system has a finite state space (only one source of infinity)

Turing powerful, and so further restrictions sensible:

Broadcast Protocols

Broadcast protocols

Introduced by Emerson and Namjoshi in LICS ’98

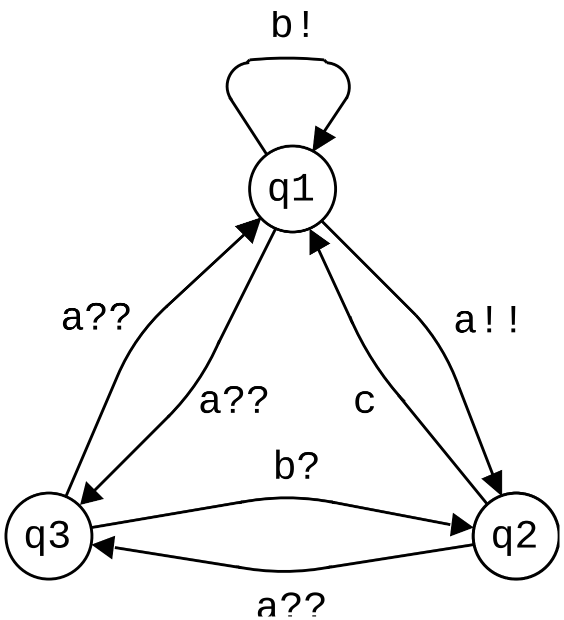
All processes execute the same algorithm, i.e., all finite automata are identical

Processes are undistinguishable (no IDs)

Communication mechanisms:

> Rendezvous: two processes exchange a message and move to new states
>
> Broadcasts: a process sends a message to all others
> all processes move to new states

Syntax



a!! : broadcast a message along (channel) *a*
a?? : receive a broadcasted message along *a*
b! : send a message to one process along *b*
b? : receive a message from one process along *b*
c : change state without communicating with anybody

Semantics

The global state of a broadcast protocol is completely determined by the number of processes in each state.

Configuration: mapping $: S \to \mathbb{N}$, seen as element of $\mathbb{N}^n$, where $n = |S|$

Semantics for each n: finite transition system

- configurations as nodes
- channel names as transition labels

In our example:

$$(3,1,2) \xrightarrow{c} (4,0,2) \quad \text{(silent move)}$$

$$(3,1,2) \xrightarrow{b} (3,2,1) \quad \text{(rendezvous)}$$

$$(3,1,2) \xrightarrow{a} (2,1,3) \quad \text{(broadcast)}$$

Semantics (continued)

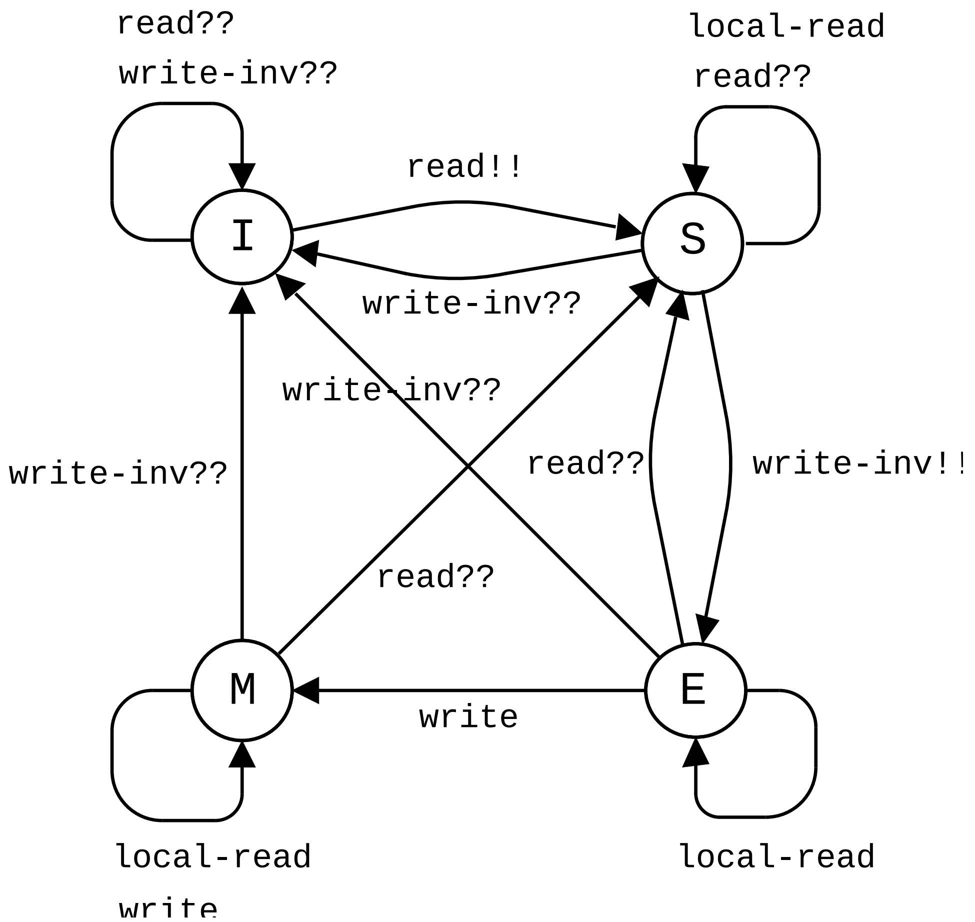
Parametrized configuration: partial mapping $p : Q \longrightarrow \mathbb{N}$

- Intuition: “configuration with holes”
- Formally: set of configurations (total mappings matching p)

(Infinite) transition system of the broadcast protocol:

- Fix an initial parametrized configuration p_0.
- Take the union of all finite transition systems $\mathcal{K}_c$ for each configuration $c \in p_0$.

A MESI-protocol



The automata-theoretic approach

System $S \Longrightarrow$ Kripke structure $\mathcal{K} \Longrightarrow$ Languages $\mathcal{L}(\mathcal{K})$, $\mathcal{L}_\omega(\mathcal{K})$ of finite and infinite computations

If systems closed under product with automata then $\mathcal{B}_{\neg\phi} \times \mathcal{K} \Longrightarrow S_{\neg\phi}$

Safety and liveness problems reducible to

- Reachability

  Given: system S, sets I and F of initial and final configurations of $\mathcal{K}$
  To decide: if F can be reached from I, i.e., if there exist $i \in I$ and $f \in F$ such that $i \to f$

- Repeated reachability

  Given: System S, sets I and F of initial and final configurations of S
  To decide: if F can be repeatedly reached from I, i.e. if there exist $i \in I$ and $f_1, f_2, \ldots \in F$ such that $i \to f_1 \to f_2 \cdots$

Shape of I and F depend on the class of atomic propositions

Model checking broadcast protocols

Repeated reachability is undecidable even for very simple sets I and F

It is undecidable if there is a value of n such that for this value
the broadcast protocol has an infinite computation

Reachability is decidable for upward-closed sets I and F

U is an upward-closed set of configurations if

$$c \in U \text{ and } c' \geq c \text{ implies } c' \in U$$

where $\geq$ is the pointwise order on $\mathbb{N}^n$.

Safety property: upward-closed set D of dangerous configurations

Example: in the MESI protocol the states M and S should be mutually exclusive

$$D = \{(m, e, s, i) \mid m \geq 1 \wedge s \geq 1\}$$

Symbolic search: forward and backward

Let C denote a (possibly infinite) set of configurations

Forward search

$post(C)$ = immediate successors of C

Initialize $C := I$

Iterate $C := C \cup post(C)$ until

$C \cap F \neq \emptyset$; return "reachable", or

a fixpoint is reached; return "non-reachable"

Backward search

$pre(C)$ = immediate predecessors of C

Initialize $C := F$

Iterate $C := C \cup pre(C)$ until

$C \cap I \neq \emptyset$; return "reachable", or

a fixpoint is reached; return "non-reachable"

Problem: when are the procedures effective?

Forward search effective if . . .

. . . there is a family $\mathcal{C}$ of sets such that

1. each $C \in \mathcal{C}$ has a symbolic finite representation;
2. $I \in \mathcal{C}$;
3. if $C \in \mathcal{C}$, then $C \cup post(C) \in \mathcal{C}$;
4. emptyness of $C \cap F$ is decidable;
5. $C_1 = C_2$ is decidable (to check if fixpoint has been reached); and
6. any chain $C_1 \subseteq C_2 \subseteq C_3 \ldots$ reaches a fixpoint after finitely many steps

(1)—(5) guarantee partial correctness, (6) guarantees termination

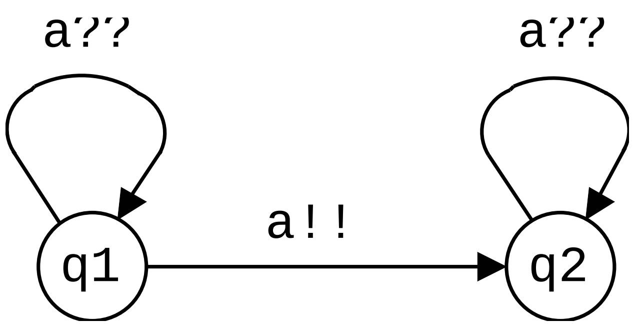
For backward search substitute $post(C)$ by $pre(C)$ and exchange I and F

Important difference: backward search starts from F instead from I;
I and F may have different properties!

Forward search in broadcast protocols

$\mathcal{C}$ must contain all parametrized configurations.

Satisfies (1)—(5) but not (6). Termination fails in very simple cases.



$$(\sqcup, 0) \xrightarrow{a} (\sqcup, 1) \xrightarrow{a} (\sqcup, 2) \xrightarrow{a} \dots$$

Backward search in broadcast protocols

[Abdulla, Cerāns, Jonsson, Tsay '96], [E., Finkel, Mayr '99]

The family of all upward-closed sets satisfies (1)—(6)

1. An upward-closed set can be represented by its set of minimal elements w.r.t. the pointwise order $\leq$ (Dickson's Lemma)

3. If U is upward-closed then so is $U \cup pre(U)$.

$$\begin{array}{ccc} c & \xrightarrow{a} & u \in U \\ \leq & & \leq \\ c' & \xrightarrow{a} & u' \in U \end{array}$$

6. Any chain $U_1 \subseteq U_2 \subseteq U_3 \ldots$ of upwards closed sets reaches a fixpoint after finitely many steps (Dickson's lemma + some reasoning)

Application to the MESI-protocol

Are the states M and S mutually exclusive?

Check if the upward-closed set with minimal element

$$m = 1, e = 0, s = 1, i = 0$$

can be reached from the initial p-configuration

$$m = 0, e = 0, s = 0, i = \sqcup\,.$$

Proceed as follows:

$$\begin{aligned}
U\colon &\quad m \geq 1 \land s \geq 1 \\
U \cup pre(U)\colon &\quad (m \geq 1 \land s \geq 1) \lor \\
&\quad (m = 0 \land e = 1 \land s \geq 1) \\
U \cup pre(U) \cup pre^2(U)\colon &\quad U \cup pre(U)
\end{aligned}$$

Other models

FIFO-automata with lossy channels

[Abdulla and Jonsson ’93], [Abdulla, Bouajjani, Jonsson ’98]

Configuration: pair $(q, \mathbf{w})$, where q state and $\mathbf{w}$ vector of words representing the queue contents

Class $\mathcal{C}$: upward-closed sets with the subsequence order

Backward search satisfies (1)—(6)

Timed automata

[Alur and Dill ’94]

Configuration: pair $(q, \mathbf{x})$, where q state and $\mathbf{x}$ vector of real numbers

Class $\mathcal{C}$: regions

Forward search satisfies (1)—(6)

Accelerations and widenings: setup

$post[\sigma](C)$ = set of configurations reached from C by the sequence σ

Compute a symbolic reachability graph with elements of $\mathcal{C}$ as nodes:

Add I as first node

For each node C and each label a, add an edge $C \xrightarrow{a} post[a](C)$

Accelerations

Replace $C \xrightarrow{\sigma} post[\sigma](C)$ by $C \xrightarrow{\sigma} X$, where X satisfies

(1) $post[\sigma](C) \subseteq X$, and

(2) X contains only reachable configurations

Condition (1) guarantees the acceleration

Condition (2) guarantees that only reachable configurations are computed

Acceleration through loops

A loop is a sequence $C \xrightarrow{\sigma} post[\sigma](C)$ such that

$$C \xrightarrow{\sigma} post[\sigma](C) \xrightarrow{\sigma} post[\sigma^2](C) \xrightarrow{\sigma} post[\sigma^3](C) \cdots$$

Syntactic loops (e.g. $s \xrightarrow{a!} s$ in FIFO-systems)

Semantic loops defined through simulations:

$$\begin{array}{lcl} C_1 & \text{is simulated by} & C_2 \\ \downarrow a & & \downarrow a \\ C_1' & \text{is simulated by} & C_2' \end{array}$$

If $post[\sigma](C)$ simulates C, then $C \xrightarrow{\sigma} post[\sigma](C)$ is a loop

Example: $M \xrightarrow{\sigma} M \geq M$ in Petri nets

Acceleration: given a loop $C \xrightarrow{\sigma} post[\sigma](C)$, replace $post[\sigma](C)$ by

$$X = post[\sigma^*](C) = C \cup post[\sigma](C) \cup post[\sigma^2](C) \cup \ldots$$

Problem: find a class of loops such that $post[\sigma^*](C)$ belongs to $\mathcal{C}$

Accelerations in broadcast protocols

Class $\mathcal{C}$: parametrized configurations

Class of loops: given by the following simulation

$$\text{If } \sqcup > n \text{ for all } n \text{ then } \begin{array}{ccc} p_1 & \leq & p_2 \\ \downarrow a & & \downarrow a \\ p_1' & \leq & p_2' \end{array}$$

So if $C \leq post[\sigma](C)$ then $post[\sigma](C)$ simulates C

$post[\sigma^*](p)$ may not be a parametrized configuration

Other models I

Counter machines [Boigelot, Wolper 94]

Configuration: pair $(q, n_1, \ldots, n_k)$, where q state $n_1, \ldots, n_k$ integers

Class $\mathcal{C}$: Presburger sets

Class of loops: syntactic

Pushdown automata [Bouajjani, E., Maler '97]

Configuration: pair (q, w), where q state and w stack content

Class $\mathcal{C}$: regular sets

Class of loops: through semantic loops $(q, aw) \xrightarrow{\sigma} (q, aw'w)$

Acceleration guarantees termination for both forward and backward search!

Other models II

FIFO-automata with lossy channels [Abdulla, Bouajjani, Jonsson '98]

Configuration: pair $(q, \mathbf{w})$, where *s* state and $\mathbf{w}$ vector of words representing the contents of the queues

Class $\mathcal{C}$: regular sets represented by simple regular expressions

Class of loops: arbitrary

Other examples

FIFO-automata with perfect channels [Boigelot, Godefroid], [Bouajjani, Habermehl '98]

Arrays of parallel processes [Abdulla, Bouajjani, Jonsson, Nilsson '99]

Widenings

Accurate widenings

Replace $C \xrightarrow{\sigma} post[a](C)$ by $C \xrightarrow{\sigma} X$, where X satisfies

(1) $post[a](C) \subseteq X$, and

(2') X contains only reachable final configurations

Notice that X may contain unreachable non-final configurations!

Inaccurate widenings

Replace $C \xrightarrow{\sigma} post[a](C)$ by $C \xrightarrow{\sigma} X$, where X satisfies

(1) $post[a](C) \subseteq X$

If no configuration of the graph belongs to F, then no reachable configuration belongs to F

If some configuration of the graph belongs to F, no information is gained

Accurate widenings in broadcast protocols

Fact: $post[\sigma](p) = T_\sigma(p)$ for a linear transformation $T_\sigma(p) = M_\sigma \cdot x + b_\sigma$

It follows: $post[\sigma^*](p) = \bigcup_{n \geq 0} T_\sigma^n(p)$

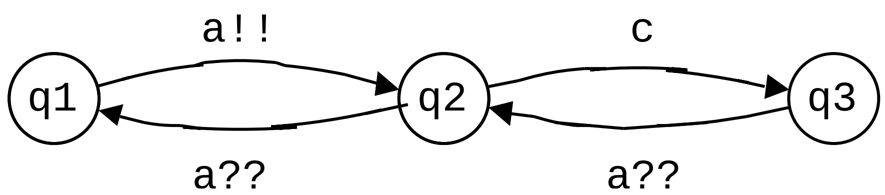
Accurate widening: widen $post[\sigma^*](p)$ to $lub\{T_\sigma^n(p) \mid n \geq 0\}$

Theorem: if the set F is upward closed, this widening is accurate

Does widening lead to termination?

For arbitrary broadcast protocols: NO [E., Finkel, Mayr '99]

Example in which the acceleration doesn't have any effect:



$$p_0 = (\sqcup, 0, 0)$$

For rendezvous communication only: YES
[Karp and Miller '69], [German and Sistla '92]

Implementing backwards reachability

Linear constraints as finite representation of sets of configurations.

The variable x_i represents the number of processes in state q_i

Set of configurations $\rightarrow$ set of constraints over $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$

(interpreted disjunctively)

Immediate predecessors computed symbolically

Union and intersection $\longrightarrow$ disjunction and conjunction

Containment test $\longrightarrow$ entailment

Label $a \longrightarrow$ linear transformation with guard.

In our example

- Guard $\mathbf{G_a}$: $x_1 \geq 1$
- Linear transformation $\mathbf{M_a x} + \mathbf{b_a}$:

$$M_a = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix} \quad b_a = \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}$$

Symbolic computation of pre must satisfy

$$pre(\Phi) \;\equiv\; \bigvee_{a \in \Sigma, \phi \in \Phi} \mathbf{G_a} \wedge \phi[\mathbf{x} \,/\, \mathbf{M_a x} + \mathbf{b_a}]$$

Which class of constraints?

Able to express all upward-closed sets

Efficient computation of pre

Efficient entailment test

- Entailment test co-NP-complete for arbitrary constraints

Natural candidates

L-constraints

Conjunction of inequations of shape $x_1 + \ldots + x_n \geq c$

Closed under broadcast transformations.

Entailment co-Np-complete even for single constraints

WA-constraints

Conjunction of inequations of shape $x_i \geq c$

Entailment is polynomial (quadratic)

Not closed under broadcast transformations.
L-constraints equivalent to sets of WA-constraints, but with exponential blow-up:

$$x_{i_1} + \ldots + x_{i_m} \geq c \quad \equiv \bigvee_{c_1 + \ldots + c_m = c} x_{i_1} \geq c_1 \wedge x_{i_2} \geq c_2 \wedge \ldots \wedge x_{i_m} \geq c_m$$

Using WA-constraints

[Delzanno and Raskin '00]

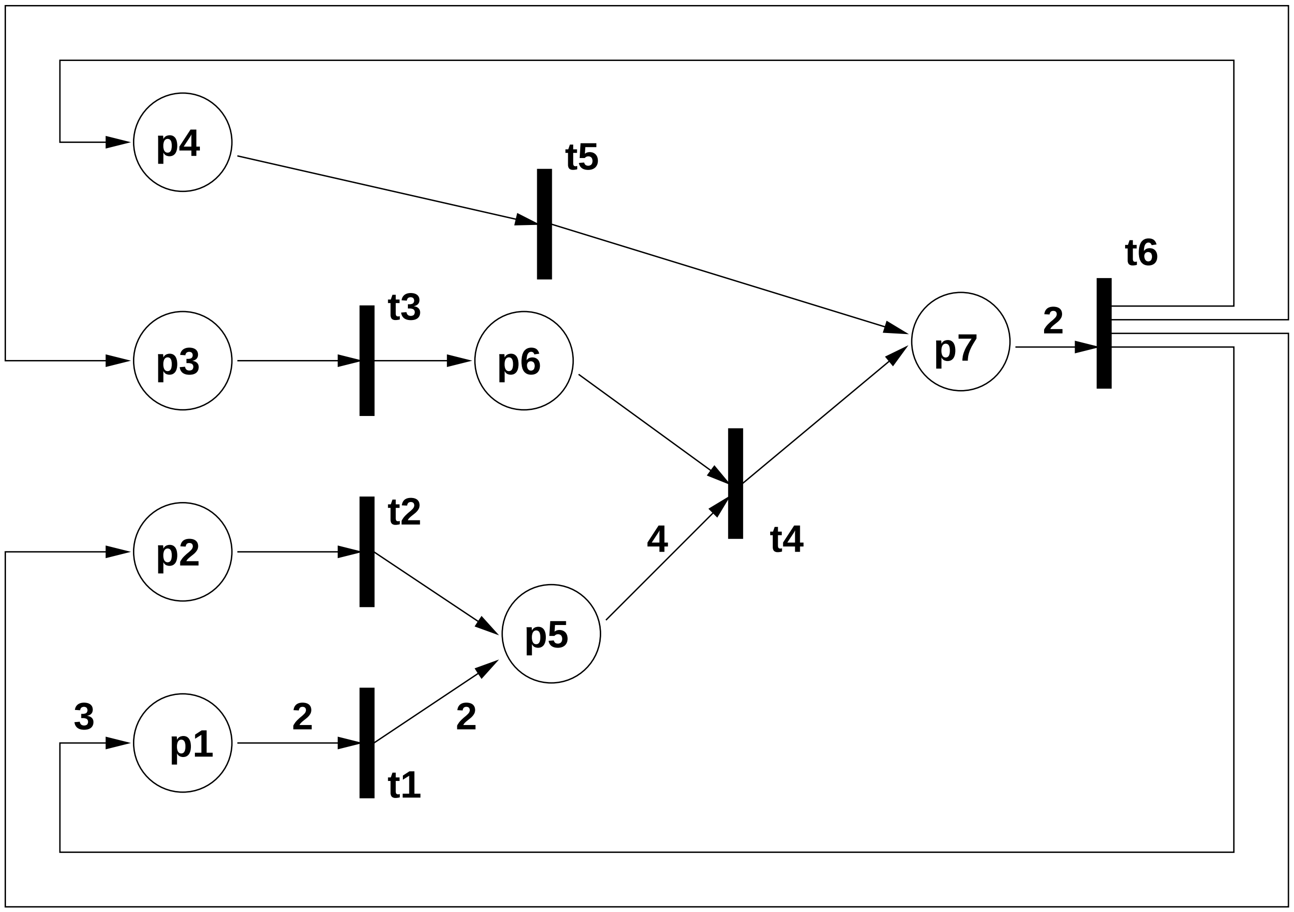
Represent the constraint $x_1 \geq c_1 \wedge \ldots \wedge x_n \geq c_n$ by $(c_1, \ldots c_n)$

Use sharing trees to represent sets of constraints

A sharing tree is an acyclic graph with one root and one terminal node such that

- all nodes of layer i have successors in the layer $i + 1$
- a node cannot have two successors with the same label
- two nodes with the same label in the same layer do not have the same set of successors

A small Petri net experiment [Teruel '98]



Deadlock-free states are the predecessors of an upward-closed set

Deadlock-free initial markings:

$P_1 \geq 10, P_2 \geq 1, P_3 \geq 2$ $\quad$ $P_1 \geq 8, P_3 \geq 3$ $\quad$ $P_1 \geq 12, P_3 \geq 2$

$P_1 \geq 6, P_2 \geq 5, P_3 \geq 2$ $\quad$ $P_1 \geq 8, P_3 \geq 1, P_4 \geq 1$ $\quad$ $P_1 \geq 6, P_4 \geq 2$

$P_1 \geq 6, P_2 \geq 1, P_3 \geq 1, P_4 \geq 1$

Computation time (Sun Ultra Sparc):

Sharing trees	HyTech	Presburger
39s	$>$ 24h	19h50m

Using L-constraints

[Delzanno, E., Podelski '99], [Delzanno '00]

First simplification: entaiment need only be computed for single constraints

Replace

$$\textbf{until } \mathit{Entail}(\Phi, \mathit{old\_}\Phi)$$

by the stronger condition

$$\textbf{until forall } \phi \in \Phi \textbf{ exists } \psi \in \mathit{old\_}\Phi : \quad \mathit{Entail}(\phi, \psi)$$

Possibly slower, but still guaranteed termination

But entailment for L-constraints co-Np-complete even for single constraints!

Second simplification: interpret entailment over the reals

Again, stronger **until**-condition which does not spoil termination

Case studies (by G. Delzanno)

Broadcast protocols must be extended with more complicated guards.

Termination guarantee gets lost

Berkeley RISC, Illinois, Xerox PARC Dragon, DEC Firefly

- At most 7 iterations and below 100 seconds (SPARC5, Pentium 133)

Futurebus +

- 8 steps and 200 seconds (Pentium 133)

Conclusions

Decidability analysis very advanced

Many algorithms useful in practice

In the next years: improve implementations, integrate in tools.

Challenge: several sources of infinity.