

Non-Deterministic Automata for Efficient Verification of Infinite-State Systems

Pierre Ganty Laurent Van Begin
Département d'informatique – Université Libre de Bruxelles
{pganty,lvbegin}@ulb.ac.be

28 march 2004

What systems do we analyze?

- Parametrized systems:

$$\overbrace{C_1}^{k_1} \parallel \dots \parallel \overbrace{C_n}^{k_n}$$

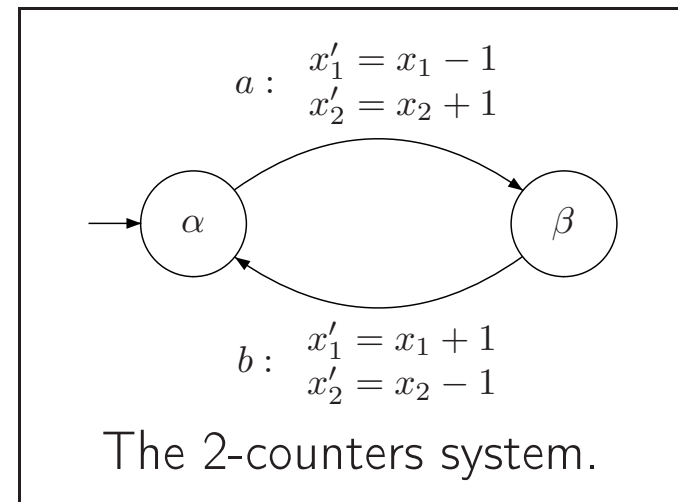
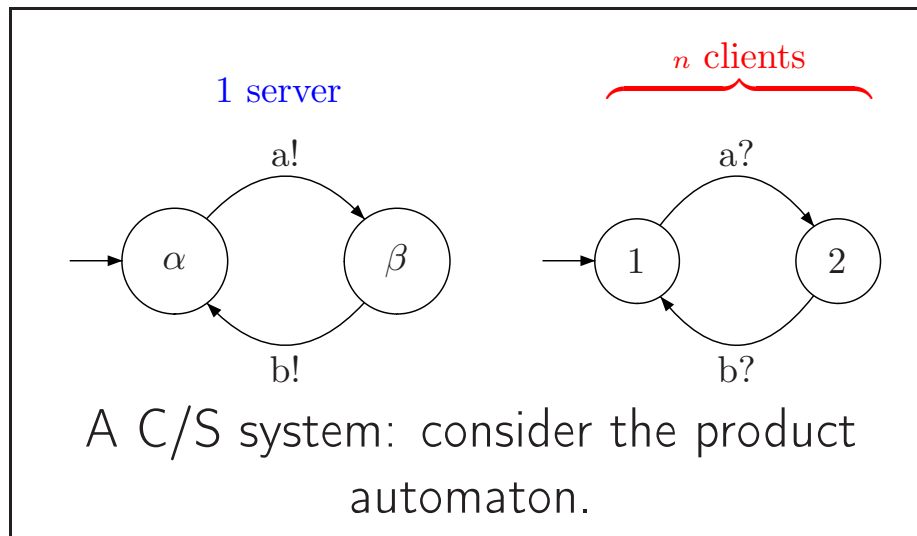
where n is a fixed value and k_i denotes the number of copies of component C_i (for $1 \leq i \leq n$). k_i can be either a fixed value or a **parameter**.

- Real-life applications:
 - Multi-Threaded programs featuring the dynamic creation of instances of threads. JAVA, for instance, supports such features.
 - Cache coherence protocols.
 - Broadcast protocols.
- Parametrized systems can be viewed as infinite-state systems.

Counter Systems to model Multi-Threaded programs

- A program consists in a **finite set of type of threads**.
- Each type of thread is modeled by a **finite state automaton**.
- Each instance (viz. thread) behaves **independently** from its identity.
- **Counter systems count how many instances lie in each state.**
- With the **dynamic creation**, some threads might be instantiated an **arbitrary number** of times.
- So we assume an unbounded number of instances and thus **unbounded** counters.

Counter Systems: a simple 2-counters example



Reachability and coverability analysis

Given a k -counter system \mathcal{M} , a set of initial states I and a set of bad states U .

Reachability Starting from I and using the transitions of \mathcal{M} , is it possible to reach any state of U ?

Undecidable in general.

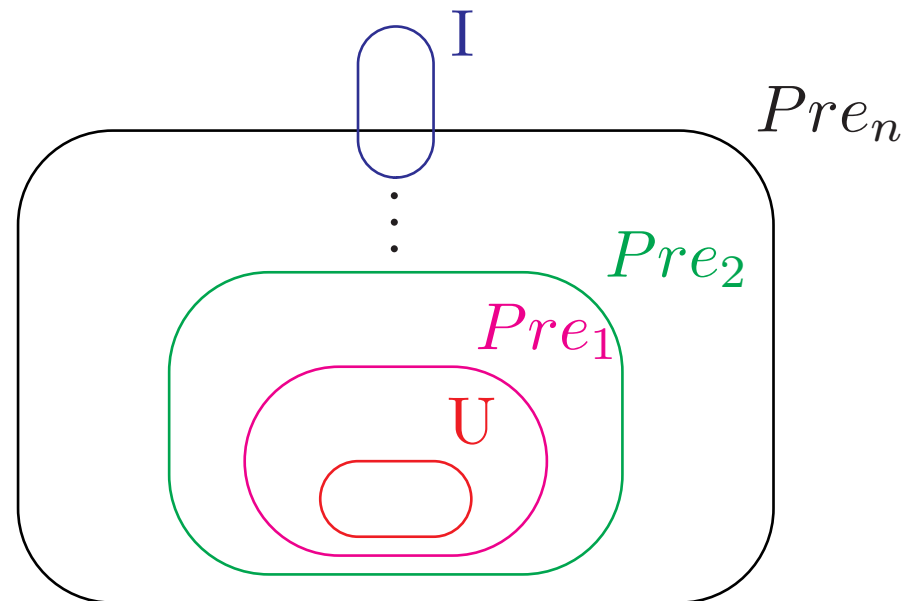
Coverability Starting from I and using the transitions of \mathcal{M} , is it possible to reach a state $\mathbf{u}' \succeq \mathbf{u}$ with $\mathbf{u} \in U$?

Given $\mathbf{u}' = \langle u'_1, \dots, u'_k \rangle$ and $\mathbf{u} = \langle u_1, \dots, u_k \rangle$ $\mathbf{u}' \succeq \mathbf{u}$ iff $u'_i \geq u_i$ for $1 \leq i \leq k$.

Decidable for monotonic systems.

Verification of Counter System: A Backward Approach

Given a counter system \mathcal{M} , a set of initial states I and a set of bad states U :



Needs for Verification

- What the backward approach does require ?

To manipulate possibly **infinite sets** of states of a k -counter system.

- How to represent such sets ?

To fit in a computer we need a **finite symbolic representation**.

- What else ?

The representation must be **closed** to some operations (*e.g.* \cup , \cap)

Some useful problems must be **decidable** (*e.g.* emptiness, inclusion)

- Not enough ?

Moreover, we need **efficiency**.

A denotational representation of sets

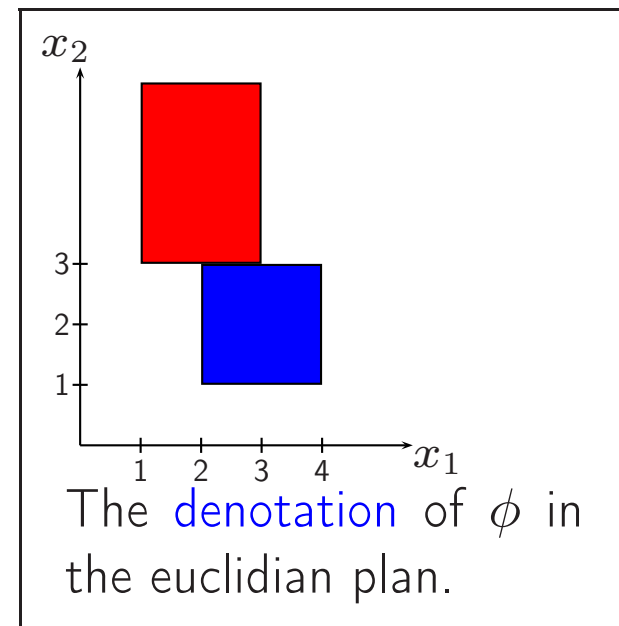
Let $V = \{x_1, \dots, x_k\}$ denote a **finite set of variables**. The following grammar defines the set of \mathcal{I}_k -formulas:

$$\Phi ::= x_i \in [a, b] \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \quad \text{where } a \in \mathbb{N}, b \in \mathbb{N} \cup \{+\infty\}$$

- \mathcal{I}_k -formulas **denotes sets** of states of a k -counter system.
- to this end, a **bijection** associates V to the k counters.
- the **denotation** of an atom: $\llbracket x_i \in [a, b] \rrbracket$ is **the set of natural** values within the interval $[a, b]$.
- the **emptiness** and **inclusion** test of denotations are **decidable** for the \mathcal{I}_k -formulas.

A very simple \mathcal{I}_2 -formula and its denotation

$$\phi \equiv \left(x_1 \in [1, 3] \wedge x_2 \in [3, +\infty) \right) \vee \left(x_1 \in [2, 4] \wedge x_2 \in [1, 3] \right)$$



The Backward Approach using \mathcal{I}_k -formulas

Input: Φ_I, Φ_U : \mathcal{I}_k -formula

Result: boolean

begin

$\Phi \leftarrow \Phi_U$

$\Psi \leftarrow \text{empty}$ (i.e. $\Psi = \bigwedge_{i:1}^k x_i \in [a, b]$ with $a > b$)

while $\llbracket \Phi \rrbracket \not\subseteq \llbracket \Psi \rrbracket$ **do**

$\Psi \leftarrow \Phi$

$\Phi \leftarrow \Psi \vee \text{pre}_{\mathcal{M}}(\Psi)$

od

if $\llbracket \Phi_I \wedge \Phi \rrbracket \neq \emptyset$ **then** return **true** **else** return **false**

end

To ease their manipulation, \mathcal{I}_k -formulas are in **disjunctive form**.

Not enough?

- As any other approach, we face the **symbolic state explosion problem**.
- In fact, during the backward approach the **size** of the manipulated \mathcal{I}_k -formulas (Φ and Ψ) **blows up**.
- To palliate the problem, we defined a data structure to **efficiently** represent huge formulas.

Towards an efficient representation of sets

We propose a symbolic representation for \mathcal{I}_k -formulas called **Interval Sharing Tree** (IST for short).

The **salient features** of IST are:

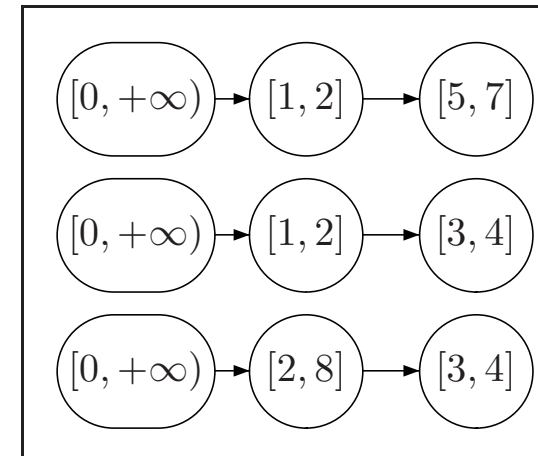
- **directed acyclic graphs**;
- underlying **non-determinism**;
- **symbolic manipulations**.

Interval Sharing Trees: an intuitive approach

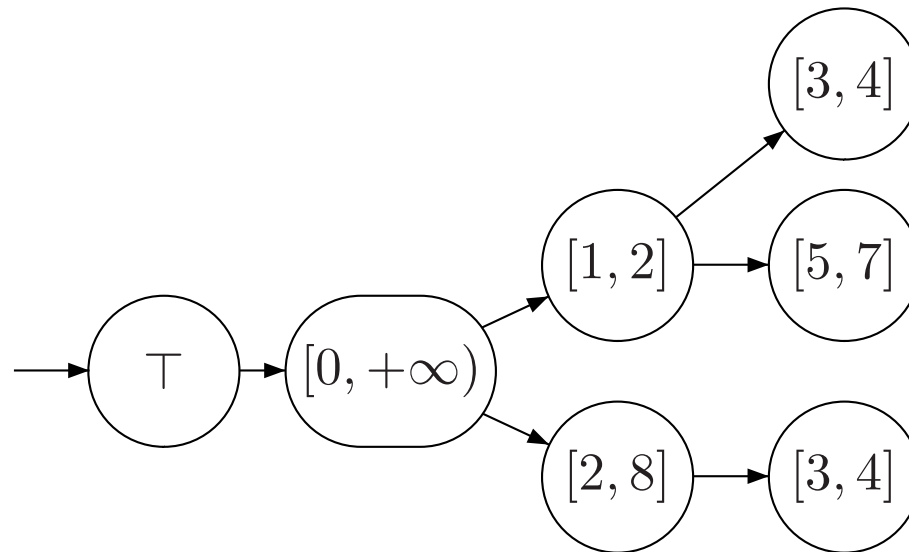
$$\phi \equiv \left(x_1 \in [0, +\infty) \wedge x_2 \in [1, 2] \wedge x_3 \in [5, 7] \right)$$

$$\vee \left(x_1 \in [0, +\infty) \wedge x_2 \in [1, 2] \wedge x_3 \in [3, 4] \right)$$

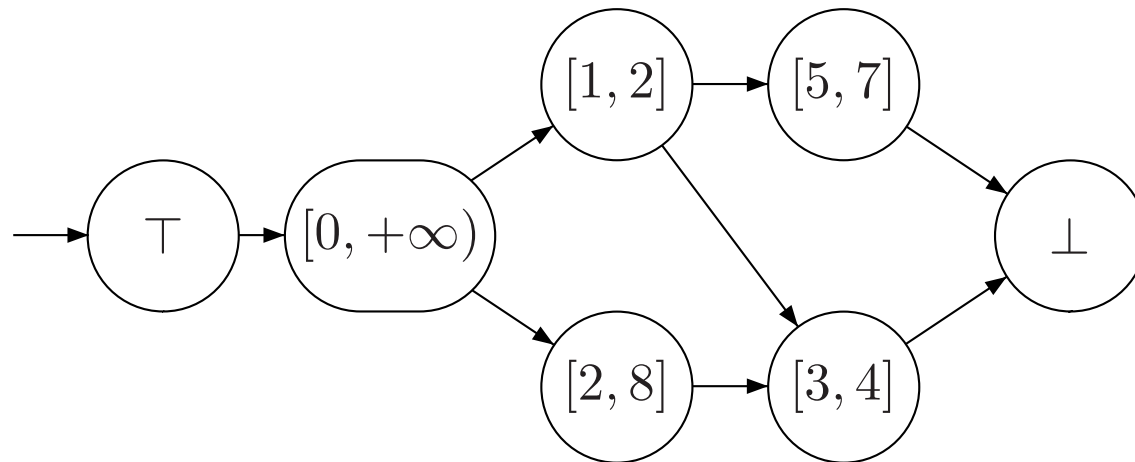
$$\vee \left(x_1 \in [0, +\infty) \wedge x_2 \in [2, 8] \wedge x_3 \in [3, 4] \right)$$



ISTs: Sharing of Prefixes



ISTs: Sharing of Suffixes



Operations on ISTs

Theorem 1 Given an IST S , its **size** (viz., $|S| \stackrel{def}{=} N_S + E_S$) can be **logarithmic** with respect to the number of paths it counts.

Theorem 2 **No polynomial** algorithm exists for a few **operations**. Some **tests** turn out to be **co-NP** complete problems.

But **all** operations/tests are defined **symbolically** on ISTs: **w/o enumerating paths**

Empirical results

	C	T	NI	ET (sec.)
P/C	44	37	25	3.7
P/C _{fixed}	44	38	1	< 0.1
I/D	32	28	29	0.5
dl	48	42	38	5
del ₁	50	52	54	1570
del ₂	50	52	18	1.9
queue	80	104	49	3300
trans	90	117	1	< 0.1
conf	30	35	33	1.2
pnca ₁	31	36	38	22.6
pnca ₂	31	36	17	0.6
r/w	22	24	118	532.6

Table 1: On a Intel© Xeon CPU 3.06 GHz with 4 Gb of memory

Conclusion & Future Works

Conclusion

- IST-based verification procedure;
- Efficient on Parametrized Abstraction of Multi-Threaded JAVA programs;
- Implementation under the GPL available (just send me a mail).
- Web page: <http://www.ulb.ac.be/di/ssd/lvbegin/IST/IST.html>

Future Works

- Widening Techniques;
- Integration of the library into a complete “Verification-Abstraction” algorithm.

Definitions et complexity of operations on STs (w.r.t. their size)

Test/Oper	Effect	Alg
Emptyness	$is_empty(S)$ iff $elem(S) = \emptyset$	const
Membership	$member(t, S)$ iff $t \in elem(S)$	linear
Inclusion	$contained(S, T)$ iff $elem(S) \subseteq elem(T)$	poly
Union	$elem(union(S, T)) = elem(S) \cup elem(T)$	poly
Intersection	$elem(intersection(S, T)) = elem(S) \cap elem(T)$	poly
Difference	$elem(minus(S, T)) = elem(S) \setminus elem(T)$	poly

Definitions et complexity of operations on ISTs (w.r.t. their size)

Test/Oper	Effect	Pb	Alg
Emptyness	$empty_{ist}(S)$ iff $fill(S) = \emptyset$	P	const
Satisfiability	$sat_{ist}(t, S)$ iff $fill(t) \subseteq fill(S)$	P	linear
Subsumption	$subsumed_{ist}(S, T)$ iff $fill(S) \subseteq fill(T)$	co-NP	exp
Substitution	$substitute_{ist}(S, c, i) = T$ t.q. $fill(T) = fill(S)[t_i + c/t_i]$	poly	exp
Disjonction	$union_{ist}(S, T) = U$ t.q. $fill(U) = fill(S) \cup fill(T)$	poly	poly
Conjonction	$intersection_{ist}(S, T) = I$ t.q. $fill(I) = fill(S) \cap fill(T)$	poly	exp