

Flat acceleration.

Sébastien Bardin Alain Finkel Jérôme Leroux
Philippe Schnoebelen

LSV - CNRS & ÉNS de Cachan

Journées Systèmes Infinis 2005

What is it about?

Symbolic model-checking

- extend model-checking to infinite systems,
- many techniques and tools

Problem of convergence

- theory: often undecidable
- practice: the iterative fixpoint computation is not sufficient.

Acceleration

- methods to enhance convergence, mainly by computing the transitive closure of some part of the system.
- promising case-studies
- BUT (too many!) different techniques and different tools.

What is it about?

Toward a unification of acceleration techniques

1. General framework for symbolic model-checking with acceleration, encompassing most existing techniques.
2. We identify three main levels for acceleration techniques (*loop*, *flat*, and *global*). For each one:
 - a symbolic procedure computing reachability sets
 - characterization of the class of systems on which it terminates.
3. Several algorithmic/heuristic improvements for flat acceleration.

What is it about?

Unification and clarification of previous works in the field.

- common theoretical background justifying existing tools (ALV, LASH, FAST, TREX)
- meaningful comparisons of techniques and tools.
- guidelines to improve or design from scratch symbolic model checkers with acceleration.

Context

- We do not check programs but mathematical models.
- Automata extended with a finite number of variables.
- Many applications.

- Communication protocols,
- Embedded systems,
- Program abstractions, ...

- VASS/Petri Nets,
- Timed Automata,
- Hybrid systems,
- (lossy) CFSMs, ...

Systems

Systems $S = (\Sigma, Q, T, D, \llbracket \cdot \rrbracket)$

- an infinite set of formulas Σ ,
- a finite labeled control structure
 - finite set Q of locations
 - finite set of transitions $T \subseteq Q \times \Sigma \times Q$.
- an interpretation of formulas $I = (\Sigma, D, \llbracket \cdot \rrbracket)$
 - a domain of interpretation D ,
 - an interpretation function $\llbracket \cdot \rrbracket : \Sigma \rightarrow 2^{D \times D}$

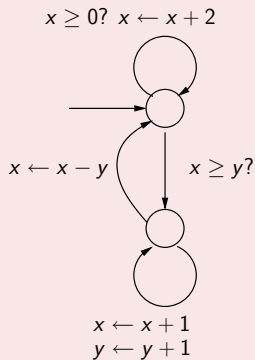
Systems

Ex: Counter Systems

1- Σ = linear assignments, linear inequalities on variables x, y .

2- $D = \mathbb{N}^2$

3- $[[\cdot]]$: " $x \geq 0? x \leftarrow x + 2$ " \rightarrow
 $\{(x, y, x', y') \mid x \geq 0 \wedge x' = x + 2 \wedge y' = y\}$



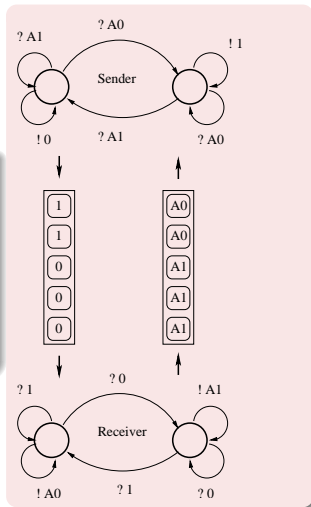
Systems

Ex: Channel Systems

1- $\Sigma = ?a$ or $!a$ with $a \in \{0, 1, A0, A1\}$.

2- $D = (0, 1)^* \times (A0, A1)^*$

3- $\llbracket \cdot \rrbracket$: “!A1” \rightarrow
 $\{((w1, w2), (w1, A1 \cdot w2))\}$



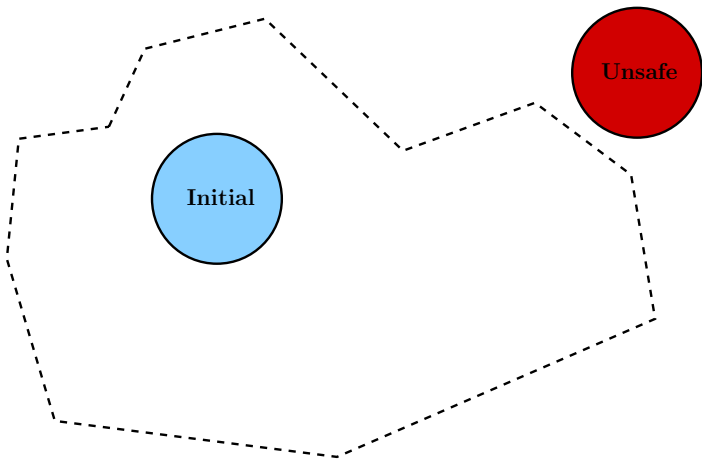
Systems - semantics

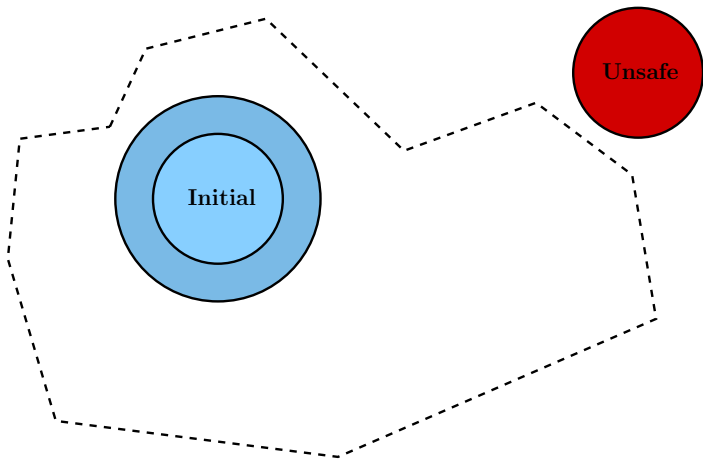
Transition system, reachability set

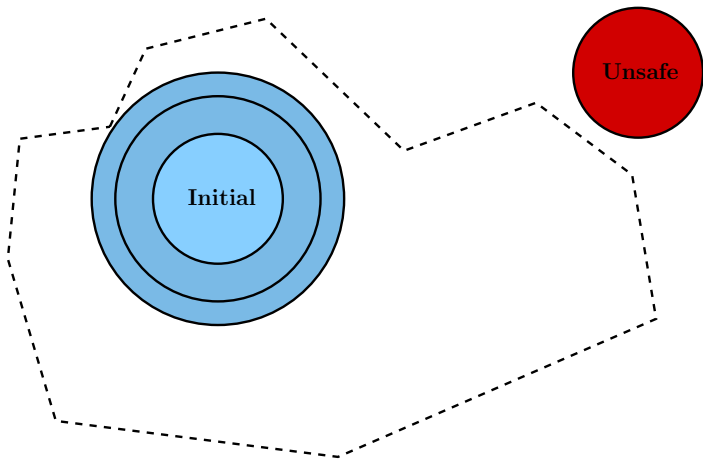
- configuration $c = (q, d) \in Q \times D$
- $(q, d) \xrightarrow{(q', l, q')} (q', d')$ iff $(d, d') \in \llbracket l \rrbracket$.
- $c' \in \text{post}^*(c)$ iff $c \xrightarrow{t_1} c_1 \xrightarrow{t_2} \dots \xrightarrow{t_n} c'$.
- $\text{post}^*(c_0)$ is the reachability set from c_0 .

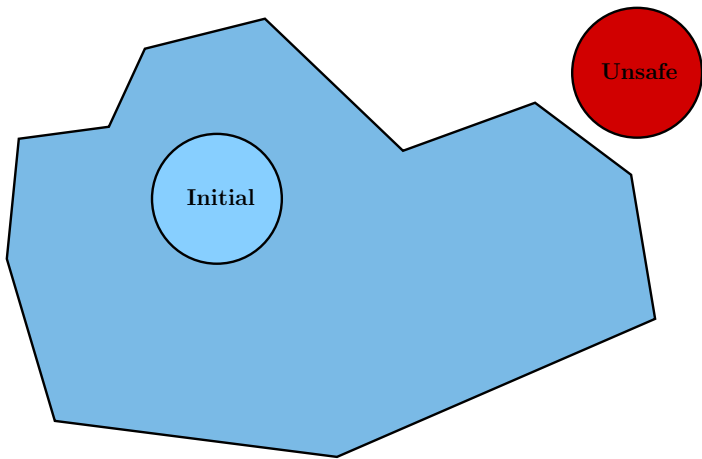
Safety properties = properties on reachable configurations.

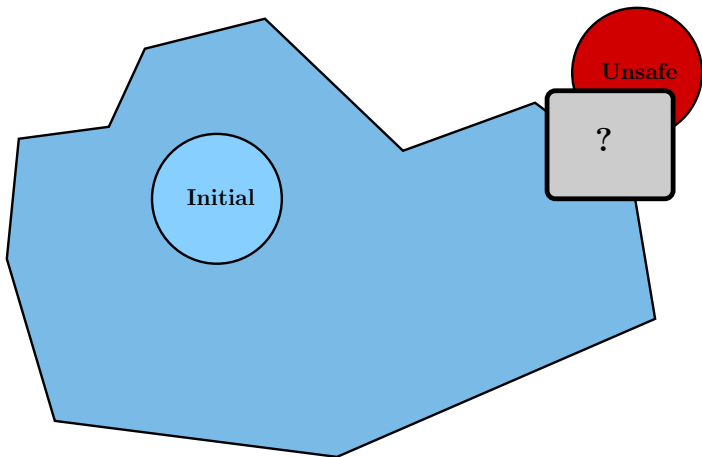
- interesting: mutual exclusion, deadlock freedom, ...
- can be checked easily from the reachability set.
- we focus on the computation of $\text{post}^*(c_0)$

Iterative computation of $\text{post}^*(c)$ 

Iterative computation of $\text{post}^*(c)$ 

Iterative computation of $\text{post}^*(c)$ 

Iterative computation of $\text{post}^*(c)$ 

Iterative computation of $\text{post}^*(c)$ 

Comments

Works well for systems with a finite number of configurations.

Problem in the infinite case

- the state space is infinite.
- enumerating concrete configurations cannot be sufficient.

Solution

- Manipulate symbolic configurations (regions) $x \in L$ encoding (infinite) sets of configurations.
- Symbolic representations must at least provide symbolic operations for post, \cup , \subseteq (denoted POST , \sqcup , \sqsubseteq)
- Depends heavily on the underlying interpretation $(\Sigma, D, \llbracket \cdot \rrbracket)$.

Symbolic iterative computation of $\text{post}^*(c)$ **procedure** REACH1(x_0)parameters: S input: initial region $x_0 \in L$ 1: $x \leftarrow x_0$ 2: **while** $\text{post}(x) \not\sqsubseteq x$ **do**3: $x \leftarrow \text{post}(x) \sqcup x$ 4: **end while**5: return x

Symbolic framework

Symbolic framework

A *symbolic framework* $SF = (\Sigma, D, \llbracket \cdot \rrbracket_1, L, \llbracket \cdot \rrbracket_2)$

- $I = (\Sigma, D, \llbracket \cdot \rrbracket_1)$ is an interpretation,
- L is a set of formulas called *regions*,
- $\llbracket \cdot \rrbracket_2 : L \rightarrow 2^D$ is a *region concretization*,

such that there exist a decidable relation \sqsubseteq and recursive functions \sqcup, POST satisfying

- 1 there exists an element $\perp \in L$ such that $\llbracket \perp \rrbracket_2 = \emptyset$.
- 2 $x_1 \sqsubseteq x_2$ iff $\llbracket x_1 \rrbracket_2 \subseteq \llbracket x_2 \rrbracket_2$.
- 3 $\llbracket x_1 \sqcup x_2 \rrbracket_2 = \llbracket x_1 \rrbracket_2 \cup \llbracket x_2 \rrbracket_2$.
- 4 $\forall a \in \Sigma, \llbracket \text{POST}(a, x) \rrbracket_2 = \llbracket a \rrbracket_1 (\llbracket x \rrbracket_2)$.

Symbolic framework - Examples

Ex: Counter Systems

1- Σ = linear assignments, linear inequalities on variables x, y .

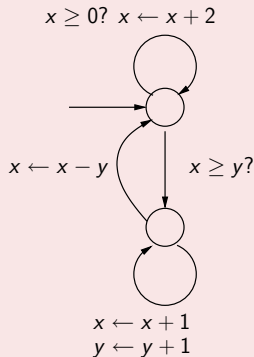
2- $D = \mathbb{N}^2$

3- $[\cdot]_1$: " $x \geq 0? x \leftarrow x + 2$ " \rightarrow
 $\{(x, y, x', y') \mid x \geq 0 \wedge x' = x + 2 \wedge y' = y\}$

4- L = Presburger formulas over free variables x, y

5- $[\cdot]_2$: the corresponding semi-linear set over \mathbb{N}^2 .

6-POST, \sqcup , \sqsubseteq : corresponding operations on Presburger formulas



Symbolic framework - Examples

Ex: Channel Systems

1- $\Sigma = ?a$ or $!a$ with $a \in \{0, 1, A0, A1\}$.

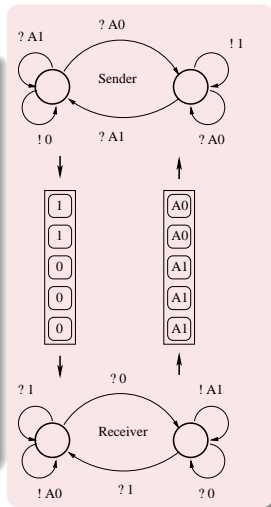
2- $D = (0, 1)^* \times (A0, A1)^*$

3- $[\cdot]$: $!A1 \rightarrow \{((w1, w2), (w1, A1 \cdot w2))\}$

4- $L = \text{regexp}(\{0, 1, A0, A1, \#\})$

5- $[\cdot]_2$: the corresponding language as channel content ($\#$ separates queues).

6- POST , \sqcup , \sqsubseteq : corresponding operations on regular expressions



Symbolic framework - Examples

- Counter systems: CST, NDD
- Timed automata: Difference Bound Matrices
- Hybrid systems: Convex polyhedra, RVAs, \approx PDBM
- CFSM: QDD, CQDD, SLRE
- lossy CFSM: SRE
- Token ring of arbitrary size: APC.
- Pushdown systems: regular expressions.

Limits of the iterative symbolic computation

- 1 Given a symbolic framework (I, L) and an initial region x_0 , then it may be the case that $\text{post}^*(\llbracket x_0 \rrbracket) \notin L$.
- 2 Even for systems for which $\text{post}^*(\llbracket x_0 \rrbracket) \in L$, REACH1 does not often converge (timed automata or pushdown).
- 3 Practical termination is limited to very specific systems.

Outline

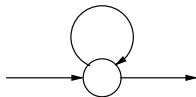
- 1 Motivations
- 2 **Beyond iterative computation: Acceleration**
- 3 Flat acceleration in depth
- 4 Framework of flat acceleration, applications
- 5 Conclusion

Notion of acceleration

Acceleration

Enhance convergence of the iterative procedure by computing directly the infinite iteration of some transitions of the system.

If $x \geq 0$ then $x \leftarrow x + 2$



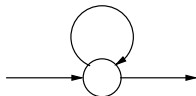
If $c_0 = \{0\}$ then

Notion of acceleration

Acceleration

Enhance convergence of the iterative procedure by computing directly the infinite iteration of some transitions of the system.

If $x \geq 0$ then $x \leftarrow x + 2$

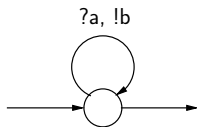


If $c_0 = \{0\}$ then $\text{post}^*(c_0) = 2.\mathbb{N}$ in one step.

Notion of acceleration

Acceleration

Enhance convergence of the iterative procedure by computing directly the infinite iteration of some transitions of the system.

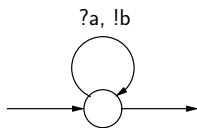


If $c_0 = a^*$ then

Notion of acceleration

Acceleration

Enhance convergence of the iterative procedure by computing directly the infinite iteration of some transitions of the system.



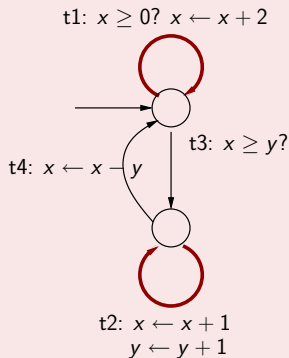
If $c_0 = a^*$ then $\text{post}^*(c_0) = a^* \cdot b^*$ in one step.

Different levels of acceleration

Loop acceleration

Compute the transitive closure of **simple loops** of the control graph.

Here: $t1^*$ and $t2^*$.

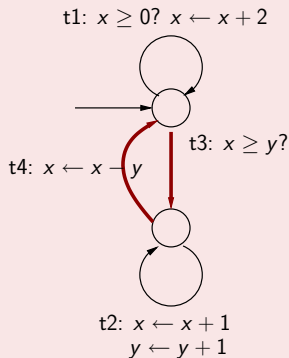


Different levels of acceleration

Flat acceleration

Compute the transitive closure of **any circuit** of the control graph.

Here: $t1^*$ and $t2^*$, but also $(t3 \cdot t4)^*$

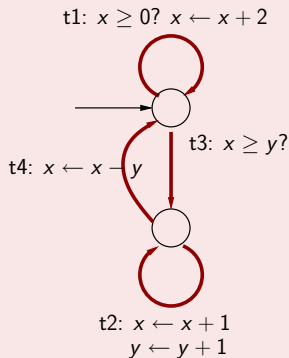


Different levels of acceleration

Global acceleration

Compute the transitive closure of **any regular language over transitions**.

Here: $t1^*$, $t2^*$, $(t1 \cdot t3 \cdot t2 \cdot t4)^*$ but also $(t1^* \cdot t3 \cdot t2^* \cdot t4)^*$



Formal definition

Different levels of acceleration

A symbolic framework SF supports

- 1 **loop acceleration** if there exists a recursive function $\text{POST_STAR} : \Sigma \times L \rightarrow L$ such that $\forall a \in \Sigma, \forall \mathbf{x} \in L$,
 $\llbracket \text{POST_STAR}(a, \mathbf{x}) \rrbracket = \llbracket a \rrbracket^* (\llbracket \mathbf{x} \rrbracket)$;
- 2 **flat acceleration** if there exists a recursive function $\text{POST_STAR} : \Sigma^* \times L \rightarrow L$ such that $\forall \pi \in \Sigma^*, \forall \mathbf{x} \in L$,
 $\llbracket \text{POST_STAR}(\pi, \mathbf{x}) \rrbracket = \llbracket \pi \rrbracket^* (\llbracket \mathbf{x} \rrbracket)$;
- 3 **global acceleration** if there exists a recursive function $\text{POST_STAR} : \text{RegExp}(\Sigma) \times L \rightarrow L$ such that for any regular expression a over Σ , for any $\mathbf{x} \in L$,
 $\llbracket \text{POST_STAR}(a, \mathbf{x}) \rrbracket = \llbracket a \rrbracket (\llbracket \mathbf{x} \rrbracket)$.

In practice

Loop acceleration

- timed automata,
- Minsky machines,
- (lossy) cfsm,
- linear counter systems (with finite monoid),
- ...

In practice

Flat acceleration

- linear counter systems (with finite monoid) equipped with Presburger formulas [Boigelot, Finkel-Leroux].
- CFSM with CQDD [Bouajjani-Habermehl'99],
- non-counting CFSMs equipped with SLRE [FPS-IC03] or QDD [BGWW-SAS97],
- lossy CFSMs equipped with sre [ABJ-2000].
- \approx Restricted counter systems with arithmetics [AAB-SPIN00].

In practice

Global acceleration

- pushdown systems with regular languages
- timed automata [Comon-Jurski'99]
- Reversal-counter systems [Ibarra2002] with Presburger formulas,
- 2-dim VASS [Leroux-Sutre'04], lossy VASS and other subclasses of VASS with Presburger formulas [LS],
- semi-commutative rewriting systems with APC [Bouajjani-Muscholl-Touilli].

Comments

- Acceleration appears to be a well-spread notion in symbolic model-checking.
- Given a symbolic framework, $\text{Global} \Rightarrow \text{Flat} \Rightarrow \text{Loop}$.
- Loop acceleration: easy to obtain, but rarely sufficient to lead to fixpoint computation.
- Flat acceleration is more flexible, but requires good compositional properties of Σ , and often complex constructions for POST_STAR .
- Global acceleration is a very strong property, ensuring the effective computation of $\text{post}^*(\llbracket x \rrbracket)$ for any $x \in L$.

Comments

For Turing powerful systems, global acceleration is not available.
Then flat acceleration seems to be the best compromise.

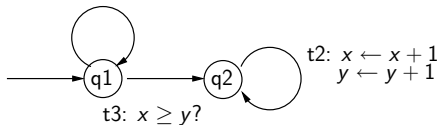
- 1 Motivations
- 2 Beyond iterative computation: Acceleration
- 3 Flat acceleration in depth
- 4 Framework of flat acceleration, applications
- 5 Conclusion

Flat systems

For which systems does flat acceleration ensure convergence?

A system is flat iff the control graph has no nested loop.

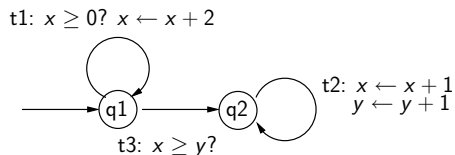
t1: $x \geq 0?$ $x \leftarrow x + 2$



Flat systems

For which systems does flat acceleration ensure convergence?

A system is flat iff the control graph has no nested loop.



Reachability set computation:

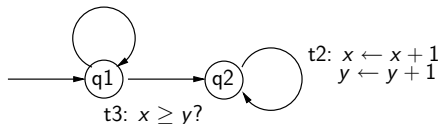
$$\begin{aligned}
 &(\{x = 0 \wedge y = 0\}, q1) \xrightarrow{t1^*} (\{\exists k, x = 2k \wedge y = 0\}, q1) \xrightarrow{t3} \\
 &(\{\exists k, x = 2k \wedge y = 0\}, q2) \xrightarrow{t2^*} (\{\exists \theta, \exists k, x = 2k + \theta \wedge y = \theta\}, q2)
 \end{aligned}$$

Flat systems

For which systems does flat acceleration ensure convergence?

A system is flat iff the control graph has no nested loop.

t1: $x \geq 0? \ x \leftarrow x + 2$



Given a symbolic framework (I, L) supporting flat acceleration, if S is flat then for all $x_0 \in L$, $\text{post}^*(\llbracket x_0 \rrbracket)$ is effectively definable in L .

Flattening

The issue is to deal with a non flat system S .

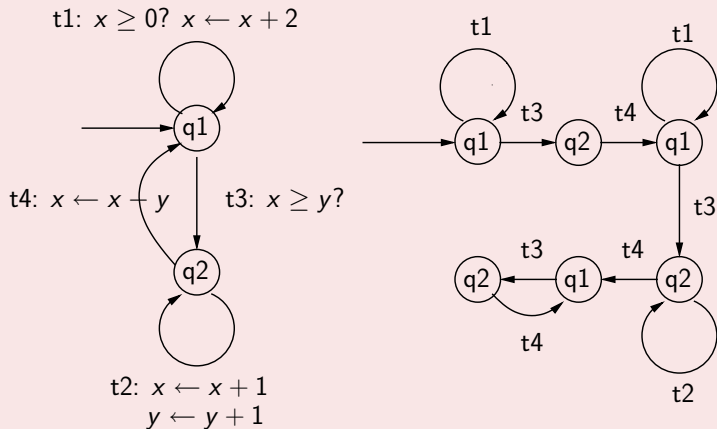
Remark that if

- 1 we know a flat system S' such that S and S' are equivalent w.r.t. reachability, and
- 2 we can compute $\text{post}_S^*(c)$ from $\text{post}_{S'}^*(c)$

Then we can compute easily $\text{post}_S^*(c)$.

A way to achieve these conditions is to consider flattenings (\approx unfoldings) of S .

Flattening - 2



Flattening

$S' = (Q', \Sigma, T', D, \llbracket \cdot \rrbracket)$ is a flattening of $S = (Q, \Sigma, T, D, \llbracket \cdot \rrbracket)$ if

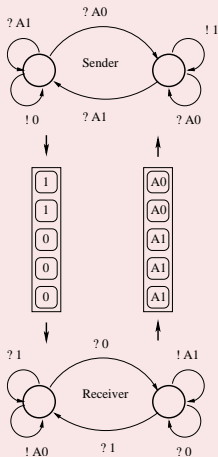
- ① S' is flat
- ② there is a mapping $z : Q' \rightarrow Q$ such that $\forall (q_1, w, q_2) \in T', (z(q_1), w, z(q_2)) \in T$.

flattable

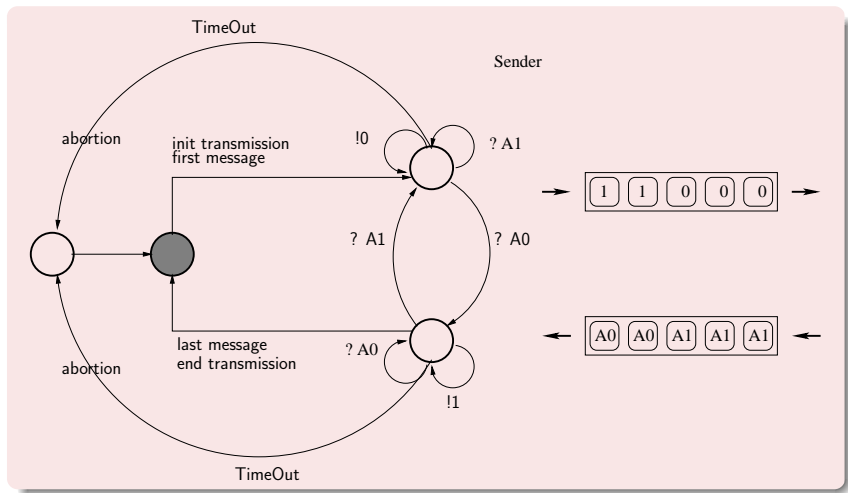
A system $S = (Q, \Sigma, T, D, \llbracket \cdot \rrbracket)$ is *L-forward flattable* iff for any $x \in L$, there exists a flattening $S' = (Q', \Sigma, T', D, \llbracket \cdot \rrbracket)$ of S and $x' \in L$ such that S and S' are equivalent w.r.t. reachability.

Let S be a L-forward flattable system supporting flat acceleration. Then $\text{post}^*(\llbracket x \rrbracket)$ is effectively L-definable.

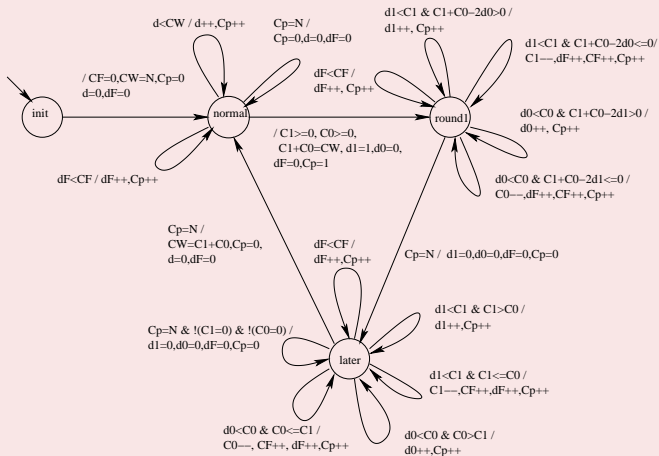
Flattable systems everywhere!!



Flattable systems everywhere!!



Flattable systems everywhere!!



Flattable systems everywhere!!

More generally

- timed automata [Comon-Jurski'99],
- 2-dim VASS [Leroux-Sutre'04], lossy VASS and other subclasses of VASS, k-reversal counter machines [to appear]
- WSTS (backward from upward closed sets)
- most successful case-studies from ALV, FAST, LASH, TREX, and so on.

Limits

- “Is a system S flattable?” is undecidable
- For some systems S , $\text{post}^*(c_0)$ is L -definable for all c_0 , but S is still not flattable (*lossy channels*).

Procedure

Procedure for flattable systems:

- 1 enumerate all flattenings S' of S
- 2 compute $\text{post}_{S'}^*(c')$, test if it is the fixpoint of S
 - yes: return
 - no: goto 1

Flattenings are not very easy to manipulate.

A restricted regular linear expression over Σ is a regular expression of the form $a_1^* a_2^* \dots a_n^*$, where $a_i \in \Sigma^*$.

A system $S = (Q, \Sigma, T, D, \llbracket \cdot \rrbracket)$ is L-forward flattable iff for all $x \in L$, there exists a rlre ρ over T such that $\text{post}^*(\llbracket x \rrbracket) = \text{post}(\rho, \llbracket x \rrbracket)$ (computable with flat acceleration).

Procedure - 2

```
procedure REACH2( $x_0$ )  
input:  $x_0 \in L$   
1:  $x \leftarrow x_0$   
2: while  $\text{POST}(x) \not\sqsubseteq x$  do  
3:   Choose fairly  $w \in T^*$   
4:    $x \leftarrow \text{POST\_STAR}(w, x)$   
5: end while  
6: return  $x$ 
```

- 1 When REACH2 terminates, $\llbracket \text{REACH2}(x_0) \rrbracket = \text{post}^*(\llbracket x_0 \rrbracket)$
(*partial correction*).
- 2 REACH2 terminates on any input iff S is L-forward flappable
(*termination*).

Refinements

procedure REACH3(x_0)

parameters: S, L

input: $x_0 \in L$

1: $x \leftarrow x_0 ; k \leftarrow 0$

2: $k \leftarrow k + 1$

3: **start**

4: **while** POST(x) $\not\sqsubseteq x$ **do** /* k-flattable */

5: Choose fairly $w \in T^{\leq k}$

6: $x \leftarrow \text{POST_STAR}(w, x)$

7: **end while** /* end k-flattable */

8: **with**

9: **when** Watchdog stops **goto** 2

10: **return** x

Refinements -2

REACH3 still correct and complete for flattable systems.

Technical issues:

- implementation of Choose
- implementation of Watchdog

Still a problem: cardinal of $T^{\leq k}$. Adapt reduction techniques [Finkel-Leroux 2002].

Refinements -2

Reduction

Replace $T^{\leq k}$ by T'_k such that

- equivalent w.r.t. reachability. $\text{post}^*(T^{\leq k}) = \text{post}^*(T'_k)$
- flat acceleration can still be performed on T'_k
- $|T'_k|$ is much smaller than $|T^{\leq k}|$

Refinements -2

Example: reduction technique of FAST [Finkel-Leroux 2002].

- apply to finite monoid linear systems
- $\Phi_1? x' \leftarrow f(x)$ and $\Phi_2? x' \leftarrow f(x)$: transform into $\Phi_1 \vee \Phi_2? x' \leftarrow f(x)$.
- $|T'_k|$ is polynomial in k .

Remark

- acceleration of particular nested loops.
- go outside strict flat acceleration.

- 1 Motivations
- 2 Beyond iterative computation: Acceleration
- 3 Flat acceleration in depth
- 4 Framework of flat acceleration, applications
- 5 Conclusion

Framework

Key points

- 1 a system S
- 2 a symbolic framework (I, L)
- 3 an acceleration function
- 4 a procedure to find cycles

Problems

- 1 S closed enough to the real world?
- 2 $\text{post}^*(c_0)$ not representable in L
- 3 $\text{post}^*(c_0)$ not computable through acceleration
- 4 practical (time, memory)

Comparison of techniques/tools

ALV, FAST, LASH and TRES are very closed tools designed to compute reachability sets of counter systems.

	ALV	LASH	FAST	TRES
system	full	linear		restricted
symbolic fram.	Presburger (automata)			Arith. (<i>undec.</i> \sqsubseteq)
acceleration	no	flat		flat (<i>partial. rec.</i>)
termination	0-F	1-F	F	k-F (+ oracle \sqsubseteq)

F: flattable systems.

k-F: flattable using elementary cycles of length $\leq k$.

System	ALV	LASH	FAST
TTP	no	yes	yes (1)
prod/cons (2)	no	yes	yes (1)
prod/cons (N)	no	no	yes (2)
lift control, N	no	no	yes (2)
train	no	no	yes (2)
consistency	no	no	yes (3)
CSM, N	no	no	yes (2)
PNCSA	no	no	no
IncDec	no	no	no
BigJAVA	no	no	no

forward computation

yes = termination within 1200 seconds.

Experiments are closely related with the comparison through our framework.

Improvement of existing tools

TREX for lossy channels [??]

- system: lossy channels systems (*?read, !write*)
- symbolic framework: SRE $((a + \varepsilon) \cdot (a + b + c)^* + \dots)$
- flat acceleration
- **procedure**: search cycles of length $\leq k$ (k statically defined)

Propositions to improve TREX

TREX respects almost all the flat acceleration framework.

- increase k dynamically to have a complete heuristics,
- adapt reduction techniques to reduce the number of cycles

- 1 Motivations
- 2 Beyond iterative computation: Acceleration
- 3 Flat acceleration in depth
- 4 Framework of flat acceleration, applications
- 5 **Conclusion**

Conclusion (1)

Summary

- A generic framework for acceleration
- Three levels of acceleration (loop, flat, global). For each case,
 - characterization of termination
 - complete procedure to compute $\text{post}^*(c_0)$ when feasible
- Refinements for flat acceleration
 - heuristic
 - algorithmic

Conclusion (2)

Accelerated symbolic model-checking

- Acceleration is a central issue in infinite model-checking.
- Flat acceleration is a good compromise: most usual systems support it, many subclasses are flattable, and many successful case-studies (FAST, LASH, TREX).

Framework for flat acceleration

A generic framework for acceleration in 4 key points

- clarify and unify existing work
- meaningful comparison of techniques/tools
- guidelines to build new tools or improve existing ones.
- given a symbolic framework and a flat acceleration we provide an efficient heuristic, complete for flattable systems.