Fast: Theory and practice of acceleration

Sébastien Bardin

LSV - CNRS & ÉNS de Cachan

6 mars 2006
Verification of reactive systems

Reactive systems
- Software and/or hardware
- Autonomous
- Critical
The TTP protocol

Processors embedded in cars

The TTP protocol
- fault tolerance
- ensure no fault will propagate

TTP is supported by Audi, PSA, Renault, ...
A model of the TTP [Bouajjani-Merceron 2002]

If \( d < N \) Do
\[
\begin{align*}
    d &:= d + 1; \\
    C_p &:= C_p + 1
\end{align*}
\]
End Do

\[ N : \text{number of processors} \]
\[ C_p : \text{round number} \]
\[ d : \text{number of processors having broadcast so far} \]
A model of the TTP [Bouajjani-Merceron 2002]

Intuition- Motivations
Question
In the red location, does $C_p = N \Rightarrow (C_0 = 0 \lor C_1 = 0)$?

Objective
Automatic verification for any value of $N$
Our problem: counter system verification

Counter systems

- we study mathematical models of concrete systems
- automata extended with unbounded integer variables

Properties to check

Reachability properties = properties of reachable configurations.
- useful: mutual exclusion, deadlock freedom, ...
- easy to check from the reachability set.
Our problem: counter system verification

Problems

- Undecidable for two counters with $(+1, -1, \neq 0)$
- One of the issues: infinite reachability set
Back to the finite case
Back to the finite case
Back to the finite case
Back to the finite case
Back to the finite case

Initial

Unsafe

?
Extend verification to infinite systems

Enumerative methods do not work any more

Algorithms for decidable subclasses
- Petri nets,
- timed automata, ...

or Semi-algorithms to compute the reachability set
- more expressive/realistic systems
- no guarantee of termination, we hope practical termination
- Extend iterative computation for infinite sets
  - (symbolic model-checking)
Extend verification to infinite systems

Enumerative methods do not work any more

Algorithms for decidable subclasses
- Petri nets,
- timed automata, ...

or Semi-algorithms to compute the reachability set
- more expressive/realistic systems
- no guarantee of termination, we hope practical termination
- Extend iterative computation for infinite sets
- (symbolic model-checking)
Extend verification to infinite systems

Enumerative methods do not work any more

Algorithms for decidable subclasses
- Petri nets,
- timed automata, ...

or **Semi-algorithms to compute the reachability set**
- more expressive/realistic systems
- no guarantee of termination, we hope **practical termination**
- Extend iterative computation for infinite sets
- *(symbolic model-checking)*
Issue 1: infinite set of reachable configurations.

Idea = manipulate infinite sets of configurations

- sets are represented symbolically.
- need basic symbolic operations \( \text{POST}, \sqcup, \sqsubseteq \).

Example: intervals of integers

- Formula \( \phi_X : \{ x > 5 \} \) means that \( x \) ranges over all integers greater than 5.

- After transition \( y := x + 1 \), the possible values of \( y \) are exactly represented by \( \phi_Y = \{ y > 6 \} \).
Issue 1: infinite set of reachable configurations.

Idea = manipulate infinite sets of configurations
- sets are represented symbolically.
- need basic symbolic operations \( \text{POST}, \sqcup, \sqsubseteq \).

Example: intervals of integers
- Formula \( \phi_X : \{x > 5\} \) means that \( x \) ranges over all integers greater than 5.
- After transition \( \frac{y := x + 1}{\text{POST}} \), the possible values of \( y \) are exactly represented by \( \phi_Y = \{y > 6\} \).
First (and basic) symbolic procedure

Iterative computation of $\text{post}^*_S(X_0)$

1. $X \leftarrow X_0$
2. If $\text{POST}(X) \subseteq X$ Goto 5
3. $X \leftarrow \text{POST}(X) \sqcup X$
4. Goto 2
5. Return $X$

Issue 2: termination is scarce because of circuits in the control graph ...

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

If $X_0 = \{0\}$ then
First (and basic) symbolic procedure

Iterative computation of $\text{post}^*(X_0)$

1. $X \leftarrow X_0$
2. If $\text{POST}(X) \subseteq X$ Goto 5
3. $X \leftarrow \text{POST}(X) \sqcup X$
4. Goto 2
5. Return $X$

Issue 2: termination is scarce because of circuits in the control graph ...

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

If $X_0 = \{0\}$ then $X = \{0\}$
First (and basic) symbolic procedure

Iterative computation of \( \text{post}^*_S(X_0) \)

1. \( X \leftarrow X_0 \)
2. If \( \text{POST}(X) \subseteq X \) Goto 5
3. \( X \leftarrow \text{POST}(X) \sqcup X \)
4. Goto 2
5. Return \( X \)

Issue 2: termination is scarce
because of circuits in the control graph ...

If \( x \geq 0 \) Do \( x \leftarrow x + 2 \)

If \( X_0 = \{0\} \) then \( X = \{0, 2\} \)
First (and basic) symbolic procedure

Iterative computation of \( \text{post}^S(X_0) \)

1. \( X \leftarrow X_0 \)
2. If \( \text{POST}(X) \sqsubseteq X \) Goto 5
3. \( X \leftarrow \text{POST}(X) \sqcup X \)
4. Goto 2
5. Return \( X \)

**Issue 2: termination is scarce**

because of circuits in the control graph ...

If \( x \geq 0 \) Do \( x \leftarrow x + 2 \)

If \( X_0 = \{0\} \) then \( X = \{0, 2, \ldots, 2k\} \)
Circuit acceleration

Enhance the convergence of the iterative symbolic procedure by computing in one step the iteration of a sequence of transitions (circuit).

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

If $X_0 = \{0\}$ then
Enhance the convergence of the iterative symbolic procedure by computing in one step the iteration of a sequence of transitions (circuit).

```
If x ≥ 0 Do x ← x + 2
```

If $X_0 = \{0\}$ then $\text{post}^*(X_0) = 2 \mathbb{N}$, in one step.
About acceleration of counter systems

State-of-the-art in 2002

(Karp-Miller 1969)
(Fribourg 1990)
[Boigelot-Wolper 1994],
[Boigelot-Wolper 1998],
[Annichini-Asarin-Bouajjani 2000], (+ temps)
[Finkel-Leroux 2002],

Remarks

1. very different techniques, no unifying view (comparison?)
2. still a gap between acceleration algorithm and fixpoint computation (how to select circuits?)
About acceleration of counter systems

State-of-the-art in 2002

*(Karp-Miller 1969)*

*(Fribourg 1990)*

[Boigelot-Wolper 1994],

[Boigelot-Wolper 1998],

[Annichini-Asarin-Bouajjani 2000], (+ temps)

[Finkel-Leroux 2002],

Remarks

1. very different techniques, no unifying view *(comparison?)*

2. still a gap between acceleration algorithm and fixpoint computation *(how to select circuits?)*
Symbolic computation in counter systems, 2002

- **A symbolic representation: automata**
  - DFA [Boudet-Comon 1996],
  - NDD [Wolper-Boigelot 2000]).

- **Various circuit acceleration algorithms**
  - \( f(x) = M.x + v \) with *finite monoid* and convex guard [Boigelot 1998]
  - \( f(x) = M.x + v \) with *finite monoid* and Presburger guard [Finkel-Leroux 2002]
  - functions “à la” timed automata [Annichini-Asarin-Bouajjani 2000]

- **Circuit selection** no argued heuristic

- **Tools**
  - **ALV** (no acceleration) [Bultan]
  - **LASH** (acceleration but no circuit selection) [Wolper]
  - **TReX** (acceleration and circuit selection, but no argument) [Bouajjani]
Symbolic computation in counter systems, 2002

- A symbolic representation: automata
  - DFA [Boudet-Comon 1996],
  - NDD [Wolper-Boigelot 2000]).

- Various circuit acceleration algorithms
  - $f(x) = M.x + v$, with finite monoid and convex guard [Boigelot 1998]
  - $f(x) = M.x + v$ with finite monoid and Presburger guard [Finkel-Leroux 2002]
  - functions “à la” timed automata [Annichini-Asarin-Bouajjani 2000]

- Circuit selection no argued heuristic

- Tools
  - ALV (no acceleration) [Bultan]
  - LASH (acceleration but no circuit selection) [Wolper]
  - TReX (acceleration and circuit selection, but no argument) [Bouajjani]
A symbolic representation: automata
- DFA [Boudet-Comon 1996],
- NDD [Wolper-Boigelot 2000]).

Various circuit acceleration algorithms
- \( f(x) = M.x + v \), with finite monoid and convex guard [Boigelot 1998]
- \( f(x) = M.x + v \) with finite monoid and Presburger guard [Finkel-Leroux 2002]
- functions "à la" timed automata [Annichini-Asarin-Bouajjani 2000]

Circuit selection no argued heuristic

Tools
- \textsc{Alv} (no acceleration) [Bultan]
- \textsc{Lash} (acceleration but no circuit selection) [Wolper]
- \textsc{Trex} (acceleration and circuit selection, but no argument) [Bouajjani]
Symbolic computation in counter systems, 2002

- A symbolic representation: automata
  - DFA [Boudet-Comon 1996],
  - NDD [Wolper-Boigelot 2000]).

- Various circuit acceleration algorithms
  - \( f(x) = M.x + v \), with finite monoid and convex guard [Boigelot 1998]
  - \( f(x) = M.x + v \) with finite monoid and Presburger guard [Finkel-Leroux 2002]
  - functions “à la” timed automata [Annichini-Asarin-Bouajjani 2000]

- Circuit selection no argued heuristic

- Tools
  - ALV (no acceleration) [Bultan]
  - LASH (acceleration but no circuit selection) [Wolper]
  - TREX (acceleration and circuit selection, but no argument) [Bouajjani]
Symbolic computation in counter systems, 2002

- **A symbolic representation: automata**
  - DFA [Boudet-Comon 1996],
  - NDD [Wolper-Boigelot 2000]).

- **Various circuit acceleration algorithms**
  - \( f(x) = M.x + v \), with *finite monoid* and convex guard [Boigelot 1998]
  - \( f(x) = M.x + v \) with *finite monoid* and Presburger guard [Finkel-Leroux 2002]
  - functions “à la” timed automata [Annichini-Asarin-Bouajjani 2000]

- **Circuit selection** no argued heuristic

- **Tools**
  - **ALV** (no acceleration) [Bultan]
  - **LASH** (acceleration but no circuit selection) [Wolper]
  - **TReX** (acceleration and circuit selection, but no argument) [Bouajjani]
Issues / Results

Issue : Various acceleration techniques
Results : Unifying framework encompassing most of acceleration theorems [ATVA’05]

Issue : How to select circuits?
Results : Maximal heuristic, efficient in practice [CAV’03, ATVA’05]

Issue : Improve practical efficiency of acceleration
Results : “Convex acceleration” algorithm [TACAS’04]

Issue : Experimentations
Results : implementation of FAST [CAV’03], Verification of the TTP [TACAS’04] and others

These works have been partially supported by ACI Persée.
Issues / Results

Issue : Various acceleration techniques
Results : Unifying framework encompassing most of acceleration theorems [ATVA’05]

Issue : How to select circuits?
Results : Maximal heuristic, efficient in practice [CAV’03, ATVA’05]

Issue : Improve practical efficiency of acceleration
Results : “Convex acceleration” algorithm [TACAS’04]

Issue : Experimentations
Results : implementation of FAST [CAV’03], Verification of the TTP [TACAS’04] and others

These works have been partially supported by ACI Persée.
Issues / Results

Issue: Various acceleration techniques
Results: Unifying framework encompassing most of acceleration theorems [ATVA’05]

Issue: How to select circuits?
Results: Maximal heuristic, efficient in practice [CAV’03, ATVA’05]

Issue: Improve practical efficiency of acceleration
Results: “Convex acceleration” algorithm [TACAS’04]

Issue: Experimentations
Results: implementation of FAST [CAV’03],
Verification of the TTP [TACAS’04] and others

These works have been partially supported by ACI PERSÉE.
Issues / Results

Issue: Various acceleration techniques
Results: Unifying framework encompassing most of acceleration theorems [ATVA’05]

Issue: How to select circuits?
Results: Maximal heuristic, efficient in practice [CAV’03, ATVA’05]

Issue: Improve practical efficiency of acceleration
Results: “Convex acceleration” algorithm [TACAS’04]

Issue: Experimentations
Results: implementation of FAST [CAV’03], Verification of the TTP [TACAS’04] and others

These works have been partially supported by ACI Persée.
Issues / Results

Issue: Various acceleration techniques
Results: Unifying framework encompassing most of acceleration theorems [ATVA’05]

Issue: How to select circuits?
Results: Maximal heuristic, efficient in practice [CAV’03, ATVA’05]

Issue: Improve practical efficiency of acceleration
Results: “Convex acceleration” algorithm [TACAS’04]

Issue: Experimentations
Results: implementation of FAST [CAV’03], Verification of the TTP [TACAS’04] and others

These works have been partially supported by ACI PERSÉE.
Outline

1. Introduction
2. Counter systems
3. Circuit acceleration
4. Circuit selection
5. The tool FAST
6. Applications
7. Conclusion
Presburger arithmetic

First order arithmetics without $\times$-operator

$$\phi ::= t \leq t | \neg \phi | \phi \lor \phi | \exists k. \phi | true$$

$$t ::= 0 | 1 | y | t - t | t + t.$$
Counter systems

- finite set of $m$ variables $x, y, z, \ldots$ over $\mathbb{N}$
- finite set of P-affine functions $f = (M, v, G)$
  - $G \subseteq \mathbb{N}^m$ Presburger guard
  - $M$ square matrix
  - $v$ vector
- $\overrightarrow{\text{var}}' = f(\overrightarrow{\text{var}})$ iff
  - $\overrightarrow{\text{var}} \in G$
  - and $\overrightarrow{\text{var}}' = M.\overrightarrow{\text{var}} + v$
Symbolic representations: Automata

Automata to recognize sets of integer vectors

- a non-negative integer in basis 2 is a word over \{0, 1\}
- automata recognize sets of words
- extensions
  - any integer: 2-complement encoding
  - vectors: tuples of letters, or variables entanglement

Presburger sets (and a little bit more) are recognized by automata.

Common operations on sets \[\rightarrow\] standard operations on automata
Symbolic representations: Automata

DFA [Boudet-Comon 1996],
NDD [Wolper-Boigelot 2000].
1 Introduction
2 Counter systems
3 Circuit Acceleration
4 Circuit Selection
5 The tool FAST
6 Applications
7 Conclusion
Monoid of a function $f = (M, \nu, G)$: $\{1, M, M^2, \ldots, M^n, \ldots\}$

**Theorem [Finkel-Leroux 2002]**

Let $f = (M, \nu, G)$ a P-affine function with finite monoid. Then $f^*$ is effectively defined by a Presburger formula

$$f^* = \{(x, x') | x \in G \land \exists k \geq 0. x' = \bar{f}^k(x) \land \forall i. 0 \leq i < k, \bar{f}^i(x) \in G\}$$

Building the formula is 3-EXP in $|\mathcal{A}(G)|$, $|\nu|_{max}$, $|M|_{max}$ et $m$. 
Idea of the algorithm

- $f = (M, v, G)$ with finite monoid $< M >$.
- $\bar{f} : \mathbb{Z}^m \to \mathbb{Z}^m, \forall x \in \mathbb{Z}^m, \bar{f}(x) = M.x + v$

- $< M >$ finite, then $\exists (a, b) \in \mathbb{N} \times \mathbb{N}$ such that $M^{a+b} = M^a$
- We deduce that $\forall n \in \mathbb{N}, \forall x \in \mathbb{Z}^m, \bar{f}^{a+n.b} = \bar{f}^a(x) + n.M^a.\bar{f}^b(0)$
- It comes that $\bar{F} = \{(i, x, x') \in \mathbb{N} \times \mathbb{Z}^m \times \mathbb{Z}^m, x' = \bar{f}^i(x)\} \iff \bigvee_{r=0}^{a-1} \{(i, x, x') | x' = \bar{f}^r(x) \land i = r\} \bigvee_{r=0}^{b-1} \{(i, x, x') | \exists n \geq 0 (x' = \bar{f}^{a+r}(x) + n.M^{a+r}.\bar{f}^b(0)) \land (i = a + r + n.b)\}$

- $f^* = \{(x, x'), \exists i \geq 0, x' = f^i(x)\} \iff \{(x, x'), \exists i \geq 0 [(i, x, x') \in \bar{F} \land (\forall k(0 \leq k < i), \exists x'' \in G, (k, x, x'') \in \bar{F})]\}$
Faster acceleration ...

Convex translations [TACAS’04]

\[ f = (I_m, \nu, G) \] where \( I_m \) is the identity matrix and \( G \) convex

- No need to test if the predecessors are in the guard.
- The construction can be simplified.

Theorem [TACAS’04]

Convex acceleration is quadratic in \(|A(G)|\).
Faster acceleration ... complexity

<table>
<thead>
<tr>
<th>parameter</th>
<th>magnitude</th>
<th>standard algorithm</th>
<th>convex algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>A(G)</td>
<td>$</td>
<td>100.000</td>
</tr>
<tr>
<td>$m$</td>
<td>5-50</td>
<td>3-EXP</td>
<td>EXP</td>
</tr>
<tr>
<td>$</td>
<td>v</td>
<td>_{max}$</td>
<td>$\leq 10$</td>
</tr>
<tr>
<td>$</td>
<td>M</td>
<td>_{max}$</td>
<td>$\leq 10$</td>
</tr>
</tbody>
</table>
$\mathcal{A}(f^*) = \text{automaton representing } f^* \text{ (transducer)}$

| $|\mathcal{A}(f^*)|$ | Time (seconds) | Memory (MB) |
|-----------------|----------------|--------------|
|                 | Standard/Convex | Standard/Convex |
| 16,766          | 10/7            | 31/13        |
| 26,409          | 5/2             | 17/18        |
| 41,950          | 18/10           | 52/30        |
| 190,986 (TTP)   | 50/9            | 400/140      |
| 380,332 (TTP)   | ↑↑↑/34          | ↑↑↑/534      |
| ?               | ↑↑↑/ >900       | ↑↑↑/ >500    |

Acceleration - Convex acceleration
Outline

1. Introduction
2. Counter systems
3. Circuit acceleration
4. Circuit selection
5. The tool FAST
6. Applications
7. Conclusion
What is computable with circuit acceleration?

First answer
Flat system \( = \) at most 1 elementary circuit on each control node

\[ t_1: x \geq 0? \ x \leftarrow x + 2 \]

\[ t_2: x \leftarrow x + 1 \]
\[ y \leftarrow y + 1 \]

\[ t_3: x \geq y? \]

Circuit acceleration + flat \( S = \) computation of \( \text{post}^*_S(X_0) \)
S' is a flattening of S iff 
S' is flat and S simulates S'

Flattenings of non-flat systems
(S, X₀) is fltable iff ∃S' such that
1. S' is a flattening of S
2. S and S' are equivalent for reachability.

Theorem 1 [ATVA’05]

post*(X₀) is computable by circuit acceleration iff (S, X₀) is fltable.
Other characterization

A restricted linear regular expression (rlre) over \( T \)
\[
\rho = w_1^* w_2^* \ldots w_n^*, \text{ where } w_i \in T^*.
\]

\( \text{post}(\rho, X) = \) configurations reachable following transitions in \( \rho \)

Theorem 2 [ATVA’05]

\( \text{post}^*(X_0) \) is computable by circuit acceleration \( \text{iff} \) \( \exists \) a rlre \( \rho \) over \( T \) such that \( \text{post}^*(X_0) = \text{post}(\rho, X_0) \).

Remark: \( \text{post}(\rho, X) \) is computable with circuit acceleration
A restricted linear regular expression (rlre) over $T$

$$\rho = w_1^* w_2^* \ldots w_n^*$$, where $w_i \in T^*$.

$\text{post}(\rho, X) =$ configurations reachable following transitions in $\rho$

**Theorem 2 [ATVA’05]**

$\text{post}^*(X_0)$ is computable by circuit acceleration iff $\exists$ a rlre $\rho$ over $T$ such that $\text{post}^*(X_0) = \text{post}(\rho, X_0)$.

Remark: $\text{post}(\rho, X)$ is computable with circuit acceleration
A restricted linear regular expression (rlre) over $T$
\[ \rho = w_1^* w_2^* \ldots w_n^*, \text{ where } w_i \in T^*. \]

post($\rho, X$) = configurations reachable following transitions in $\rho$

**Theorem 2 [ATVA’05]**

post*$^*(X_0)$ is computable by circuit acceleration \iff \exists a rlre $\rho$ over $T$ such that post*$^*(X_0) = \text{post}(\rho, X_0)$.

Remark: post($\rho, X$) is computable with circuit acceleration
Selection Heuristic (I)

Input: \((S, X_0)\)

1. \(X \leftarrow X_0; \ k \leftarrow 0\)
2. \(k \leftarrow k + 1\)
3. Lunch
4. If \(post(X) \subseteq X\) Goto 10
5. Enumerate the next \(\rho\) rlre over \(T\)
6. \(X \leftarrow post(\rho, X)\)
7. Goto 4
8. In parallel with
9. When \(Watchdog\) stops Goto 2
10. Return \(X\)

maximal procedure: terminates iff \((S, X_0)\) is flattable

PB1(time) find quickly a good rlre

PB2(space) avoid as much as possible unnecessary expensive steps of computations
Selection Heuristic (I)

Input: \((S, X_0)\)

1. \(X \leftarrow X_0; \ k \leftarrow 0\)
2. \(k \leftarrow k + 1\)
3. **Lunch**
4. If \(\text{post}(X) \subseteq X\) Goto 10
5. *Choose fairly* \(w \in T^{\leq k}\)
6. \(X \leftarrow \text{post}(w^*, X)\)
7. Goto 4
8. **In parallel with**
9. **When** *Watchdog* stops **Goto** 2
10. Return \(X\)

The procedure is still maximal

PB1(time) partitioning + *Watchdog*

PB2(space) *Choose*
Results

The selection heuristic design is reduced to designing

- Choose (a standard solution is given)
- Watchdog (a standard solution is given)

The obtained procedure is then

- maximal
- efficient: good results on counter systems (cf. FAST)
| $T|^{\leq k}$ may be exponential in $k$.

Idea = reduce $|T|^{\leq k}$ by removing redundant functions.

Three reductions:

- **union-reduction** [Finkel-Leroux 2002]
  - if $f = (M, v, G_1)$ and $g = (M, v, G_2)$,
  - let $h = (M, v, G_1 \lor G_2)$
  - then $(f + g)^* = h^*$

- **commutation-reduction** [CAV’03]
  - if $f$ and $g$ commute then $f^*g^* = (f \cdot g)^* = (g \cdot f)^*$

- **conjugacy-reduction** [ATVA’05]
  - $(f_2 \cdot f_3 \cdot f_1)^* = l_d + f_2 \cdot f_3 \cdot (f_1 \cdot f_2 \cdot f_3)^* \cdot f_1$
| $T| \leq k$ may be exponential in $k$.

Idea = reduce $|T| \leq k$ by removing redundant functions.

Three reductions:

- **union-reduction** [Finkel-Leroux 2002]
  - if $f = (M, \nu, G_1)$ and $g = (M, \nu, G_2)$,
  - let $h = (M, \nu, G_1 \lor G_2)$
  - then $(f + g)^* = h^*$

- **commutation-reduction** [CAV’03]
  - if $f$ and $g$ commute then $f^* g^* = (f \cdot g)^* = (g \cdot f)^*$

- **conjugacy-reduction** [ATVA’05]
  - $(f_2 \cdot f_3 \cdot f_1)^* = l_d + f_2 \cdot f_3 \cdot (f_1 \cdot f_2 \cdot f_3)^* \cdot f_1$
| $T| \leq k$ may be exponential in $k$.

Idea = reduce $|T| \leq k$ by removing redundant functions.

Three reductions:

- **union-reduction** [Finkel-Leroux 2002]
  - if $f = (M, \nu, G_1)$ and $g = (M, \nu, G_2)$,
  - let $h = (M, \nu, G_1 \lor G_2)$
  - then $(f + g)^* = h^*$

- **commutation-reduction** [CAV’03]
  - if $f$ and $g$ commute then $f^*g^* = (f \cdot g)^* = (g \cdot f)^*$

- **conjugacy-reduction** [ATVA’05]
  - $(f_2 \cdot f_3 \cdot f_1)^* = l_d + f_2 \cdot f_3 \cdot (f_1 \cdot f_2 \cdot f_3)^* \cdot f_1$
\(|T| \leq k\) may be exponential in \(k\).

Idea = reduce \(|T| \leq k\) by removing redundant functions.

Three reductions:

- **union-reduction** [Finkel-Leroux 2002]
  - if \(f = (M, \nu, G_1)\) and \(g = (M, \nu, G_2)\),
  - let \(h = (M, \nu, G_1 \lor G_2)\)
  - then \((f + g)^* = h^*\)

- **commutation-reduction** [CAV’03]
  - if \(f\) and \(g\) commute then \((f^* g^*) = (f \cdot g)^* = (g \cdot f)^*\)

- **conjugacy-reduction** [ATVA’05]
  - \((f_2 \cdot f_3 \cdot f_1)^* = l_d + f_2 \cdot f_3 \cdot (f_1 \cdot f_2 \cdot f_3)^* \cdot f_1\)
Practical results

| system      | $|T|$ | $k$ | $|C^{\leq k}|$ | $U$ | $C_m$ | $C_j$ | $U+C_m$ |
|-------------|-----|-----|----------------|-----|-------|-------|--------|
| csm         | 13  | 1   | 14             | 14  | 14    | 14    | 14     |
|             | 13  | 2   | 183            | 103 | 57    | 99    | 35     |
| consistency | 8   | 1   | 9              | 9   | 9     | 9     | 9      |
|             | 8   | 2   | 68             | 45  | 44    | 39    | 30     |
|             | 8   | 3   | 484            | 172 | 299   | 178   | 98     |
| swimming    | 6   | 1   | 7              | 7   | 7     | 7     | 7      |
| pool        | 6   | 2   | 43             | 21  | 24    | 25    | 16     |
|             | 6   | 3   | 259            | 56  | 114   | 97    | 28     |
|             | 6   | 4   | 1555           | 126 | 614   | 421   | 47     |
|             | 6   | 5   | 9331           | 252 | 3591  | 1977  | 86     |

$U$, $C_m$, $C_j$: reductions (union, commutation, conjugacy)
1 Introduction
2 Counter systems
3 Circuit acceleration
4 Circuit selection
5 The tool FAST
6 Applications
7 Conclusion
The previous results are implemented in **Fast**

**Fast** works well in practice

- successfully verify 80% of 40 infinite systems [CAV’03].
- first automatic verification of TTP [TACAS’04]
- first automatic verification of CES
## Technological comparison

<table>
<thead>
<tr>
<th></th>
<th>\textsc{Alv}</th>
<th>\textsc{Lash}</th>
<th>\textsc{Fast}</th>
<th>\textsc{TReX}</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>relational</td>
<td>affine</td>
<td>restricted</td>
<td></td>
</tr>
<tr>
<td>symb. rep</td>
<td></td>
<td>automata</td>
<td></td>
<td>arith. + pdbm (undec. (\leq))</td>
</tr>
<tr>
<td>acceleration</td>
<td>no</td>
<td>circuits</td>
<td></td>
<td>circuits (partial.rec.)</td>
</tr>
<tr>
<td>circuit selection</td>
<td>no</td>
<td>yes</td>
<td>yes, (\leq k)</td>
<td></td>
</tr>
</tbody>
</table>
# Practical comparison

<table>
<thead>
<tr>
<th>System</th>
<th>ALV</th>
<th>LASH</th>
<th>FAST</th>
<th>k</th>
<th>TReX</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTP (bounded)</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>Lamport (bounded)</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>Dekker (bounded)</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>ticket 2</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>kanban</td>
<td>↑</td>
<td>T</td>
<td>T</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>multipoll</td>
<td>↑</td>
<td>T</td>
<td>T</td>
<td>1</td>
<td>↑</td>
</tr>
<tr>
<td>prod/cons (2)</td>
<td>↑</td>
<td>T</td>
<td>T</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>ttp</td>
<td>↑</td>
<td>T</td>
<td>T</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>prod/cons (N)</td>
<td>↑</td>
<td>↑</td>
<td>T</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>lift control, N</td>
<td>↑</td>
<td>↑</td>
<td>T</td>
<td>2</td>
<td>T</td>
</tr>
<tr>
<td>train</td>
<td>↑</td>
<td>↑</td>
<td>T</td>
<td>2</td>
<td>T</td>
</tr>
<tr>
<td>csm, N</td>
<td>↑</td>
<td>↑</td>
<td>T</td>
<td>2</td>
<td>↑</td>
</tr>
<tr>
<td>consistency</td>
<td>↑</td>
<td>↑</td>
<td>T</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>swimming pool</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>4</td>
<td>↑</td>
</tr>
<tr>
<td>pncsa</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>?</td>
<td>↑</td>
</tr>
<tr>
<td>incdec</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>?</td>
<td>↑</td>
</tr>
<tr>
<td>bigjava</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>?</td>
<td>↑</td>
</tr>
</tbody>
</table>

**T**: success within 20 minutes  
**↑**: no success within 20 minutes  
**k**: circuit length for **FAST**  
**-**: not an input of **TReX**
Verification of TTP by FAST

Applications - The TTP protocol
Verification of TTP by **FAST**

1 error [TACAS’04]

16 transitions, 9 variables, complex guards
- automatic verification
- Pentium 4 2.4 GHz, 1 Gbyte RAM : 940 sec. and 73 Mbytes.

Other tools:
- **Alg** does not terminate
- **Lash** terminates when good circuits are provided
- **TTP** does not fit **TREX** input model.

2 errors [TACAS’04]

20 transitions, 18 variables, even more complex guards
- standard acceleration does not work
- convex acceleration + overapproximation.
Verification of TTP by Fast

1 error [TACAS’04]

16 transitions, 9 variables, complex guards

- automatic verification
- Pentium 4 2.4 GHz, 1 Gbyte RAM : 940 sec. and 73 Mbytes.

Other tools:

- ALV does not terminate
- LASH terminates when good circuits are provided
- TTP does not fit TReX input model.

2 errors [TACAS’04]

20 transitions, 18 variables, even more complex guards

- standard acceleration does not work
- convex acceleration + overapproximation.
The CES protocol - presentation

- Supported by Philips
- Multimedia streaming
- Ensures reliable communications over lossy channels

- Colored Petri net modeling of the CES,
- Infinite system, counters and queues of parameterized length
- (Complex) proofs of many properties of the CES (ex: size of the reachability set w.r.t. the buffer lengths)
The CES protocol - verification with Fast

**Modeling issues**

*Fast* does not handle queues.

- queues *simulated* by counters,
- correctness of the simulation is expressed as a reachability property of the counter system, and it is checked by *Fast* automatically.

**Results**

Properties proved in [Billington-Liu 2002] are checked easily.
Verification of pointer systems (work in progress)

Manual management of memory resources (language C)

- memory heap = collection of memory cells
- a cell contains: data or address
- addresses ∈ {valid, invalid, NULL}
- primitives: new, free, successor

Common errors
- memory violation
- memory leak

Work supported by EDF (2002-2004),
and by RNTL AVÉRILES (2005-2008)
 Programs:
  - only one successor (lists, no trees)
  - no data, only pointers

List reverse(List x) {
  List y,t;
  y=NULL;
  while (x!=NULL) {
    t=y;
    y=x;
    x=x->n;
    y->n=t;
    t=NULL;
  }
  return y;
}
Verification of pointer systems [AVIS’06, AVIS’04]

- Encode infinite sets of memory graphs by Presburger sets
- Bisimulation between the pointer system and a counter system
- Verification by FAST

A prototype is in progress (with A. Sangnier and É. Lozes)

- Works well for \( \approx 10 \) small standard examples
- Both qualitative and quantitative properties
- Allows to check programs with counters + pointers
Outline

1. Introduction
2. Counter systems
3. Circuit acceleration
4. Circuit selection
5. the tool FAST
6. Applications
7. Conclusion
Our methodology

System

Conclusion- Summary
Our methodology

System

Symb Rep

System
Our methodology

- SymbRep
- System
- Acceleration

Conclusion - Summary
Our methodology

Symb Rep Acceleration Heuristic System

Symb Rep Acceleration
Results

1. Generic methodology [ATVA’05]
   - unified acceleration framework
   - power and limits (flattable systems)
   - maximal circuit selection
   - generic optimizations (reductions)

2. Instantiation to counter systems
   - two acceleration algorithms
     - [Finkel-Leroux 2002]
     - [TACAS’04]
   - a reduction fit to counters [Finkel-Leroux 2002]
   - The tool FAST
3. Many experimentations

- **Counter systems**
  - 40 infinite systems [CAV’03]
  - TTP [TACAS’04]

- **Counters + queues**
  - CES (in my PhD thesis, work with Laure Petrucci)
  - *Stop and Wait Protocol* [Billington-Gallasch-Petrucci 2005]

- **Pointer systems**
  - translation into counter systems [AVIS’06, AVIS’04]
  - prototype, works on 10 standard examples (work with Étienne Lozes and Arnaud Sangnier)
Perspectives

Short term

A new version FASTER is released [submission CAV’06]

- acceleration engine totally independent of the Presburger library
- a Presburger package based on shared automata and cache computation (Couvreur)
- a richer user language

Future works

Scale-up our methods: abstract-refine and checks methods

Timed counter systems? (TPN, TA + counters, …)