FASTer acceleration of counter automata in practice

Sébastien Bardin, Alain Finkel, and Jérôme Leroux

LSV, CNRS UMR 8643
ENS de Cachan
61 avenue du président Wilson
F-94235 CACHAN Cedex
FRANCE
{bardin,finkel,leroux}@lsv.ens-cachan.fr

Abstract. We compute reachability sets of counter automata. Even if the reachability set is not necessarily recursive, we use symbolic representation and acceleration to increase convergence. For functions defined by translations over a polyhedral domain, we give a new acceleration algorithm which is polynomial in the size of the function and exponential in its dimension, while the more generic algorithm is exponential in both the size of the function and its dimension. This algorithm has been implemented in the tool FAST. We apply it to a complex industrial protocol, the TTP membership algorithm. This protocol has been widely studied. For the first time, the protocol is automatically proved to be correct for 1 fault and N stations, and using abstraction we prove the correctness for 2 faults and N stations also.

Keywords: acceleration, counter automata, reachability set, convex translation, TTP protocol.

1 Introduction

Context. Many real systems are infinite, because of parameters or unbounded data structures, such as counters, queues or stacks. We focus here on systems modelled by *counter systems*, i.e. automata extended with unbounded integer variables. Counter systems are a valuable abstraction since they allow to model a large range of complex systems, from communication protocols to multithreaded JAVA programs [Del]. Moreover many well-known models, reset/transfert Petri nets [DFS98], Broadcast protocols [EFM99] are subcases of counter systems. In general, reachability properties are undecidable for counter systems.

Counter systems. Let us recall that Presburger arithmetics is the first order additive theory $< \mathbb{N}^m, \leq, + >$. This theory is decidable, and Presburger sets (i.e. sets defined by a Presburger formula) can be represented symbolically by means of automata [BC96,WB00,Ler03b]. This representation is closed under common operations and both emptiness and inclusion are decidable. Moreover, the successor and predecessor of a Presburger set by an affine function are still Presburger sets. Thus the automata representation provides an efficient way to perform model-checking (or at least to check safety properties) on counter systems, building the reachability set of the system and testing for the inclusion with Presburger sets representing the property to verify. Several symbolic model-checkers use this framework [ALV,LAS,FAS].

State of the art. However, for a general counter system, the reachability set is not necessarily a Presburger set nor a recursive set. The classical fixpoint computation algorithm, consisting in firing one by one the transitions of the system until all the reachable states have been computed, may not terminate. A solution to help convergence is to use *acceleration*. Acceleration allows to compute in one step the exact effect of iterating an arbitrary number of times a control loop of the counter system. Acceleration is also called *meta-transition* [BW94] or *exact widening* in abstract interpretation. In [BW94,Boi98], it is proved that, given a convex polyhedral set S, under some algebraic conditions on the affine function σ , $post^*_\sigma(S)$ and $pre^*_\sigma(S)$ are also computable Presburger sets. Actually, we can even compute the transition relation as a Presburger formula. In [FL02], this result is extended to all Presburger sets.

The problem comes down to finding the good cycles whose accelerations will lead to the reachability set computation. An important step is the *reduction result* given in [FL02], which allows to replace a set C of cycles of length k ($|C|$ is exponential in k) by another equivalent set of cycles C', such that $|C'|$ is polynomial in k.

Tools for acceleration. To the best of our knowledge, three symbolic model-checkers (Lash,Fast,TReX) with acceleration are available for counter systems. Lash and Fast use the automata representation for Presburger sets. Lash is limited to convex guards and the user has to provide cycles to accelerate. Fast has full Presburger guards and can find automatically cycles to acccelerate. TReX [TRE] is designed to verify timed counter systems. TReX uses Constrained Parametric DBMs [AAB00] instead of automata to represent symbolic sets. Untimed models of TReX are restrictive counter systems, because guards and actions are strictly included in those of Fast and Lash. In the other hand, dense clocks are allowed and in particular cases, non linear sets may be computed.

Our results. In this paper, we focus on applying effectively acceleration techniques on counter systems. We investigate the specific case of *counter automata*, where functions are translations over convex polyhedral domains.

1. We give a 3-EXPTIME bound in the size of the domain for the generic Presburger acceleration. For counter automata, we give a simpler acceleration formula (*polyhedral acceleration*), which is proved to be *at most quadratic in the size of the domain.* [BW94,Boi98] also investigate functions with convex

domains, but they are not restricted to translations. Thus their acceleration cannot be expressed so easily than ours, and even if the resulting automata are the same, the intermediate computations are likely to be smaller with our formula. No complexity result is given in [BW94,Boi98].
2. The polyhedral acceleration is implemented in the tool FAST and applied to a non-trivial case study, the membership algorithm of the TTP protocol. This protocol was proved manually to be correct for k faults and N stations in [BM02] and a non-automatic verification with LASH and ALV is performed for 1 fault and N stations. In this paper, *the protocol is verified fully automatically for 1 fault and N stations, and using abstraction it is verified for 2 faults and N stations.*

Outline. Section 2 gives basic definitions and an overview of the main results on acceleration for counter systems. Section 3 investigates the specific case of counter automata, and gives the polyhedral acceleration algorithm. Finally in section 4, FAST is used to verify the complex industrial TTP protocol.

2 Acceleration of counter systems

2.1 Counter systems

We are interested in accelerating transitions for counter systems whose transitions are guarded affine functions. Firstly we introduce Presburger arithmetics, which is the first order theory $< \mathbb{N}^m, \leq, + >$. Then we describe counter systems.

Definition 1 (Presburger logic). *Consider a finite set X of free variables x. The set of Presburger formulas ϕ over X is defined by the grammar:*

$t ::= 0|1|x|t-t|t+t$

$\phi ::= t \leq t|\neg\phi|\phi \vee \phi|\exists y; \phi|true$

Definition 2 (Presburger-linear function [FL02]). *A* Presburger-linear function *f over m counters is a tuple $f = (M, v, D)$ such that $\forall x \in D, f(x) = M.x + v$, with M a square matrix, v a vector and $D \subseteq \mathbb{N}^m$ a Presburger set called the guard of f.*

Definition 3 (Counter systems). *A* counter system *over m counters $\boldsymbol{L}$ is a tuple $\boldsymbol{L} = (\Sigma, f_\Sigma)$ where Σ is a finite alphabet of actions and $f_\Sigma = \{f_a; a \in \Sigma\}$ is a set of Presburger-linear functions over m counters.*

Remark 1. We can add a control structure to a counter system without changing the expressibility of the models, encoding control states as a variable.

Definition 4 (The monoid of a counter system). *We call the monoid of a counter system $\boldsymbol{L}$ the multiplicative monoid generated by the set of square matrices $\{M_a;\ a \in \Sigma\}$ of $\boldsymbol{L}$. When $\boldsymbol{L}$ is composed of a unique function $f(s) = M.s + v$, then this monoid is simply written $< M >$.*

Counter systems with a finite monoid have nice acceleration properties and appear to be well-spread in practice. For example all transfer/reset/inhibitors Petri Nets and all Broadcast protocols are counter systems with a finite monoid.

2.2 Unambiguous Binary Automata

The automata approach is very fruitful to represent Presburger sets. An integer is represented by its encoding into a basis r. Then a set of integers is represented by an automaton whose associated language is exactly this set [BC96]. Number Decision Diagrams (NDDs) [Boi98,WB00] are usually used to represent any Presburger set of $\mathbb{N}^m$. Unambiguous Binary Automata [Ler03b,Ler03a] (UBA) is a similar approach, but they are proved to be smaller than NDDs [Ler03a].

Let $|\mathcal{A}|$ be the number of states of the automaton $\mathcal{A}$. Recall these results useful to bound the size of the UBA $\mathcal{A}(X)$ when X is defined by a first order formula (these results are deduced from results on NDDs summarized in [BB02]):

- the UBA $\mathcal{A}(X)$ where $X = \{x \in \mathbb{N}^m;\ \sum_{i=1}^{m} \alpha_i.x_i \# c\}$ with $\alpha_i, c \in \mathbb{Z}$ and $\# \in \{\leq, \geq, =\}$ can be computed in time and space bounded by $m.(\sum_{i=1}^{m} |\alpha_i| + |c|) + 1$.
- the UBA $\mathcal{A}(X)$ where $X = \{x \in \mathbb{N}^m;\ \sum_{i=1}^{m} \alpha_i.x_i = c[k]\}$ with $\alpha_i, c, k \in \mathbb{Z}$ and $c[k]$ denotes c *modulo* k, can be computed in time and space bounded by $2.m.|k| + 1$.
- the UBA $\mathcal{A}(X \cap Y)$ can be computed in time and space bounded by $|\mathcal{A}(X)|.|\mathcal{A}(Y)|$.
- the UBA $\mathcal{A}(\Pi(X))$ where Π is a projection function (removing some components) can be computed in time and space bounded by $m.2^{|\mathcal{A}(X)|}$.
- the UBA $\mathcal{A}(\mathbb{N}^m \backslash X)$ can be computed in time and space bounded by $|\mathcal{A}(X)|$.

2.3 Main results on acceleration

Let f be a function, and S a set, we define the acceleration of f, denoted f^*, by $f^*(S) = \bigcup_{i \in \mathbb{N}} f^i(S)$. R_f^* is the relation associated with f^*. The results on acceleration are summarised in this section. We denote by $||v||_\infty$ the infinite norm of the vector v, and by F the function f with no guard.

Acceleration of a cycle. In [FL02] it is proved that for a Presburger-linear function $f = (M, v, D)$ with a finite monoid, the transition relation R_f^* can be computed as a Presburger formula, of the form

$$R_f^* = \{(x, x') | x \in D \wedge (\exists k \geq 0; x' = F^k(x) \wedge (\forall i; 0 \leq i < k, F^i(x) \in D))\} \quad (1)$$

This result extends the one of [Boi98] which is restricted to affine functions over convex guards. No complexity result is given in the literature for this construction. We propose an upper-bound of the complexity.

Proposition 1. *Let $f = (M, v, D)$ be a Presburger linear function with a finite monoid. An UBA that represents the relation R_f^* can be computed in 3-EXPTIME in $|\mathcal{A}(D)|$, $||v||_\infty$ and $||M||_\infty$ and 5-EXPTIME in m.*

Remark 2. Getting an exponential lower-bound is an open problem. However it seems that this bound can be reached in the worst case.

Remark 3. In practice m, $||v||_\infty$ and $||M||_\infty$ are small (≤ 100), while $|\mathcal{A}(D)|$ can be very large (from 100 up to several millions, see section 3).

Finding the good cycles. Acceleration allows to help the convergence of the reachability set computation. Now the problem is to find the sequence of *good cycles* to accelerate, i.e. cycles whose acceleration will lead to the reachability set computation. But for a finite counter system $\boldsymbol{L} = (\Sigma, f_\Sigma)$, the number of Presburger-linear functions in the set $C_k = \{f_\sigma;\ \sigma \in \Sigma^{\leq k}\}$ may be exponential in k. However, this exponential number of functions can be *reduced* to a set of Presburger-linear functions $[C_k]$ with a polynomial size in k [FL02].

2.4 The tool Fast and its heuristic

Fast [BFLP03] is a tool dedicated to the analysis of counter systems, using symbolic representation with automata and acceleration. To find the cycles to accelerate, Fast provides an automatic search, which is often sufficient (all the examples on the Fast Web page [FAS] have been verified fully automatically).

We present the heuristic used in Fast to compute the reachability set of an initial set S_0, given a finite counter system $\boldsymbol{L}$. The semi-algorithm we propose can be seen as an extension of the semi-algorithm presented in [Boi98]. The basic idea is to add cycles to the initial set of linear transitions, and to accelerate them. These cycles are called *meta-transitions.* In [Boi98] cycles to be accelerated are provided by the user. Here we want these cycles to be found automatically.

Our problem is divided into two separate steps: the computation of the interesting cycles from a set of transitions, and the search heuristic given a set of cycles. For *the cycles computation*, the main problem is *the potentially exponential number of cycles.* For *the search heuristic*, the problem is *the automata explosion problem.* Because there is no relationship between the size of the set which is represented and the size of the automaton which represents it, choosing a bad function can lead to a set whose representation by automata is very large, and thus, the subsequent operations will take too much time.

The cycle computation. We make the assumption that, *in pratical cases, good cycles have a small length.* So we do not try to consider all the cycles at once, but only all the cycles of length less or equal to a constant k. We compute the cycles in a *static and incremental way. Static* because the set of cycles we use is fixed during the search. *Incremental* because if the search fails (according to a *stop criterion*), the length of cycles is increased and a new search is done. To efficiently compute the cycles, we use the reduction result from section 2.

(0) $k \leftarrow 1$
(1) *Compute* C_k, *the reduced set of cycles of length* $\leq k$
(2) *Use the search algorithm with* S_0 *and* $L \cup C_k$
(3) *if a fixpoint* S *is found then return* S
else (*the stop criterion is met*) *do* $k \leftarrow k + 1$, *goto* (1)

The search heuristic. The main point is to overcome the automata explosion problem. For this purpose, we introduce in the classic fixpoint algorithm a *minimization* step where we only try to reduce the size of the automaton computed so far. Thus our heuristic can be seen as two nested greedy algorithms. The first one tries to build new states (*new states first*). Once it is done we enter the minimization step (*smaller automaton first*), where transitions are chosen if they lead to smaller automaton. When it is not possible anymore, we come back to the first part. The search finishes when a fixpoint is found or when the stop criterion is met. Moreover, we choose the transitions to be fired with *fairness* assumptions, in order to avoid choosing always the same.

$S \leftarrow S_0$
while there exists f *such that* $f^*(S)$ *reaches new states do*
  $S \leftarrow f^*(S)$
  while there exists f *such that* $|\mathcal{A}(f^*(S))| < |\mathcal{A}(S)|$ *do*
    $S \leftarrow f^*(S)$
  end while
end while
return S

The stop criterion. Building a good stop criterion is not easy. After lots of experiments, we distinguish two simple and relevant factors to know when to increase cycle length. The first factor is the *size of the automaton built.* When it becomes too large, computation cannot be managed anymore and so the semi-algorithm will certainly not terminate within reasonable time. The second factor is *the depth of the search.* After lots of experiments, it seems that when the heuristic finishes, it ends rather quickly. So if the search is going too deep, the cycle length is probably too small.

In FAST, the user can define the maximal depth and the maximal size of the automaton. In practice, it is not very difficult to use, because the default setup (no limit on the size, 100 for the maximal depth) is often sufficient.

3 Acceleration of counter automata

3.1 The generic acceleration may be too complex

The generic acceleration technique we use may be very expensive and lead to very large automata. In practice it works well, but there are few examples where FAST cannot compute an accelerated transition relation because the automata manipulated are too large.

For example, considering the transition in figure 1, when FAST tries to compute the acceleration of the transition relation, the size of the internal automata explodes and the tool stops. This is due to the fact that we use MONA [MON] as an automata library. In this library, the number of states of the automata is limited (to 2^{24}), and during the computation this limit is exceeded. This example is taken from the TTP protocol (see section 4). It is the only practical example we found which brings MONA, and thus FAST, out of its limits.

$$Cp_1 \geq N \wedge Cp_2 < N \wedge d_{11} < C_{11} \wedge$$
$$d_1 + d_{11} - dA_{11} - dF_{11} - dA_{10} + dF_{10} - d_0 - d_{10} - d_{00} + dA_{00} + dF_{00} \leq 0$$
$$\rightarrow \; dF' := dF + 1, Cp_1' := Cp_1 + 1, Cp_2' := Cp_2 + 1, dF_{11}' := dF_{11} + 1, C_{11}' := C_{11} + 1$$

Fig. 1. A transition which brings FAST out of its limits

3.2 A simpler acceleration for counter automata

We can notice that the example above belongs to a particular case: functions are translations over a (convex) polyhedral domain.

Definition 5 (convex translation). *A convex translation f is a Presburger-linear function $f = (I, v, D)$ where I is the identity matrix and D is a polyhedron.*

Definition 6 (counter automaton). *A counter automaton $\boldsymbol{L}$ is a counter system whose functions are all convex translations.*

In such a case we can use a simpler acceleration formula, because we do not need to test if all the successors are in the guard. As long as the first element and the last element are in the guard, the intermediate elements are in the guard as well. The transition relation can be computed as

$$R_f^* = \{(x, x') | x \in D \wedge (\exists k \geq 0; x' = F^k(x) \wedge k > 0 \Rightarrow F^{k-1}(x) \in D)\} \quad (2)$$

[BW94,Boi98] study functions over convex domains, but because these functions are not restricted to translations they cannot use the above argument. We will present in the following an acceleration formula based on ideas from (2) which is proved to be quadratic in the domain of the function.

Proposition 2. *Let $f = (I, v, D)$ be a convex translation. The accelerated transition relation of f is equal to:*

$$R_f^* = I \cup \{(x, x') \in D \times (D + v);\; x' - x \in \mathbb{N}.v\} \quad (3)$$

Proof. Let $R = \{(x, x') \in D \times (D + v);\; x - x' \in \mathbb{N}.v\}$. Consider $(x, x') \in R_f^*$ and let us prove that $(x, x') \in I \cup R$. There exists $n \geq 0$ such that $x' = f^n(x) = x + n.v$. If $n = 0$ then $(x, x') = (x, x) \in I \cup R$. Otherwise, we have $n \geq 1$. From $f^{n-1}(x) \in D$, we deduce $f^n(x) \in f(D) = D + v$. Therefore $(x, x') \in I \cup R$. Let us prove the converse by considering $(x, x') \in I \cup R$. Remark that if $(x, x') \in I$, then $(x, x') \in R_f^*$. So we can assume that $(x, x') \notin I$. In this case, $(x, x') \in D \times (D + v)$ and there exists $n \geq 1$ such that $x' = x + n.v$. As $x + n.v \in D + v$, we have $x + (n-1).v \in D$. As D is a convex set and x and $x + (n-1).v \in D$, for any $k \in \{0, \ldots, n-1\}$, we have $F^k(x) = x + k.v \in D$. Therefore $x' = f^n(x)$ and we have proved $(x, x') \in R_f^*$. □

Theorem 1. *Let $f = (I, v, D)$ be a convex translation. An UBA $\mathcal{A}(R_f^*)$ representing the relation R_f^* can be computed in time and space bounded by*

$$|\mathcal{A}(R_f^*)| \leq |\mathcal{A}(D)|^2.4.(4.m.\,||v||_\infty + 1)^{3.m}$$

Proof. Let us consider the relation $R = \{(x, x') \in \mathbb{N} \times \mathbb{N};\ x' - x \in \mathbb{N}.v\}$ and let $I_0 = \{i \in \{1, \dots, m\};\ v_i \neq 0\}$ and $I = \{i \in \{1, \dots, m\};\ v_i = 0\}$. Remark that if $I_0 = \emptyset$ then $R_f^* = I$ and in this case $|\mathcal{A}(R_f^*)| = 4$. So, we can assume that there exists an index $i_0 \in I_0$. We denote by ϕ the Presburger fomula $\phi := (x'_{i_0} - x_{i_0} \geq 0)$ if $v_{i_0} > 0$ and $\phi := (x'_{i_0} - x_{i_0} \leq 0)$ otherwise.

Let $a[b]$ denote the value a modulo b. We now prove that R is defined by the following Presburger formula:

$$\phi \bigwedge_{i \in I} (x'_i = x_i) \bigwedge_{i \in I_0 \setminus \{i_0\}} ((x'_i - x_i).v_{i_0} = (x'_{i_0} - x_{i_0}).v_i) \bigwedge_{i \in I_0} (x'_i - x_i = 0[v_i])$$

Let $(x, x') \in R$. There exists $n \geq 0$ such that $x' - x = n.v$. For every $i \in I$, we have $x'_i = x_i$. Moreover, for every $i \in I_0$, we have $x'_i - x_i = n.v_i$ and $x'_{i_0} - x_{i_0} = n.v_{i_0}$. Hence $(x'_i - x_i).v_{i_0} = (x'_{i_0} - x_{i_0}).v_i$ and $x'_i - x_i = 0[v_i]$. From $x'_{i_0} - x_{i_0} = n_{i_0}.v_{i_0}$, we deduce that ϕ is true. Let us prove the converse by considering a tuple (x, x') such that ϕ is true and for every $i \in I$, we have $x'_i = x_i$ and for every $i \in I_0$ we have $(x'_i - x_i).v_{i_0} = (x'_{i_0} - x_{i_0}).v_i$ and $x'_i - x_i = 0[v_i]$. As $x'_i - x_i = 0[v_i]$, there exists $n_i \in \mathbb{Z}$ such that $x'_i - x_i = n_i.v_i$. From the equality $(x'_i - x_i).v_{i_0} = (x'_{i_0} - x_{i_0}).v_i$, we deduce $n_i.v_i.v_{i_0} = n_{i_0}.v_{i_0}.v_i$. As $v_i.v_{i_0} \neq 0$, we have $n_i = n_{i_0}$ for every $i \in I_0$. In particular, we have $x' = x + n_{i_0}.v$. As ϕ is true, we deduce $n_{i_0} \geq 0$ from $x'_{i_0} - x_{i_0} = n_{i_0}.v_{i_0}$. So $(x, x') \in R$.

An UBA that represents ϕ or $(x'_i = x_i)$ can be computed in time and space bounded by $2.m+1$. We can also compute an UBA that represents $(x'_i - x_i).v_{i_0} = (x'_{i_0} - x_{i_0}).v_i$ in time and space bounded by $m.(2.|v_{i_0}| + 2.|v_i|) + 1 \leq 4.m.\,||v||_\infty + 1$. Moreover, we can compute an UBA that represents $x'_i - x_i = 0[v_i]$ in time and space bounded by $2.m.|v_i| + 1 \leq 2.m.\,||v||_\infty + 1$. Therefore, we can compute an UBA that represents R in time and space bounded by $(4.m.\,||v||_\infty + 1)^{2.m}$.

From the equality $R_f^* = I \cup ((D \times (D + v)) \cap R)$, we deduce that R_f^* is computable in time and space bounded by $|\mathcal{A}(I)|.|\mathcal{A}(D)|.|\mathcal{A}(D+v)|.|\mathcal{A}(R)|$. From [BB03], we deduce that an UBA that represents $D + v$ can be computed in time and space bounded by $|\mathcal{A}(D)|.(m.\,||v||_\infty + 1)^m$. Moreover, recall that $|\mathcal{A}(I)| = 4$.

We have proved that R_f^* can be computed in time and space bounded by:

$$\begin{aligned} &|\mathcal{A}(D)|^2.4.(4.m.\,||v||_\infty + 1)^{2.m}.(m.\,||v||_\infty + 1)^m \\ &\leq |\mathcal{A}(D)|^2.4.(4.m.\,||v||_\infty + 1)^{3.m} \end{aligned}$$

□

Remark 4. For the polyhedral acceleration, the complexity is quadratic in the size of the automaton representing the guard D, polynomial in $||v||_\infty$ and exponential in the number of counters m. This is a major improvement compared to the generic case (section 2), where the complexity is 3-EXPTIME in the size of the automaton representing the guard D, 3-EXPTIME in $||v||_\infty$ and 5-EXPTIME in m. Note that the resulting automaton is the same (the representation is canonical), the difference is on the intermediate automata.

Results. With this simpler acceleration formula, the acceleration relation of the transition of figure 1 can be computed, avoiding the automata explosion problem. The computation takes 18 seconds, 260 Mbytes (!). The resulting automaton has 413,447 states. For comparison, automata representing accelerations of transitions in [FAS] have roughly 300 states.

4 FAST in practice: the TTP/C

This section describes the verification of the TTP/C protocol with FAST. In previous work, the protocol has been semi-automatically verified by the tools ALV and LASH for N stations (microprocessors) and 1 fault, but the tools fail for 2 faults. Here FAST verifies the protocol fully automatically for N stations and 1 fault, and using abstraction the protocol is verified for N stations and 2 faults.

4.1 Presentation of the TTP protocol

The TTP [KG94] is used in car industry to manage embedded microprocessors. We focus here on the *group membership algorithm* of the TTP. It is a *fault-tolerant algorithm*. It ensures that if a fault occurs, after a certain amount of time the embedded microprocessors which are still valid keep on communicating with each other, without any partition among the microprocessors. We were interested in verifying such a protocol since it is a complex industrial case study which needs a very expressive model, with two causes of infinity: the number of stations and the number of faults.

The time is divided into rounds. Each round is divided into as many slots of communication as the number of stations (microprocessors). Each station s_i has a *membership list* stating which stations s_j are considered as invalid. During a slot, only one station sends a message, the others are listening. A sender sends its membership list to all listeners. A listener which receives a list different from its own list considers the message as invalid (and updates its own list). When a station receives in a round more invalid messages than valid ones, *the station considers itself as invalid*. It becomes inactive (it listens but does not send anymore). The goal of the membership algorithm is to ensure that after a certain amount of time following a fault, the system reaches a configuration where the active stations are all corresponding with each other, i.e. they have all the same membership list. For a more complete description, one can refer to [KG94,BM02].

4.2 Previous non-automatic verification of the TTP

There have been lots of studies on the TTP protocol in general, and on its membership algorithm in particular. We start from the work of Merceron and Bouajjani. In [BM02] they propose a family of counter systems abstractions, depending on the number of faults considered. They prove manually that the algorithm is valid for any number of stations N and any number of faults k. Actually they prove more: the algorithm stabilizes within two rounds after the k-ieth fault occurs.

They also try to prove automatically with different tools (LASH and ALV) the correctness of the protocol. For 1 fault and N stations, they verify it in a "user-guided way": they divide the protocol in two submodules. They compute the reachability set of the first submodule and prove automatically a nice property on it (true only for 1 fault). Then they use this property to simplify the computation of the second submodule. For N stations and more than 1 fault, the computation does not terminate.

4.3 Automatic verification for 1 fault and N stations

The model. The abstraction we use was proposed by Merceron and Bouajjani in [BM02]. The corresponding counter system is given in figure 2. N is the number of stations. C_W (resp. C_F) is the number of working (resp. faulty) stations. A fault splits stations into two cliques C_1 and C_0 of stations which only communicate within the same clique. C_p is the number of elapsed slots in the round. It is reseted to 0 when $C_p = N$ (and a new round begins). Variable d (resp. d_0, d_1, d_F) is the number of working stations (resp. in clique 0, in clique 1, faulty) which have emitted a message during the round. The control node `normal` represents the normal behaviour of the system. When a failure occurs, the system moves into the control node `Round1`, and then this round is finished, the system moves into the control node `later`. The property to check is that, two rounds after the failure occurs, valid stations are all communicating with each other, which is expressed by:

$$(P_1)\ state = later \wedge C_p = N \Rightarrow (C_1 = 0 \vee C_0 = 0)$$

The translation of this abstraction in counter systems is not direct, because of the nondeterministic affectations from control node `normal` to control node `round1`. The transition from `later` to `normal` indicates that the protocol comes back to the normal behaviour, and that another failure can occur. Fortunately, it is not relevant for our property because we are interested in what happens in control node `later`. Thus we can remove this transition. Then the nondeterministic affectation will happen only once, hence it will be encoded in the initial configuration.

Results. FAST checks *fully automatically* that property P_1 is verified. Tests have been performed on an Intel Pentium 4 at 2.4 GHz with 1 Gbyte of RAM. Computing the reachability set only requires cycles of length 1. It takes 940 seconds and 73 Mbytes. The reachability set has 27,932 states.

4.4 Abstraction & automatic verification for 2 faults and N stations

The model. The abstraction proposed in [BM02] for 2 faults is converted into a counter system as in the 1-fault case. The counter system is presented in figure 3.

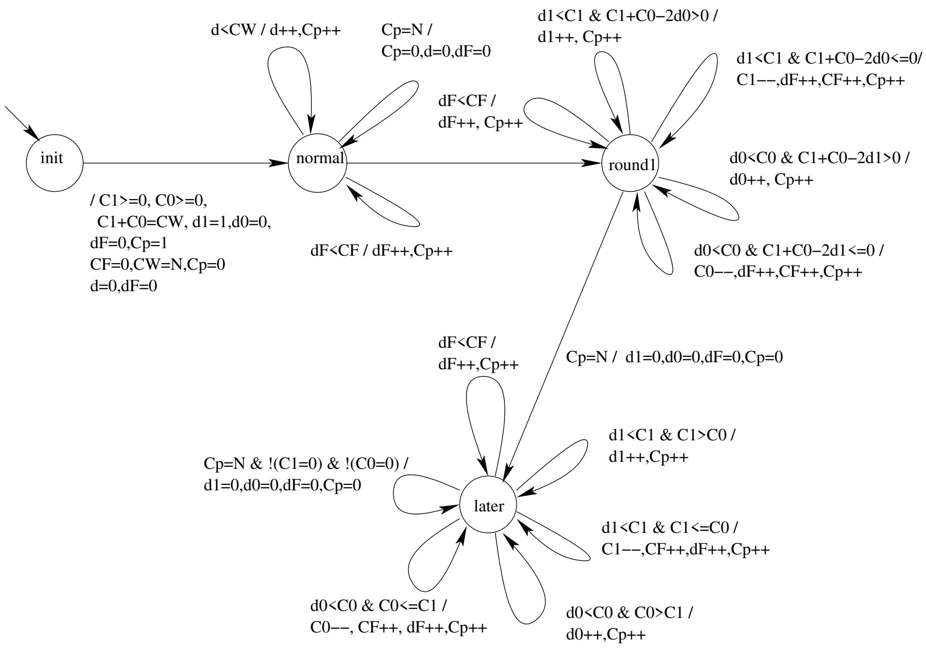

Fig. 2. Counter system for the TTP/C protocol, 1 fault and N stations

Differences with the 1-fault case are mainly because there are now three different cliques. Moreover the behaviour during the end of $Round_0$, the round where the first failure occurs (this round starts with the first failure) has to be separated from the round $Round_1$ starting with the second failure. The first failure splits stations in cliques C_1 and C_0, then the second failure occurs in C_1 which is split into C_{11} and C_{10}. C_0 becomes C_{00} and C_{01} does not exists. Variables d_0 and d_1 are the number of stations of C_0 and C_1 which have emitted after the first failure and before the second. C_{p1} (resp. C_{p2}) is the number of elapsed slots since the first failure (resp. second failure) occured. Variable d_{00} (resp. d_{10}, d_{11}, d_F) is the number of stations in clique C_{00} (resp. in clique C_{10}, in clique $_{C11}$, faulty stations) which have emitted a message during the round following the second failure. Variable dA_{00} (resp. dA_{11}, dA_{10}) is the number of stations in clique C_{00} (resp. C_{11}, C_{10}) which have emitted after the end of $Round_0$ and before the end of $Round_1$. Variable dF_{00} (resp. dF_{11}, dF_{10}) is the number of faulty stations in clique C_{00} (resp. C_{11}, C_{10}) whose time slot is elapsed after the end of $Round_0$ and before the end of $Round_1$. The property to check is still the absence of clique, which is expressed by P_2 as follows:

$$(P_2)\ state = later \wedge C_{p2} = N \Rightarrow ((C_{11} \neq 0 \wedge C_{10} = 0 \wedge C_{00} = 0) \vee$$
$$(C_{11} = 0 \wedge C_{10} \neq 0 \wedge C_{00} = 0) \vee (C_{11} = 0 \wedge C_{10} = 0 \wedge C_{00} \neq 0))$$

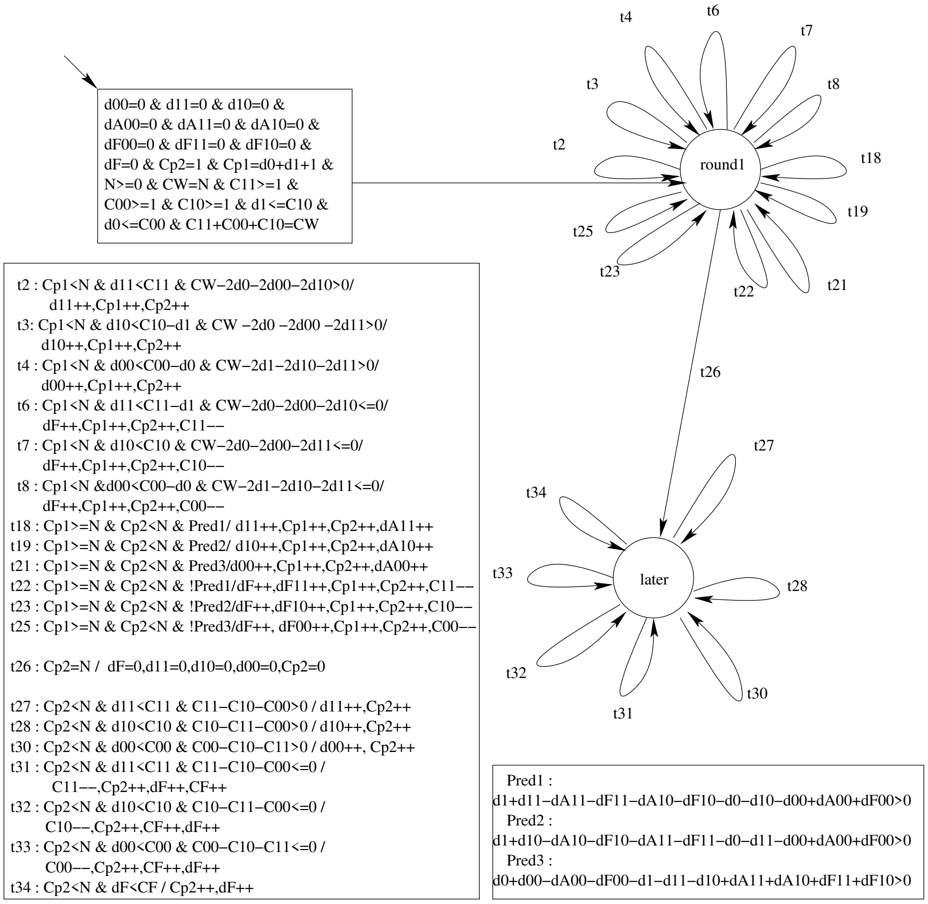

Fig. 3. Counter system for the TTP/C protocol, 2 faults and N stations

Presburger acceleration fails. When accelerating the transitions, the size of the internal automata manipulated by FAST explodes and the tool stops (see section 3). It highlights the complexity of this protocol.

Using polyhedral acceleration for a small number of stations. All the transitions except t_{26} are convex translations. Moreover t_{26} does not need to be accelerated because it is not a loop. Hence polyhedral acceleration can be used instead of the Presburger acceleration technique. FAST manages to compute accelerations of the transitions. Unfortunately, with an arbitrary number of stations, the MONA size limit is exceeded during the reachability set computation and FAST stops. However, when fixing the value of N to 5, the reachability set is computed fully automatically and the property P_2 is verified. For N=5, it takes 446 seconds and 588 Mbytes. The reachability set has 5,684 states.

Using abstraction for an arbitrary number of stations. We were unable to compute the whole reachability set, but we can still compute an *over-approximation* of the reachability set and check that P_2 is valid.

Firstly we try to simplify some guards on the finite case. When the over-approximation is sufficient to prove P_2, we use it for the infinite case. Thus guards of $t_2, t_3, t_4, t_6, t_7, t_8, t_{18}, t_{19}, t_{21}, t_{22}, t_{23}, t_{25}$ are simplified. Then we try to *reduce the number of variables*. We use simple invariants such as $C_W = C_{11} + C_{10} + C_{00}$ and $C_{p1} = C_{p2} + d_0 + d_1$ to remove variables C_W and C_{p1}. Moreover, d_F is useless in the first part of the algorithm, we need it only in `later`, and its value is then $N - C_{11} - C_{00} - C_{10}$. So d_F is removed. Finally after simplifications in guards, some variables are not used anymore. Hence, d_0 and d_1 are removed, and dA_{11}, dA_{10} and dA_{00} are removed from the first part. This way, FAST computation terminates. The clique avoidance algorithm is valid for 2 faults and N stations.

Results. Presburger acceleration does not terminate. With *polyhedral acceleration*, FAST checks *fully automatically* that property P_2 is verified *for a small number of stations*. For N=5 it takes 446 seconds and 588 Mbytes. The reachability set has 5,684 states. For *an arbitrary value of N*, the internal representation explodes when computing the reachability set. We have to use an *abstraction*. FAST checks that property P_2 is verified *for an arbitrary number N of stations*. It takes 175 seconds and 210 Mbytes with the polyhedral acceleration. These results are summarized in table 1. The symbol $\uparrow$ indicates that FAST limits are exceeded, hence intermediate automata have more than 2^{24} states. We can notice that polyhedral acceleration works better than Presburger acceleration, in both space and time, except for the last case (the abstraction). Here, functions are simple so the maximal amount of memory used represents the size of intermediate automata representing the reachability set, and not the size of the acceleration relations like in the other examples.

5 Conclusion and Future Work

The polyhedral acceleration appears to be very interesting since it allows to compute acceleration relations for which the Presburger acceleration takes too much memory. We can probably define other acceleration algorithms, more restrictive than Presburger acceleration but more efficient. Another direction is to find a generic acceleration more efficient than the one described here, using smart intermediate computations. Finally, the TTP protocol is a really challenging case-study. Even when the reachability set is computed by FAST, we are never far from the limits of the tool. More efficient Presburger automata libraries, using for example cache systems or modular computation, are necessary to scale up acceleration to wider systems.

	Presburger acceleration		polyhedral acceleration		
	time1 seconds	memory1 Mbytes	time2 seconds	memory2 Mbytes	number of states
1 fault, N stations	940	73	600	63	27,932
2 faults, 5 stations	↑	↑	446	588	5,684
2 faults, 10 stations	↑	↑	12,365	588	273,427
2 faults, 15 stations	↑	↑	↑	↑	↑
2 faults, N stations	↑	↑	↑	↑	↑
2 faults, N stations (abstraction)	210	200	175	200	11,036

Table 1. Benchmark for the verification of the TTP/C with FAST

References

[AAB00] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *Proc. 12th Int. Conf. Computer Aided Verification (CAV'2000), Chicago, IL, USA, July 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 419–434. Springer, 2000.

[ALV] ALV *homepage*. `http://www.cs.ucsb.edu/~bultan/composite/`.

[BB02] C. Bartzis and T. Bultan. Efficient symbolic representations for arithmetic constraints in verification. Technical Report ucsb cs:TR-2002-16, University of California, Santa Barbara, Computer Science, 2002.

[BB03] C. Bartzis and T. Bultan. Efficient image computation in infinite state model checking. In *Proc. 15th Int. Conf. Computer Aided Verification (CAV'2003), Boulder, CO, USA, July 2003*, volume 2725 of *Lecture Notes in Computer Science*, pages 249–261. Springer, 2003.

[BC96] A. Boudet and H. Comon. Diophantine equations, Presburger arithmetic and finite automata. In H. Kirchner, editor, *Proc. Coll. on Trees in Algebra and Programming (CAAP'96)*, volume 1059 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 1996.

[BFLP03] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. FAST: Fast Acceleration of Symbolic Transition systems. In *Proc. 15th Int. Conf. Computer Aided Verification (CAV'2003), Boulder, CO, USA, July 2003*, volume 2725 of *Lecture Notes in Computer Science*, pages 118–121. Springer, 2003.

[BM02] A. Bouajjani and A. Merceron. Parametric verification of a group membership algorithm. In *Proc. 7th Int. Symp. Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'2002), Oldenburg, Germany, Sep. 2002*, volume 2469 of *Lecture Notes in Computer Science*, pages 311–330. Springer, 2002.

[Boi98] B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces.* PhD thesis, Université de Liège, 1998.

[BW94] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *Proc. 6th Int. Conf. Computer Aided Verification (CAV'94), Stanford, CA, USA, June 1994*, volume 2725 of *Lecture Notes in Computer Science*, pages 55–67. Springer, 1994.

[Del] G. Delzanno. *Home Page – Giorgio Delzanno.* `http://www.disi.unige.it/person/DelzannoG/`.

[DFS98] C. Dufourd, A. Finkel, and P. Schnoebelen. Reset nets between decidability and undecidability. In *Proc. 25th Int. Coll. Automata, Languages, and Programming (ICALP'98), Aalborg, Denmark, July 1998*, volume 1443 of *Lecture Notes in Computer Science*, pages 103–115. Springer, 1998.

[EFM99] J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *Proc. 14th IEEE Symp. Logic in Computer Science (LICS'99), Trento, Italy, July 1999*, pages 352–359. IEEE Comp. Soc. Press, 1999.

[FAS] Fast *homepage.* `http://www.lsv.ens-cachan.fr/fast/`.

[FL02] A. Finkel and J. Leroux. How to compose Presburger-accelerations: Applications to broadcast protocols. In *Proc. 22nd Conf. Found. of Software Technology and Theor. Comp. Sci. (FST&TCS'2002), Kanpur, India, Dec. 2002*, volume 2556 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2002.

[KG94] H. Kopetz and G. Grünsteidl. A time trigerred protocol for fault-tolerant real-time systems. In *IEEE computer*, volume January, pages 14–23, 1994.

[LAS] lash *homepage.* `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`.

[Ler03a] J. Leroux. *Algorithmique de la vérification des systèmes à compteurs. Approximation et accélération. Implémentation de l'outil* Fast. PhD thesis, École Normale Supérieure de Cachan, 12^{th} december 2003.

[Ler03b] J. Leroux. The affine hull of a binary automaton is computable in polynomial time. In *Proc. 5th Int. Workshop on Verification of Infinite State Systems (INFINITY'2003), Marseille, France, Sep. 2003*, Electronic Notes in Theor. Comp. Sci. Elsevier Science, 2003.

[MON] *The* MONA *project.* `http://www.brics.dk/mona/`.

[TRE] TReX *homepage.* `http://www.liafa.jussieu.fr/~sighirea/trex/`.

[WB00] P. Wolper and B. Boigelot. On the construction of automata from linear arithmetic constraints. In *Proc. 6th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2000), Berlin, Germany, Mar.-Apr. 2000*, volume 1785 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2000.