 Electronic Notes in Theoretical Computer Science 98 (2004) 89–104 

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

The Affine Hull of a Binary Automaton is Computable in Polynomial Time

# J´erˆome Leroux[1](#_bookmark0)

*Laboratoire Sp´ecification et V´erification, CNRS UMR 8643 & ENS de Cachan, 61 av. du Pr´esident Wilson,*

*94235 Cachan cedex, France*

Abstract

We present the class of binary automaton, a new representation for the subsets of N*m* that naturally extends the NDD ([[25](#_bookmark32)], [[10](#_bookmark17)]). We prove that the affine hull of the set of vectors represented by a binary automaton is computable in polynomial time. As application, we show that the set of place invariants [[11](#_bookmark19)] of a counter system (an extension of the Broadcast Protocols [[16](#_bookmark23)], [[13](#_bookmark21)], [[12](#_bookmark20)], the Reset/Transfer Petri Nets [[15](#_bookmark24)],[[11](#_bookmark19)] and the linear systems [[18](#_bookmark25)]), is computable in polynomial time.

*Keywords:* Presburger, affine hull, NDD, binary automaton, place invariant.

# Introduction.

An infinite state system is by definition a system whose the reachability set may contain an infinite number of configurations. For such a system it is therefore impossible to represent this set just by enumerating its elements. To overcome this problem, symbolic model checkers implements symbolic rep-

resentations adapted to the structure of the infinite sets represented. For instance the tool TreX [[24](#_bookmark33)], [[4](#_bookmark12)] implements SRE [[1](#_bookmark9)] for representing infinite set of words closed under the sub-word relation [[2](#_bookmark10)], the tool Babylon [[5](#_bookmark13)], [[3](#_bookmark11)] uses CST [[14](#_bookmark22)] for representing upward closed sets [[19](#_bookmark26)], and so on.

In this article, we are interested in the NDD [[25](#_bookmark32)] [[26](#_bookmark34)] (a.k.a DFA [[10](#_bookmark17)], [[23](#_bookmark30)], [[21](#_bookmark28)], [[20](#_bookmark29)]), the symbolic representation used by the symbolic model checkers

1 Email: [leroux@lsv.ens-cachan.fr](mailto:leroux@lsv.ens-cachan.fr)

1571-0661 © 2004 Published by Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

doi:10.1016/j.entcs.2003.10.007

Fast [[6](#_bookmark14)], [[17](#_bookmark27)] and Lash [[22](#_bookmark31)] to represent Presburger definable subsets of N*m*

by a finite automaton.

These tools use NDD at different levels of their implementation:

* To represent the input system. In fact, these tools analyze counter systems such that each action is represented by a NDD that represents the set of configurations which the action can be fired from (Lash requires that this set is convex [[18](#_bookmark25)]) and by an affine function that relates the value of the

counters before and after the action is fired.

* To represent the property that the input system must ensure. This property is in fact a NDD that represents the set of bad configurations that the system must avoid.
* To accelerate transitions ([[18](#_bookmark25)],[[9](#_bookmark18)],[[8](#_bookmark16)]). Under some algebraic conditions (of- ten met [[18](#_bookmark25)], [[6](#_bookmark14)]), a NDD that represents the transitive closure of a sequence of actions can be computed.
* To represent subsets of the reachability set.

In these tools, no structural analysis of the counter systems is done to simplify the computation of the reachability set. However, in the case of Petri nets [[7](#_bookmark15)], or more generally self/modifying Petri Nets [[11](#_bookmark19)], the place invariants is a useful tool:

* for reducing the number of true counters of the system [[14](#_bookmark22)] (recall that in practice, we are limited in the tools Fast and Lash by a dozen of counters).
* for helping the termination of the reachability set. In fact, we can compute an over-approximation of the configurations which a bad configuration can be reached from ([[14](#_bookmark22)]) and intersect the computation of the reachability set with this over-approximation.

We proved in this article an interesting result: place invariants can be extended to the class of counters systems used in Fast and Lash and remain computable in polynomial time. To obtain this result we prove an other result: the affine hull of the set of vectors accepted by a NDD is computable

in polynomial time.

*Our contributions*

* We introduce the class of binary automaton, a natural extension of the NDD. Whereas NDD corresponds to a word by word representation, we show that the class of binary automata is a bit by bit representation.
* We prove that the affine hull of the set of vectors represented by a binary automaton is computable in polynomial time.
* We introduce the class of counter systems, an extension of the class of Broadcast protocols, Reset/Transfer Petri Nets and the linear systems.
* We extend the definition of place invariants to this class and proved that they remain computable in polynomial time.
* We show the link between place invariants and the affine hull of the reach- ability relation.

*Plan of the paper*

In section 2, some usual notations are recalled and in section 3 we explain why all the operations used in this article over the affine spaces can be done just by using a Gauss elimination algorithm. In section 4, the binary automata are presented and in the following section, an algorithm for computing the affine hull of the set of vectors accepted by a binary automaton is provided. Finally, in section 6, this affine hull is proved to be useful for computing the place invariants of an effective counter system, a model introduced in this last section.

# Notations.

The cardinal of a finite set *X* is written card(*X*).

The set of rationals, integers and positive integers are respectively written Q, Z and N. The set of vectors with *m* components in a set *X* is written *Xm*. The *i*-th component of a vector *x* ∈ *Xm* is written *xi* ∈ *X*; we have *x* = (*x*1*,..., xm*). For any vector *v, v*' ∈ Q*m* and for any *t* ∈ Q, we define *t.v* and *v* + *v*' in Q*m* by (*t.v*)*i* = *t.vi* and (*v* + *v*')*i* = *vi* + *v*'.

*i*

A vector subspace *V* of Q*m* is a non empty subset *V* ⊆ Q*m* such that for every *v, v*' ∈ *V* and for every *t, t*' ∈ Q, we have *t.v* + *t*'*v*' ∈ *V* . An affine subspace *A* of Q*m* is a subset of Q*m* (eventually empty) such that for every *a, a*' ∈ *A* and for every *t, t*' ∈ Q such that *t* + *t*' = 1, we have *t.a* + *t*'*.a*' ∈ *A*. As any intersection of affine spaces remains an affine space, the affine hull of a subset *X* of Q*m* is well defined as the least affine space that contains *X*;

this affine space is written aff(*X*). Recall that for any set *X*, there exists

a finite subset *X*' ⊆ *X* such that aff(*X*) = aff(*X*'). In particular for any affine space *A*, there exists a finite subset *X*' ⊆ *A* such that *A* = aff(*X*'). Recall that the dimension of an affine space *A* is the minimal integer written dim(*A*) ∈ {−1*,... , m*} such that there exists a finite subset *X* ⊆ *A* satisfying aff(*X*) = *A* and card(*X*) − 1 = dim(*A*). Such a subset *X* is called a basis of *A*.

The set of square matrices over Q is written M*m*(Q). A function *f* : Q*m* →

Q*m* is said affine if these exists a square matrix *M* ∈ N*n*(Q) and a vector *v* ∈ Q*m* such that *f* (*x*) = *M.x* + *v* for any *x* ∈ Q*m*. A function *f* : Q*m* → Q is said linear if there exists a vector *v* ∈ Q*m* such that for every *x* ∈ Q*m*, we

have *f* (*x*) = Σ

*m*

*i*=1

isomorphic to Q*m*.

*vi.xi*. The set of linear functions is therefore a vector space

The set of words over a finite alphabet Σ is written Σ∗. The concatenation of two words *σ* and *σ*' in Σ∗ is written *σ.σ*'. The empty word in Σ∗ is written *ϵ*.

A finite automaton ffi is a tuple ffi = (*Q,* Σ*,* ∆*, Q*0*,F* ); *Q* is the finite set of states, Σ is the finite alphabet, ∆ ⊆ *Q* × Σ × *Q* is the transition relation, *Q*0 ⊆ *Q* is the set of initial states and *F* ⊆ *Q* is the set of final states. A finite automaton ffi is said complete and deterministic if the set *Q*0 is reduced to one element *Q*0 = {*q*0} and if there exists a function *δ* : *Q* ×Σ → *Q* such that ∆ =

{(*q, δ*(*q, a*)); *q* ∈ *Q*; *a* ∈ Σ}. A path *P* in a finite automaton ffi from a state *q* to a state *q*' is a finite sequence *q* = *q*0*,* (*q*0*, a*1*, q*1)*, q*1*,... ,* (*qn*−1*, an, qn*)*, qn* = *q*' with *n* ≥ 0 such that (*qi*−1*, ai, qi*) is a transition in ∆. The label of *P* is the

*σ*

word *σ* = *a*1 *... an* ∈ Σ∗. Such a path is written *q* −→ *q*'.

A binary relation R over a set *X* is a subset of *X* ×*X*; If a tuple (*x, x*') ∈ R, we write *x*R*x*'. The identity relation over *X* is written *IX* or just *I* and it is defined by *xIx*' iff *x* = *x*'.

# About the complexity of the affine spaces.

In this article, we often consider the affine hull of the union of two affine spaces, the image of an affine space by an affine function and we often test the inclusion of two affine spaces. We study briefly the complexity of these operations by representing an affine space *A* by a finite set *X* of at most *m* +1 vectors in *A* such that *A* = aff(*X*).

First remark that if *A* and *A*' are two affine spaces respectively represented by *X* and *X*', then by using a Gauss elimination, we can compute in poly- nomial time a finite subset *P* of at most *m* + 1 vectors in *X* ∪ *X*' such that aff(*A* ∪ *A*') = aff(*P* ).

Moreover, the affine space *f* (*A*) defined as the image of an affine space *A* = aff(*X*) by an affine function *f* (*x*) = *M.x* + *v* where *M* ∈ N*m*(Q) and *v* ∈ Q*m* is represented by the set *f* (*X*).

Finally, remark that by using a Gauss elimination, for any two sets *X* and *X*' of at most *m* + 1 vectors in Q*m*, we can check in polynomial time if aff(*X*) ⊆ aff(*X*').

# Binary automata.

Binary automata are proved to be a natural extension of NDD by providing a bit by bit representation of vectors of N*m* whereas NDD is rather a word by word representation.

Recall that a NDD is a deterministic and complete automaton over the alphabet Σ*r* = {0*,...,r* − 1} where *r* ≥ 2 is called the base of decomposition such that the length of any word accepted by the automaton can be divided by *m*. This representation is really well adapted for representing subsets of N because any *x* ∈ N can be represented as a finite sequence of “bits” in Σ*r*.

A vector *x* ∈ N*m* is also easily represented by a word *b*1*b*2 *... bn.m* such that

each *xi* is represented by *bibi*+*m ... bi*+(*n*−1)*.m*. With such a representation, we remark that the vectors *x*' represented by *b*1*b*2 *... bn.m* is related to the vectors *x* represented by *bm*+1*bm*+1 *... bn.m*, by the following formula:

*x*' = *r.x* + (*b*1*,..., bm*)

For this reason, we claim that NDD is a word by word representation that needs to read *m*-bits by *m*-bits to compute the vector represented. This re- striction seems to be in contradiction with a polynomial time computation of the affine hull of the set of vectors represented by a binary automaton due to the exponential size of card(Σ*r*)*m*.

To overcome this limitation, we propose to associate to any word *σ* ∈ Σ∗

*r*

a vector in N*m* without any restriction on the length of *σ*.

Let us introduce the affine function *λσ* : Q*m* → Q*m* defined for any word

*σ* ∈ Σ∗ and for any bit *b* ∈ Σ*r* by the following induction:

*r*

 *λb*(*x*1*,... , xm*) = (*x*2*,... , xm, r.x*1 + *b*)

 *λb.σ* = *λb* ◦ *λσ*

Definition 4.1 A binary representation of a vector *x* ∈ N*m* is a word *σ* ∈ Σ∗

*r*

such that *x* = *ρ*(*σ*) where *ρ*(*σ*) = *λσ*(0*,... ,* 0).

Definition 4.2 A binary automaton ffi is a finite automaton over the alpha- bet Σ*r*. The set of vectors represented by a binary automaton ffi is the subset *ρ*(L(ffi)) of N*m*.

We can now give the definition of NDD with our notation that is equivalent to the original definition just because for any word *b*1 *... bm* and for any *x* ∈ Q*m*, we have *λb*1*...bm* (*x*) = *r.x* + (*b*1*,..., bm*).

Definition 4.3 [[[9](#_bookmark18)]] A NDD is a deterministic and complete binary automa- ton such that *m* divides the length of any accepted words.

Remark 4.4 NDD also allow to represent vectors in Z*m* by considering a 2-complement representation. We have not considered this special feature to simplify the presentation of this article. However, the extension can be easily done by considering a 2-complement representation or by remarking that for

any vector *x* ∈ Z*m*, there exists a word *σ* ∈ Σ∗ such that *x* = *λσ*(− 1 *,... ,* − 1).

*r r r*

It seems that this later representation, more algebraic than the 2-complement

one, is also more concise (this is a work in progress).

Obviously, as proved by the following proposition, the binary automata and the NDD represent the same subsets of N*m*.

Proposition 4.5 *A subset X* ⊆ N*m is representable by a binary automaton if and only if X is representable by a NDD.*

Proof. First remark that if a subset *X* ⊆ N*m* is represented by a NDD, then *X* is represented by a binary automaton just because a NDD is a binary automaton. For the converse, let us consider a subset *X* ⊆ N*m* represented by

a binary automaton ffi.

We first prove that we can assume that L(ffi)*.*0∗ = L(ffi). As L(ffi) and 0∗ are two regular languages, the language L(ffi)*.*0∗ is also regular. So, we have just to prove that *ρ*(L*.*0∗) = *ρ*(L) for any subset L ⊆ Σ∗. From the inclusion L ⊆ L*.*0∗, we deduce *ρ*(L) ⊆ *ρ*(L*.*0∗). Let us prove the converse inclusion. Let *x* ∈ *ρ*(L*.*0∗). There exists *i* ≥ 0 and *σ* ∈ L such that *x* = *ρ*(*σ.*0*i*). From *x* = *ρ*(*σ.*0*i*) = *λσ.*0*i* (0*,... ,* 0) = *λσ*(*λi* (0*,... ,* 0)) = *λσ*(0*,... ,* 0) = *ρ*(*σ*) ∈ *ρ*(L),

*r*

0

we deduce *ρ*(L*.*0∗) ⊆ *ρ*(L). Hence, we can assume that L(ffi)*.*0∗ = L(ffi).

Remark that we can also assume that ffi = (*Q,* Σ*r, δ,* {*q*0}*,F* ) is deter- ministic and complete. We denote by [*i*] ∈ {0*,... ,m* − 1} the remainder of the Euclide division of *i* by *m*. Let us consider the binary automaton ffi' = (*Q*'*,* Σ*r, δ*'*,* {*q*' }*,F* ') defined by

0

*Q*' = *Q* × {0*,... ,m* − 1}





*δ*'((*q, i*)*, b*) = (*δ*(*q, b*)*,* [*i* + 1])

*q*' = (*q*0*,* 0)

 0



*F* ' = *F* × {0}

Remark that ffi' is a NDD. So we have just to prove that *ρ*(L(ffi)) = *ρ*(L(ffi')). By construction, we have L(ffi') ⊆ L(ffi). Hence, we have *ρ*(L(ffi')) ⊆ *ρ*(L(ffi)). Let us prove the converse inclusion. Let *x* ∈ *ρ*(L(ffi)). There exists *σ* ∈ L(ffi) such that *x* = *ρ*(*σ*). As L(ffi)*.*0∗ = L(ffi), we have *σ.*0*m*−[|*σ*|] ∈ L(ffi). As

the length of *σ.*0*m*−[|*σ*|] can be divided by *m*, we have proved that *σ.*0*m*−[|*σ*|] ∈ L(ffi'). From *x* = *ρ*(*σ*) = *λσ*(0*,... ,* 0) = *λσ*(*λm*−[*σ*](0*,... ,* 0)) = *ρ*(*σ.*0*m*−[*σ*]) ∈ *ρ*(L(ffi')), we deduce *ρ*(L(ffi)) ⊆ *ρ*(L(ffi')). Therefore ffi' is a NDD that repre- sents *X*.

0

# Affine hull of the set of vectors represented by a bi- nary automaton.

The affine hull of the set of vectors represented by a binary automaton ffi is proved to be computable in polynomial time.

This affine hull is obtained by labeling the states of ffi by some affine spaces: an affine covering.

Definition 5.1 An affine covering of a binary automaton ffi = (*Q,* Σ*r,* ∆*, Q*0*,F* ) is a sequence of affine spaces (*Aq*)*q*∈*Q* of Q*m* such that:

* for every state *qf* ∈ *F* we have (0*,... ,* 0) ∈ *Aqf* , and
* for every path *q* −→*σ*

*q*', we have *λσ*(*Aq*' ) ⊆ *Aq*.

Any binary automaton ffi has at least one affine covering (*Aq*)*q*∈*Q* because *Aq* = Q*m* for every *q* ∈ *Q* in an affine covering. Therefore, the following sequence (cov(ffi)*q*)*q*∈*Q* is defined.

cov(ffi)*q* = *Aq*

(*Ap*)*p*∈*Q* affine covering

Lemma 5.2 *For any binary automaton* ffi*, the sequence* (cov(ffi)*q*)*q*∈*Q is an aﬃne covering of* ffi*.*

Proof. Let us note *S* the set of affine covering of ffi. As any intersection of affine spaces is an affine space, (cov(ffi)*q*)*q*∈*Q* is a sequence of affine spaces.

Moreover, for every *qf* ∈ *F* , we have (0*,... ,* 0) ∈ cov(ffi)*q*

*f*

because for every

(*Ap*)*p*∈*Q*

*σ σ q σ* (*Ap*)*p*∈*Q*∈*S q*

in *S*, we have (0*,... ,* 0) ∈ *Aqf*

. Now, let us consider a path *q* −→

*q*'.

As *λ* is a one to one function, we have *λ* (cov(ffi) ' ) = *λ* ( *A* ' ) =

*σ*

(*Ap*)*p*∈*Q*∈*S λσ*(*Aq*' ) ⊆ (*Ap*)*p*∈*Q*∈*S Aq* = cov(ffi)*q*.

As proved by the previous lemma, the sequence (cov(ffi)*q*)*q*∈*Q* is an affine covering.

Definition 5.3 The least affine covering of a binary automaton ffi is the se- quence of affine spaces (cov(ffi)*q*)*q*∈*Q*.

From the least affine covering of a binary automaton, the following propo- sition [5.4](#_bookmark1) proves that we can compute the affine hull of the set of vectors accepted by a binary automaton.

Proposition 5.4 *For any binary automaton* ffi*, we have:*

aff(*ρ*(L(ffi))) = aff(

*q*0∈*Q*0

cov(ffi)*q*0 )

Proof. For every *q* ∈ *Q*, we consider the set *X* = {*ρ*(*σ*); ∃*q* −→ *q* ; *q* ∈

*σ*

*q f f*

*F* } ⊆ N*m*. We first prove that cov(ffi)*q* = aff(*Xq*) for every *q* ∈ *Q*.

To prove that cov(ffi)*q* ⊆ aff(*Xq*) for every *q* ∈ *Q*, we show that (aff(*Xq*))*q*∈*Q* is an affine covering. For every *qf* ∈ *F* , we have (0*,... ,* 0) = *ρ*(*ϵ*) ∈ *Xqf* . Let us

consider a path *q* −→*σ q*'

' *σ*'

and a vector *x* ∈ *Xq*' . There exists a state *qf* ∈ *F* and

'

a path *q* −→ *qf* such that *x* = *ρ*(*σ* ). Remark that *λσ*(*x*) = *λσ*(*λσ*' (0*,... ,* 0)) =

*ρ*(*σσ*'). As *q σ.σ*'

−−→

*qf* is a path, we have *λσ*

(*x*) = *ρ*(*σσ*') ∈ *Xq*. Hence

*λσ*(*Xq*' ) ⊆ *Xq*. Therefore aff(*λσ*(*Xq*' )) ⊆ aff(*Xq*). As *λσ* is an affine func- tion, we have aff(*λσ*(*Xq*' )) = *λσ*(aff(*Xq*' )). We have proved that (aff(*Xq*))*q*∈*Q* is an affine covering. By minimality of the sequence (cov(ffi)*q*)*q*∈*Q*, we have proved that cov(ffi)*q* ⊆ aff(*Xq*) for every *q* ∈ *Q*.

To prove that aff(*Xq*) ⊆ cov(ffi)*q* for every *q* ∈ *Q*, we show that *Xq* ⊆

*σ*

cov(ffi)*q* for every *q* ∈ *Q*. Let *x* ∈ *Xq*. There exists a path *q* −→ *qf* with

*qf* ∈ *F* such that *x* = *ρ*(*σ*). As (cov(ffi)*q*)*q*∈*Q* is an affine covering, we have

(0*,... ,* 0) ∈ cov(ffi)*q* and *λσ*(cov(ffi)*q* ) ⊆ cov(ffi)*q*. Therefore *x* = *ρ*(*σ*) =

*f*

*f*

*λσ*(0*,... ,* 0) ∈ cov(ffi)*q*. Hence *Xq* ⊆ cov(ffi)*q*. As cov(ffi)*q* is an affine space, we have proved that aff(*Xq*) ⊆ cov(ffi)*q*.

Hence, for every *q* ∈ *Q*, we have aff(*Xq*) = cov(ffi)*q*. Now, just remark that:

aff(*ρ*(L(ffi))) = aff(

*q*0∈*Q*0

= aff(

*q*0∈*Q*0

= aff(

*q*0∈*Q*0

*Xq*0 )

aff(*Xq*0 )) cov(ffi)*q*0 )

Hence, we are done.

Algorithm 1 Compute the affine hull of the set of vectors accepted by a binary automaton.

1: Input: A binary automaton ffi = (*Q,* Σ*r,* ∆*, Q*0*,F* ).

2: Output: The affine space aff(*ρ*(L(ffi))).

3:

4: Let (*Aq*)*q*∈*Q*

be the sequence defined by *Aq*

=  {(0*,... ,* 0)} if *q* ∈ *F*

*b* ' 

∅ otherwise

5: while there exists *q* −→ *q*

6: *Aq* ⇐ aff(*Aq* ∪ *λb*(*Aq*' ))

7: return aff( *q*0∈*Q*0 *Aq*0 )

in ∆ such that *λb*(*Aq*' ) /⊆ *Aq* do

By using a fix-point algorithm, we compute the affine covering of a binary automaton in polynomial time as proved in the following theorem.

Theorem 5.5 *The algorithm* [*1*](#_bookmark2) *computes the aﬃne hull of the set of vectors accepted by a binary automaton* ffi *in polynomial time.*

Proof. We first prove that line 6 is executed at most (*m* + 1)*.*card(*Q*) times. In fact, remark that each times line 6 is executed, the dimension of the affine space *Aq* strictly increase. As the dimension of an affine space of Q*m* is in the finite set {−1*,... , m*}, we have the result.

To prove that the complexity of the algorithm is polynomial, we prove that the representations of the affine spaces computed by the algorithm are some finite subsets of at most *m* + 1 vectors in the set {0*,..., r*2*.*card(*Q*) − 1}*m*.

Let us prove that the representations *X* of the affine spaces *A* computed by the algorithm are some finite subset of at most *m* + 1 vectors in {*ρ*(*σ*); *σ* ∈

Σ≤(*m*+1)*.*card(*Q*)}. To do so, we denote by (*Ai* )

the sequence of affine spaces

*r q q*∈*Q*

(*Aq*)*q*∈*Q* on line 5 in function of the number of times *i* ≥ 0 that the “while”

loop has been executed. We denote by *Xi* the representation of *Ai* . Let us

*q q*

prove by induction over *i* ≥ 0 that for every *q* ∈ *Q*, *Xq* is a finite subset of at most *m* + 1 vectors in {*ρ*(*σ*); *σ* ∈ Σ≤*i*}. Remark that the induction is true at rank *i* = 0. Assume the induction true at rank *i* ≥ 0 and let us prove that the

*i*

*r*

induction remains true at rank *i*+1. Remark that there exists *q* −→ *q*' in ∆ such

*b*

that for every *p* ∈ *Q/*{*q*}, we have *Ai*+1 = *Ai* and *Ai*+1 = aff(*Ai* ∪ *λb*(*Ai* ' )).

*p p q q q*

Therefore, for every *p* /= *q*, we have *Xi*+1 = *Xi* and *Xi*+1 ⊆ *Xi* ∪ *λb*(*Xi*' ). By

*p p q q q*

using the induction at rank *i*, we obtain *Xi*+1 ⊆ {*ρ*(*σ*); *σ* ∈ Σ≤*i*+1} for every

*p r*

*p* ∈ *Q*. There, we have proved the induction at rank *i* + 1. We have proved

the induction. In particular, as *i* ≤ (*m* + 1)*.*card(*Q*), we have proved that the representations *X* of the affine spaces computed by the algorithm are finite subsets of at most *m* + 1 vectors in {*ρ*(*σ*); *σ* ∈ Σ≤(*m*+1)*.*card(*Q*)}.

*r*

Now let us prove that {*ρ*(*σ*); *σ* ∈ Σ≤(*m*+1)*.*card(*Q*)} ⊆ {0*,..., r*2*.*card(*Q*) −1}*m*. Let us consider *σ* ∈ Σ≤(*m*+1)*.*card(*Q*). There exists *i* ≥ 0 such that *σ.*0*i* is a word of length *m.*2*.*card(*Q*). Therefore *ρ*(*σ*) ∈ {0*,..., r*2*.*card(*Q*) − 1}*m*.

*r*

*r*

We have proved that the affine spaces computed by the algorithm are represented by at most *m* + 1 vectors in {0*,... , r*2*.*card(*Q*)}*m*. The complexity of the algorithm is therefore polynomial.

To prove the correctness of the algorithm, we first show that the following assertion is an invariant: “ *Aq* ⊆ cov(ffi)*q* for every *q* ∈ *Q* and (0*,... ,* 0) ∈ *Aq* for every *qf* ∈ *F* ”. Remark that just after the execution of line 4, the assertion is true. So assume the assertion true before the executing of line 6 and let us

*f*

prove that the assertion remains true just after. Let *q* −→ *q*' be a transition

*b*

in ∆. As (cov(ffi)*q*)*q*∈*Q* is an affine covering, we have *λb*(cov(ffi)*q*' ) ⊆ cov(ffi)*q*. From the assertion we deduce *λb*(*Aq*' ) ⊆ cov(ffi)*q*. Moreover the assertion also gives *Aq* ⊆ cov(ffi)*q*. Hence *Aq* ∪ *λb*(*Aq*' ) ⊆ cov(ffi)*q*. By considering the affine hull of the previous inclusion, we obtain aff(*Aq* ∪ *λb*(*Aq*' )) ⊆ cov(ffi)*q*. Therefore the assertion is an invariant of the algorithm.

In particular, on line 7 we have *Aq* ⊆ cov(ffi)*q* for every *q* ∈ *Q* and (0*,... ,* 0) ∈ *Aqf* for every *qf* ∈ *F* . However, on this line, the while condi-

tion on line 5 is no longer valid. Hence, for every *q*

−→*b*

*q*' in ∆, we have

*σ* '

*λb*(*Aq*' ) ⊆ *Aq*. By induction, we obtain *λσ*(*Aq*' ) ⊆ *Aq* for every path *q* −→ *q* .

Moreover, as (0*,... ,* 0) ∈ *Aqf* for every *qf* ∈ *F* , we have prove that (*Aq*)*q*∈*Q* is an affine covering. Therefore, for every *q* ∈ *Q*, we have cov(ffi)*q* ⊆ *Aq*. We deduce *Aq* = cov(ffi)*q* for every *q* ∈ *Q* on line 7.

Proposition [5.4](#_bookmark1) proves that on line 7, we have aff( *q*0 ∈*Q*0 *Aq*0 ) = aff(*ρ*(L(ffi))).

# Application.

We study a natural extension of the Petri Nets, the effective counter systems, a class of systems that contains all the counter systems studied by Fast and Lash. For these systems, we naturally extend the definition of place invariants and prove that the computation of these invariants remains polynomial.

In the subsection [6.1](#_bookmark4), we prove that we can compute the affine hull of the reachability relation of a counter system in polynomial time in function of the affine hull of the reachability relation in one step. This polynomial time complexity is explained in the next subsection [6.2](#_bookmark6). In fact, we show the polynomial time link between the affine hull of the reachability relation of a counter system and a natural extension of place invariants. In the last subsection [6.3](#_bookmark8) we prove that the place invariants of an effective counter system can be computed in polynomial time.

* 1. *The aﬃne hull of the reachability relation of a counter system.*

In this subsection, we prove that the affine hull of the reachability relation of a counter system can be computed in polynomial time in function of the affine hull of the reachability relation in one step.

A counter system is a general model for representing a system that uses *m* integer variables and has a finite set of actions Σ such that the new values *x*' ∈ N*m* of the counters after executing an action *a* ∈ Σ are related to the current value *x* ∈ N*m* of the counters by a binary relation *x*R*ax*' over N*m*.

Definition 6.1 A counter system *S* is a tuple *S* = (N*m,* Σ*,* (R*a*)*a*∈Σ) where Σ is a finite set of actions and (R*a*)*a*∈Σ is a sequence of binary relations over N*m*.

The *one step reachability relation* associated to a counter system *S* is the binary relation written R*S* and defined by R*S* = *a*∈Σ R*a*. The *reachability*

*relation* associated to a counter system *S* is the binary relation R∗

*S*

equal to

reflexive and transitive closure of the one step reachability relation. Remark that *x*R∗ *x*' iff there exists a finite sequence of *n* ≥ 0 actions (*ai*)1≤*i*≤*n* in Σ and a sequence of vectors (*xi*)0≤*i*≤*n* in N*m* such that *x* = *x*0, *x*0R*a*1 *x*1, ..., *xn*−1R*a xn* and *xn* = *x*'.

*S*

*n*

The reachability problem consists in deciding for a counter system *S* and two vectors *x* and *x*' in N*m* if we have *x*R∗ *x*'. Recall that this problem is undecidable for the class of Reset/Transfer Petri Nets [[15](#_bookmark24)], a subclass of

*S*

counter systems. For this reason, we are interested in the computation of an over approximation of the reachability relation, easily computable.

The following proposition proves that the computation of the affine hull of

*I* ∪ R*S* provides the affine hull of the reachability relation R∗ .

*S*

Proposition 6.2 *For any binary relation* R *over* Q*m, we have*

aff(R∗) = aff(*I* ∪ R)

Proof. From *I* ∪ R ⊆ R∗, we deduce aff(*I* ∪ R) ⊆ aff(R∗). To prove the converse, we first show that aff(*I* ∪ R) is a transitive relation. Let us consider (*x, x*') and (*x*'*, x*'') in aff(*I* ∪ R). Remark that if *x* = *x*' = *x*'' then (*x, x*'') = (*x*'*, x*') ∈ *I* ⊆ aff(*I* ∪ R). Hence, we can assume that either *x* /= *x*' or *x*' /= *x*''. The rank of the following linear system is then equal to 2 and hence admits at least one solution (*α, β, γ*) ∈ Q3.

*α* + *β.x* + *γ.x*' = *x α* + *β.x*' + *γ.x*'' = *x*''

Let *δ* = 1 − (*α* + *β* + *γ*). From *α* + *β* + *γ* + *δ* = 1, and as (1*,* 1), (*x, x*'), (*x*'*, x*'') and (0*,* 0) are in the affine space aff(*I* ∪ R), we deduce (*x, x*'') = *α.*(1*,* 1) + *β.*(*x, x*') + *γ.*(*x*'*, x*'') + *δ.*(0*,* 0) ∈ aff(*I* ∪ R). Therefore, we have proved that the relation aff(*I* ∪ R) is transitive.

From R ⊆ aff(*I* ∪ R) and the transitivity of aff(*I* ∪ R), we deduce R∗ ⊆

aff(*I* ∪ R). Therefore aff(R∗) ⊆ aff(*I* ∪ R).

The previous proposition shows that the affine hull of R∗ is equal to the

*S*

affine hull of *I* ∪ R*S*. From aff(*I* ∪ R*S*) = aff(*I a*∈Σ aff(R*a*)), we deduce the following corollary:

Corollary 6.3 *The aﬃne hull of the reachability relation of a counter sys- tem S can be computed in polynomial time from the sequence of aﬃne spaces* (aff(R*a*))*a*∈Σ*.*

* 1. *The deﬁnition of place invariants.*

In this section we show the link between aff(R∗ ) and the notion of place

*S*

invariant of a counter system.

We slightly extend the definition of place invariants given in [[11](#_bookmark19)] in a natural way.

Definition 6.4 A place invariant of a counter system *S* is a linear function *l* : Q*m* → Q such that for every *a* ∈ Σ and for every (*x, x*') satisfying *x*R*ax*', we have *l*(*x*) = *l*(*x*'). The set of place invariants of a counter system *S* is written inv(*S*).

Remark 6.5 The set of place invariants of a counter system *S* is an affine space (and even a vector space).

The following proposition proves the link between inv(*S*) and aff(R∗ ) by proving that the set of place invariants inv(*S*) is computable in polynomial time from aff(R∗ ) and that aff(R∗ ) is computable in polynomial time from

*S*

*S S*

inv(*S*).

Proposition 6.6 *For any counter system S, we have:*

aff(R∗ ) = {(*x, x*'); *l*(*x*) = *l*(*x*') ∀*l* ∈ inv(*S*)}

*S*

inv(*S*) = {*linear functions l* : Q*m* → Q; *l*(*x*) = *l*(*x*') ∀(*x, x*') ∈ aff(R∗ )}

*S*

Proof. We first prove the equality aff(R∗ ) = {(*x, x*'); *l*(*x*) = *l*(*x*') ∀*l* ∈

*S*

inv(*S*)}.

Let (*x*0*, x*' ) ∈ aff(R∗ ) and let us prove that for any *l* ∈ inv(*S*), we have

0 *S*

*l*(*x*0) = *l*(*x*' ). From the proposition [6.2](#_bookmark5), we have (*x*0*, x*' ) ∈ aff(*I* ∪ R*S*).

0 0

There exist *t* ∈ Q, a sequence (*ta*)*a*∈Σ in Q, a vector (*x, x*) ∈ *I* and a sequence

((*xa, x*' ))*a*∈Σ in Y*a* such that *t*+Σ *ta* = 1 and such that (*x*0*, x*' ) = *t.*(*x, x*)+

Σ *a* '

*a*∈Σ *ta.*(*xa, xa*). Let *l* ∈ inv(*S*). By definition of the place invariants, we have

*a*∈Σ 0

*l*(*xa*) = *l*(*x*' ) for every *a* ∈ Σ. Hence, we have *l*(*x*0) = *l*(*t.x* + Σ

*ta.xa*) =

Σ *a* Σ '

*t.l*(*x*) +

*ta.l*(*xa*) = *t.l*(*x*) +

*ta.l*(*xa*) = *l*(*t.x* +

*ta.x*' ) = *l*(*x* ).

Therefore, we have proved the inclusion aff(Y∗ ) ⊆ {(*x, x*'); *l*(*x*) = *l*(*x*') 6*l* ∈

*a*∈Σ

*a*∈Σ

Σ *a*∈Σ '

*a*∈Σ

*a*

0

*S*

inv(*S*)}. Let us prove the converse.

Let us consider (*x*0*, x*' ) /∈ aff(Y∗ ). In this case there exists an affine

0 *S*

function *f* : Q*m* × Q*m* → Q such that *f* (*x, x*') = 0 for every (*x, x*') ∈ aff(Y∗ ) and such that *f* (*x*0*, x*' ) /= 0. As *f* is an affine function, there exist two linear functions *l* and *l*' and a rational *c* ∈ Q such that for every (*x, x*') ∈ Q*m* × Q*m*, we have *f* (*x, x*') = *l*(*x*) — *l*'(*x*') + *c*. From (*x, x*) ∈ *I* ⊆ aff(Y∗) we deduce 0 = *f* (*x, x*) = *l*(*x*) — *l*'(*x*') + *c*. Therefore *l* = *l*' and *c* = 0. Moreover, for every *a* ∈ Σ and for every (*x, x*') ∈ Y*a*, we have 0 = *f* (*x, x*') = *l*(*x*) — *l*(*x*').

*S*

0

Therefore *l* ∈ inv(*S*). From *f* (*x*0*, x*' ) /= 0, we deduce *l*(*x*0) /= *l*(*x*' ). We have

0 0

proved the inclusion {(*x, x*'); *l*(*x*) = *l*(*x*') 6*l* ∈ inv(*S*)} ⊆ aff(Y∗ ).

*S*

We deduce the first equality aff(Y∗ ) = {(*x, x*'); *l*(*x*) = *l*(*x*') 6*l* ∈ inv(*S*)}. Now, let us prove the second equality. Let *l* ∈ inv(*S*). From the previous equality, we have *l*(*x*) = *l*(*x*') for every (*x, x*') ∈ aff(Y∗ ). Therefore inv(*S*) ⊆

*S*

*S*

{linear functions *l* : Q*m* → Q; *l*(*x*) = *l*(*x*') 6(*x, x*') ∈ aff(Y∗ )}. For the converse inclusion, let us consider a linear function *l* such that *l*(*x*) = *l*(*x*') for

*S*

every (*x, x*') ∈ aff(Y∗ ). From Y*S* ⊆ aff(Y∗ ), we deduce that for every *a* ∈ Σ

*S S*

and for every (*x, x*') ∈ Y*a*, we have *l*(*x*) = *l*(*x*'). Therefore *l* ∈ inv(*S*). We have proved the second equality.

* 1. *Effective counter systems.*

In this subsection, we define the class of effective counter systems and we prove that the affine hull of the reachability relation is computable in polynomial time. As a corollary, we prove that the set of place invariants of an effec- tive counter systems are computable in polynomial time. Finally, we present

results obtained by implementing these results in our tool Fast.

Tools like Fast or Lash takes as input a class of counter systems such that each binary relation Y*a* is represented by a BA-affine function *fa*.

Definition 6.7 A BA-affine function *f* is a tuple (ffi*, M, v*) such that ffi is a binary automaton, *M* ∈ N*m*(Q) is a square matrix and *v* ∈ Q*m* is a vector. The relation Y*f* associated to a BA-affine relation *f* is defined by:

*x*Y*f x*' ⇐⇒ *x*' = *M.x* + *v* and *x* ∈ *ρ*(L(ffi))

Definition 6.8 An effective counter system *S* is a counter system *S* = (N*m,* Σ*,* YΣ)

such that every binary relation Y*a* is either represented by a binary automaton ffi*a* such that Y*a* = *ρ*(L(ffi*a*)) ⊆ N2*.m* or represented by a BA-affine function *fa* such that Y*a* = Y*fa* .

For effective counter systems, the complexity for the computation of place invariants, known to be polynomial time for Petri Nets remains polynomial time.

Theorem 6.9 *The aﬃne hull of the reachability relation of an effective counter system is computable in polynomial time.*

Proof. From the theorem [6.2](#_bookmark5), we prove that aff(Y∗ ) is equal to aff(*I* ∪ Y*S*).

*S*

Remark that aff(*I* ∪Y*S*) = aff(*I*

*a*∈Σ aff(Y*a*)). Let *a* ∈ Σ; the binary relation

Y*a* is either represented by a binary automaton ffi*a* or by a BA-affine function *fa*. If Y*a* is represented by a binary automaton ffi*a*, theorem [5.5](#_bookmark3) proves that the affine space aff(Y*a*) is computable in polynomial time. Assume that Y*a* is represented by a BA-affine function *fa* = (ffi*a, Ma, va*). Theorem [5.5](#_bookmark3) proves that aff(*ρ*(L(ffi*a*))) is computable in polynomial time. Therefore, aff(Y*fa* ) = (aff(*ρ*(L(ffi*a*))) × Q*m*) ∩ {(*x, x*') ∈ Q*m* × Q*m*; *x*' = *Ma.x* + *va*} is computable in polynomial time. Therefore, we can compute aff(Y*a*) in polynomial time for each action *a* ∈ Σ. We have proved that we can compute aff(Y∗ ) in polynomial time.

From the proposition [6.6](#_bookmark7) and the previous theorem, we deduce that the place invariants of an effective counter systems are computable in polynomial time.

*S*

Corollary 6.10 *The set of place invariants of an effective counter system is computable in polynomial time.*

We have implemented the computation of the affine covering of a binary automaton as a plug-in for our symbolic model checker Fast. The compu- tation of the place invariants of the 40 examples of counter systems provided with Fast takes between 1 minute to 1 hour to be computed. However, this time complexity do not correponds to the time complexity of an optimized al- gorithm. In fact, we are quite sure that the required time for computing these place invariants can be reduced to less than one second. For this reason, no benchmark are provided. Fast with invariants will be available as soon as this

algorithm is optimized on the fast-webpage [www.lsv.ens-cachan.fr/fast/](http://www.lsv.ens-cachan.fr/fast/)

# Conclusion.

We have presented the class of binary automaton, a new representation of sub- sets of N*m* that naturally extends the NDD. We have proved that the affine

hull of the set of vectors represented by a binary automaton is computable in polynomial time. We have introduced the model of effective counter sys- tems. We have proved that we can extend the definition of place invariants to this new class. The computation of the place invariants remains polynomial time for the class of effective counter systems. All the algorithms have been

implemented in C++ as a plug-in for Fast and they should be soon available.

# References

1. Abdulla, P. A., A. Bouajjani and B. Jonsson, *On-the-fly analysis of systems with unbounded,* *lossy FIFO channels*, in: *Proc. 10th Int. Conf. Computer Aided Verification (CAV’98), Vancouver, BC, Canada, June-July 1998*, Lecture Notes in Computer Science 1427 (1998),

pp. 305–318.

1. Abdulla, P. A. and B. Jonsson, *Verifying programs with unreliable channels*, in: *Proc. 8th IEEE Symp. Logic in Computer Science (LICS’93), Montreal, Canada, June 1993* (1993), pp. 160–170.
2. Ammirati, P., G. Delzanno, P. Ganty, G. Geeraerts, J.-F. Raskin and L. V. Begin, *Babylon: An integrated toolkit for the specification and verification of parameterized systems*, in: G. Delzanno,

S. Etalle and M. Gabbrielli, editors, *Specification, Analysis and Validation for Emerging* *Technologies in Computational Logic*, Datalogiske Skrifter 94, 2002, p. (unpaginated).

1. Annichini, A., A. Bouajjani and M. Sighireanu, *TReX: A tool for reachability analysis of* *complex systems*, in: *Proc. 13th Int. Conf. Computer Aided Verification (CAV’2001), Paris, France, July 2001*, Lecture Notes in Computer Science 2102 (2001), pp. 368–372.
2. *BABYLON*, <http://www.ulb.ac.be/di/ssd/lvbegin/CST/index.html>.
3. Bardin, S., 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*, Lecture Notes in Computer Science 2725 (2003), pp. 118–121.
4. Berthelot, G., “Transformations et Analyse de Reseaux de Petri. Applications aux Protocoles.” Th`ese d’Etat, Univ. Paris VI, 1983, newsletterInfo: 16.
5. Boigelot, B., *On iterating linear transformations over recognizable sets of integers*, Theoretical Computer Science. To appear.
6. Boigelot, B., “Symbolic Methods for Exploring Infinite State Spaces,” Ph.D. thesis, Universit´e de Li`ege (1998).
7. Boudet, A. and H. Comon, *Diophantine equations, Presburger arithmetic and finite automata*, in: *Proc. 21st Int. Coll. on Trees in Algebra and Programming (CAAP’96), Linko¨ping, Sweden, Apr. 1996*, Lecture Notes in Computer Science 1059 (1996), pp. 30–43.
8. Ciardo, G., *Petri nets with marking-dependent arc cardinality: Properties and analysis*, in: *Proc. 15th Int. Conf. Application and Theory of Petri Nets (ICATPN’94), Zaragoza, Spain, June 1994*, Lecture Notes in Computer Science 815 (1994), pp. 179–198.
9. Delzanno, G., *Constraint-based verification of parameterized cache coherence protocols*, Formal Methods in System Design. To appear.
10. Delzanno, G., *Verification of consistency protocols via infinite-state symbolic model checking: A case study*, in: *Proc. IFIP Joint Int. Conf. Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE-PSTV’00), Pisa, Italy, Oct. 2000*, IFIP Conference Proceedings 183 (2000), pp. 171–186.
11. Delzanno, G., J.-F. Raskin and L. Van Begin, *Attacking symbolic state explosion*, in: *Proc. 13th Int. Conf. Computer Aided Verification (CAV’2001), Paris, France, July 2001*, Lecture Notes in Computer Science 2102 (2001), pp. 298–310.
12. Dufourd, C., 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*, Lecture Notes in Computer Science 1443 (1998), pp. 103–115.
13. Emerson, E. A. and K. S. Namjoshi, *On model checking for non-deterministic infinite-state systems*, in: *Proc. 13th IEEE Symp. Logic in Computer Science (LICS’98), Indianapolis, IN,* *USA, June 1998* (1998), pp. 70–80.
14. *FAST*, <http://www.lsv.ens-cachan.fr/fast/>.
15. Finkel, A. 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*, Lecture Notes in Computer Science 2556 (2002),

pp. 145–156.

1. Finkel, A. and P. Schnoebelen, *Well structured transition systems everywhere!*, Theoretical Computer Science 256 (2001), pp. 63–92.
2. Klarlund, N., *Mona and fido: The logic-automaton connection in practice* (1997).
3. Klarlund, N., A. Møller and M. I. Schwartzbach, *MONA implementation secrets*, Int. J. of Foundations Computer Science 13 (2002), pp. 571–586.
4. *Lash*, <http://www.montefiore.ulg.ac.be/~boigelot/research/lash/>.
5. *MONA*, <http://www.brics.dk/mona/index.html>.
6. *TREX*, <http://www.liafa.jussieu.fr/~sighirea/trex/>.
7. Wolper, P. and B. Boigelot, *An automata-theoretic approach to Presburger arithmetic constraints*, in: *Proc. 2nd Int. Symp. Static Analysis (SAS’95), Glasgow, UK, Sep. 1995*, Lecture Notes in Computer Science 983 (1995), pp. 21–32.
8. Wolper, P. 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*, Lecture Notes in Computer Science 1785 (2000), pp. 1–19.