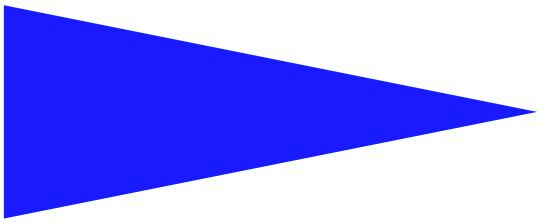
I R I S A
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION
INTERNE
N° 1279



SYMBOLIC ABSTRACTIONS OF AUTOMATA AND THEIR APPLICATION TO THE SUPERVISORY CONTROL PROBLEM

S. PINCHINAT, H. MARCHAND, M. LE BORGNE

ISSN 1166-8687

IRISA
CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

IRISA
Tél. : (33) 02 99 84 71 00 – Fax : (33) 02 99 84 71 71
http://www.irisa.fr

Symbolic Abstractions of Automata and their application to the Supervisory Control Problem

S. Pinchinat, H. Marchand, M. Le Borgne

Thème 1 — Réseaux et systèmes
Projet EP-ATR

Publication interne n ° 1279 — Novembre 1999 — 29 pages

Abstract: In this report, we describe the design of abstraction methods based on symbolic techniques: classical **abstraction by state fusion** has been considered. we present a general method to abstract automata on the basis of a *state fusion criterion*, derived from e.g. equivalence relations (such as bisimulation), partitions, ... We also introduced other kinds of abstraction, falling into the category of **abstraction by restriction**: in particular, we studied the use of the controller synthesis methodology to achieve the restriction synthesis. The methods rely on symbolic representation of the labeled transition system, namely the Intensional Labeled Transition System (ILTS). It is a behavioral model for Discrete event systems based on polynomial approach, that has effective applications for the analysis of SIGNAL programs. We finally apply this methodology to solve the Supervisory Control Problem.

Key-words: Intentional transition systems, polynomials, symbolic bisimulations, model reduction, BDD, Supervisory control Problem

(Résumé : tsvp)

Centre National de la Recherche Scientifique
(UPRESSA 6074) Université de Rennes 1 – Insa de Rennes

Institut National de Recherche en Informatique
et en Automatique – unité de recherche de Rennes

Abstractions symboliques d'automates et son application au problème de la synthèse de contrôleurs.

Résumé : Nous présentons diverses méthodes d'abstraction basées sur des techniques symboliques. Nous avons défini une méthode générale permettant d'abstraire un automate sur la base d'un *critère de fusion d'états*, dérivé par exemple de relations d'équivalence (eg bisimulation), de partitions, etc. Nous avons également introduit d'autres types d'abstraction comme par exemple *l'abstraction par restriction*. En particulier, nous avons regardé comment les techniques de synthèse de contrôleurs pouvaient être utilisées dans cette optique. L'approche est basée sur des modèles comportementaux *intentionnels* (encore appelés symboliques ou implicites), obtenus par abstraction booléenne de spécifications en langage SIGNAL. Finalement, nous avons montré comment appliquer les techniques d'abstraction par fusion au problème de la synthèse de contrôleur

Mots clés : Systèmes de transitions intentionnels, méthodes polynomiales, bisimulation symbolique, réduction de modèles, BDD, synthèse de contrôleurs

1 Introduction

In [13, 14], we presented *Implicit Labeled Transition Systems* (ILTS) as intermediate models for discrete event systems. We have studied operations of parallel composition and event hiding, as well as an equivalence criterion based on strong bisimulation semantics, namely the equivalence called *symbolic bisimulation*. The aim of the work is to rely on intensional descriptions of the systems for symbolic abstractions purposes, based on behavioral equivalences. The intensional approach we propose has the main advantage to remain at an interesting level of abstraction to handle sets (of states, of events) in which algorithms and properties can entirely be expressed, thus providing us with a high level of expressiveness. This abstract level avoids us to bother with a particular choice of sets implementation, such as Binary Decision Diagrams (BDDs), even if all operations are actually based on this representation for sets. Moreover, the intensional formalism is completely compatible with the symbolic techniques, since intensionally described sets can afterwards be implemented by any kind of methods, e.g. decision diagrams. However, other tools with various implementations for sets might be considered using the same high level framework (e.g. Mathematica, Maple).

In this report, we achieved the design of abstraction methods based on symbolic techniques: classical **abstraction by state fusion** has been considered (with associated equivalence relation computation). We also introduced other kinds of abstraction, falling into the category of **abstraction by restriction**: in particular, we studied the use of the controller synthesis methodology to achieve the restriction synthesis.

- **Abstraction by state fusion**

  The principles of abstraction by state fusion is classical : super states are obtained by fusionning concrete states and transitions are naturally lifted to super states. For this kind of abstraction, compositionality is trivial. In this setting, we propose a family of algorithms to build the abstracted system, in the symbolic/intensional philosophy. More precisely, we present a general method to abstract automata on the basis of a *state fusion criterion*, derived from e.g. equivalence relations (such as bisimulation), partitions, ...

  The approach has the advantage to deliver the abstracted model still in an intensional style, whereas in classical approaches symbolic models might become explicit when submitted to reduction. However, the state fusion computation might still have some explicit aspects features when derived from equivalence relations : we cannot yet avoid the equivalence classes enumeration, during which the reduced model is built "on the fly". Our choice of equivalence relations mainly focuses on the computation of greatest bisimulations (strong, but also weak/delay/branching), to ensure behavioral properties preservation. However, any kind of relations *larger* than the bisimulations can be considered (in particular, partitions, general relations,..., provided the user takes care of the correction of its abstraction).

  All these algorithms (strong/weak/delay/branching bisimulations, reduced model computation,...) have been implemented in the SIGALI tool box. These new functionalities of SIGALI are of high interest since the SIGNAL language is used in a lot of areas (controller synthesis [8], [20], robotics [19],...) where models equivalence checking is crucial. Indeed, using reduction techniques, we hope for dealing with smaller models, then improving the efficiency of verification/synthesis stage.

- **Abstraction by restriction**

  This abstraction aims to simplify the model by disallowing some behaviors. A naive approach would consist in modifying the structure of the automata by removing either a set of states or a set of events. The corresponding set of states could possibly be induced by a property (e.g. invariant property,...). Also, observing automata can be composed with the original one in order

to get more subtle restrictions of the behaviors. Both techniques have been considered, and for the latter one, we have explored particular kind of restrictions : as in the control theory world [23], the status of events is twofold. Some events, called *uncontrollable*, cannot be prevented from occurring (e.g. think of an open system in an environment), whereas the others, though *controllable* can be disabled. In this new framework, the abstraction by restriction needs being refined, and falls into the controller synthesis issues by adding constraints to the initial system (by means of new boolean/polynomial equations), called the *controller*.

Moreover, based on symbolic bisimulation techniques, we also present methods for solving the classical Supervisory Control Problem (SCP) [29], using algorithms based on bisimulation techniques. [3] first presented the relations between bisimulation and controllability and provides algorithms for solving the SCP. We show how to solve the same problem using Intentional Labeled Transition System (ILTS) as model for the plant. The approach, based on algebraic transformations relying on BDD techniques, leads to more efficient algorithms.

The report is organized as follows: Section 2 recalls the basic definition of the *Intensionally Labeled Transition Systems (ILTS)* framework. On the basis of those models, we then propose, in Section 3, symbolic algorithms which compute the reduced system according to a state fusion criterion as well as bisimulation relations. Section 4 is devoted to the abstraction by restriction presentation by means of restriction synthesis methodology. The application of these techniques to the supervisory control problem is finally presented in Section 5.

2 The Intentional Labeled Transition Systems (ILTS)

2.1 The Mathematical Framework

In the following, we write $\mathbb{Z}/_{p\mathbb{Z}}$ for the finite field $\{0, 1, \ldots, p-1\}$, wehre p is a prime number. Let Z be a finite set of k distinct variables $Z_1, ..., Z_k$, and similarly for X, Y, etc (note that in the sequel X (resp. Y) will generally denote the set of state variables (resp. event variables)). The ring[1] of the polynomials in the variables $Z = (Z_1, \ldots Z_l)$ with coefficients in $\mathbb{Z}/_{p\mathbb{Z}}$ will be denoted by $\mathbb{Z}/_{p\mathbb{Z}}[Z]$. Given an element of $\mathbb{Z}/_{p\mathbb{Z}}[Z]$, $P(Z_1, Z_2, \ldots, Z_k)$ (shortly $P(Z)$), we associate its set of solutions $Sol(P) \subseteq (\mathbb{Z}/_{p\mathbb{Z}})^m$:

$$Sol(P) \stackrel{\text{def}}{=} \{(z_1, ..., z_k) \in (\mathbb{Z}/_{p\mathbb{Z}})^k | P(z_1, ..., z_k) = 0\} \quad (1)$$

It is worthwhile noting that in $\mathbb{Z}/_{p\mathbb{Z}}[Z]$, $Z_1^p - Z_1, ..., Z_k^p - Z_k$ evaluate to zero. Then for any $P(Z) \in \mathbb{Z}/_{p\mathbb{Z}}[Z]$, one has $Sol(P) = Sol(P + (Z_i^p - Z_i))$. A natural abstraction modulo $\equiv$-equivalence over polynomials, where $P_1 \equiv P_2$ means $Sol(P_1) = Sol(P_2)$. We then introduce the quotient ring of polynomial functions $A[Z] = \mathbb{Z}/_{p\mathbb{Z}}[Z]/_{<Z^p - Z>}$, where all polynomials $Z_i^p - Z_i$ are identified to zero, written for short $Z^p - Z = 0$. $A[Z]$ can be regarded as the set of polynomial functions with coefficients in $\mathbb{Z}/_{p\mathbb{Z}}$ for which the degree in each variable is lower than $(p-1)$. This will be very useful for algorithmic purposes. [8] showed how to define a representative of $Sol(P)$, i.e. of each $\equiv$-equivalence class, called the *canonical generator*.

Lemma 1 [8] *Given a polynomial $P \in \mathbb{Z}/_{p\mathbb{Z}}[Z]$, there exists a canonical generator of $[P]_{\equiv}$* $\diamond$

Our techniques will rely on the following:

Property 1 *For all $P_1, P_2, P \in \mathbb{Z}/_{p\mathbb{Z}}[Z]$*

[1] A commutative ring R is a set of elements and two operations, + and *, both are commutative, associative, distributive and closed in R. + and * have identity elements, 0 and 1 respectively. For every element $a \in R$ there exists an element $b \in R$ such that $a + b = 0$, i.e. + has an inverse.

- $Sol(P_1) \subseteq Sol(P_2)$ *whenever* $(1 - P_1^{p-1}) * P_2 \equiv 0$.
- $Sol(P1) \cap Sol(P_2) = Sol(P_1 \oplus P_2)$, *where* $P_1 \oplus P_2 \stackrel{def}{=} (P_1^{p-1} + P_2^{p-1})^{p-1}$
- $Sol(P_1) \cup Sol(P_2) = Sol(P_1 * P_2)$
- $(\mathbb{Z}/_{p\mathbb{Z}})^m \setminus Sol(P) = Sol(1 - P^{p-1})$. ◇

Notations: In the following, we shall use a couple of shortcuts which fit the intuition one has about sets operations:

- $\overline{P} = 1 - P^{p-1}$. So $Sol(\overline{P}) = (\mathbb{Z}/_{p\mathbb{Z}})^m \setminus Sol(P)$.
- $P_1 \Rightarrow P_2$ will denote the set $\{z \in \mathbb{Z}/_{p\mathbb{Z}^k} | P_1(z) = 0 \Rightarrow P_2(z) = 0\}$. It is equal to $\overline{P_1} * P_2$

Finally, we introduce the *existential/universal abstractions* (or quantifications) over polynomials w.r.t. some variables. Let $P \in \mathbb{Z}/_{p\mathbb{Z}}[Z]$, we shall write $\exists Z_i P$ for the polynomial

$$\exists Z_i P \stackrel{\text{def}}{=} P|_{Z_i=1} * P|_{Z_i=2} * \ldots * P|_{Z_i=p}, \tag{2}$$

where $P|_{Z_i=v}$ is P obtained by instantiating any occurrence of variable Z_i by value v. Also when $\tilde{Z} \subset Z$ is some $\{Z_{i_1}, ..., Z_{i_r}\}$ we write $\exists \tilde{Z} P$ for $\exists Z_{i_1} ... \exists Z_{i_r} P$. Also it is possible to define a dual variable abstraction over polynomials, based on universal quantificator: $\forall Z_i P$ is computed as

$$\forall Z_i P \stackrel{\text{def}}{=} P|_{Z_i=1} \oplus P|_{Z_i=2} \oplus \ldots \oplus P|_{Z_i=p}, \tag{3}$$

which solutions are elements of the form $(z_1, ..., z_{i-1}, z_{i+1}, ..., z_k)$ s.t. $\forall z_i, (z_1, ..., z_{i-1}, z_i, z_{i+1}, ..., z_k) \in Sol(P)$.

Polynomial Implementation: it turns out that the best known implementation (for memory and computation performance) of polynomials over finite field (i.e. $\mathbb{Z}/_{p\mathbb{Z}}$) is based on their decomposition according to the Lagrange polynomials, leading to p-ary decision diagrams data structures. A classical instance of this approach is the well-known Shannon decomposition for the case $p = 2$, with associated Binary Decision Diagrams (BDD) for the boolean functions [6]. We here present the case of $p = 3$: Polynomials are then encoded by a Ternary Decision Diagrams (TDD) [16], a slight extension of BDD. The TDD are the actual implementation of the polynomial in our formal calculus system SIGALI (see Annex A).

In the quotient ring $A[Z] = \mathbb{Z}/_{3\mathbb{Z}}[Z]/_{<Z^3-Z>}$, for each variable Z_i, let us define the three polynomial functions: $e_i^1 = -Z_i^2 - Z_i$, $e_i^2 = -Z_i^2 + Z_i$, $e_i^3 = 1 - Z_i^2$.

Proposition 1 *Each* $P(Z) \in A[Z]$ *can be decomposed in a unique way as* $P(Z) = e_1^1 P_1 + e_2^1 P_2 + e_1^3 P_3$, *where polynomials* P_1, P_2, P_3 *have the following form:* $P_1 = P(1, Z_2, .., Z_n)$, $P_2 = P(-1, Z_1, .., Z_n)$, $P_3 = P(0, Z_1, .., Z_n,)$. ◇

We can then decompose all polynomial functions using the basis of monomials $e_1^{\alpha_1} \ldots e_n^{\alpha_n}$. Hence, given a polynomial function P in $A[Z]$, and an order $Z_1 \prec Z_2 \prec \ldots \prec Z_n$ on the variables, an h-expression of P is either $P(Z) = c_1 e_i^1 \ + c_2 e_i^2 \ + \ c_3 e_i^3$ where $c_i \in \mathbb{Z}/_{3\mathbb{Z}}$, or $P(Z) = e_i^1 P_1 \ + e_i^2 P_2 \ + e_i^3 P_3$ where the P_i are h-expressions with variables greater than Z_i. An h-expression may be pictured as a ternary tree as shown by Figure 2.1, through an example.

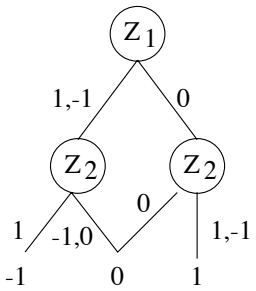


Figure 1: TDD representation of $P(Z_1, Z_2) = Z_1^2 Z_2 + Z_2^2$

2.2 Intensional transition system model

Definition 1 *An* m-dimensional intensionally Labeled Transition System (or m-iLTS) *is a structure* $T = (Q, Y, \rightarrow)$, *where* Q *is a set of states,* Y *is a set of "*m*" variables* $Y_1, ..., Y_m$, *and* $\rightarrow \subseteq Q \times \mathbb{Z}/_{p\mathbb{Z}}[Y] \times Q$. *Each transition is labeled by a polynomial over the set* Y. •

We write $q \stackrel{P(Y)}{\rightarrow} q'$ (or simply $q \stackrel{P}{\rightarrow} q'$), instead of $(q, P(Y), q') \in \rightarrow$. Then, iLTS can be understood as an "intensional" representation of classical LTSs, where the labels are tuples in $(\mathbb{Z}/_{p\mathbb{Z}})^m$: each arrow of the iLTS labeled by $P(Y)$ intensionally represents as many arrows labeled by some $y \in Sol(P(Y))$. In the sequel, we shall call $Ext(T)$ the corresponding "extensional" Labeled Transition System (LTS).

Definition 2 (Parallel composition of iLTS) *Let* $T_1 = (Q_1, Y, \rightarrow_1)$ *and* $T_2 = (Q_2, U, \rightarrow_2)$ *be two iLTSs with possible common variables between* Y *and* U. *The* parallel composition *of* T_1 *and* T_2, *written* $T_1 \mid T_2$, *is* $(Q_1 \times Q_2, Y \cup U, \rightarrow)$ *with*

$$(q_1, q_2) \stackrel{P_1(Y) \oplus P_2(U)}{\rightarrow} (q_1', q_2') \text{ whenever } \begin{cases} q_1 \stackrel{P_1(Y)}{\rightarrow_1} q_1' \text{ in } T_1 \\ q_2 \stackrel{P_2(U)}{\rightarrow_2} q_2' \text{ in } T_2 \end{cases}$$ •

Hiding events consists in abstracting from components of the label. It helps in internalizing some communications between the composed systems that are not relevant to observe in the behavior.

Definition 3 (Event hiding) *Let* $T = (Q, Y, \rightarrow)$ *be an* m*-iLTS, and* $Y_i \in Y$. *We define the* $(m-1)$*-iLTS* $(T \setminus \{Y_i\})$ *by* $(Q, Y \setminus \{Y_i\}, \rightarrow_{\setminus \{Y_i\}})$ *where* $q_1 \stackrel{\exists Y_i P}{\rightarrow} q_2$ *in* $T \setminus \{Y_i\}$ *iff* $q_1 \stackrel{P}{\rightarrow} q_2$ *in* T. *This definition can be generalized to sets* $\tilde{Y} \subseteq Y$. •

Intensional approach for labels offers a "compact" way to talk about sets of transitions in the system. However, we would like to reinforce this method in such a way that the whole system, and not only its sets of labels, can be itself described intensionally.

Definition 4 *An* (n, m)-dimensional Intensional Labeled Transition System (or ILTS) *is a structure* $S = (X, X', Y, \mathcal{T})$ *where* $X = \{X_1, ..., X_n\}$ *and* $X' = \{X_1', ..., X_n'\}$ *are two sets of* (source and target) states variables, $Y = \{Y_1, ..., Y_m\}$ *is a set of labels variables and* $\mathcal{T}(X, Y, X') \in \mathbb{Z}/_{p\mathbb{Z}}[X, Y, X']$ *is a polynomial describing the legal transitions.* •

Given some source state $x \in (\mathbb{Z}/_{p\mathbb{Z}})^n$ and some target state $x' \in (\mathbb{Z}/_{p\mathbb{Z}})^n$, the set $Sol(\mathcal{T}(x, Y, x'))$ denotes all the possible labels of transitions from state x to state x'. When states are viewed extensionally, we retrieve the iLTS of in Section 2.2, which in turn can be interpreted as a classical LTS.

Remark 1 *Actually, ILTS are also known under the name of* **(non deterministic) Polynomial Dynamical Systems** *[15] which are usually described by a set of polynomial equations (here over the field $\mathbb{Z}/_{p\mathbb{Z}}$) of the form:*

$$\begin{cases} Init(X) = 0 \\ \mathcal{T}(X, Y, X') = 0 \end{cases} \tag{4}$$

where $Init(X) = 0$ describes the set of initial states of the system. Note that an ILST can be obtain for "free" (provided we have specified the system in the high-level language, namely the SIGNAL *language [18, 5]).* ◦

Finally, Definition 2 and 3 can be rephrased at the full intensional level. Hence, the **Event hiding**, say over variable $Y_i \subseteq Y$, is obtained by considering the system

$$S \setminus \{Y_i\} \stackrel{\text{def}}{=} (X, X', Y \setminus \{Y_i\}, \exists Y_i \mathcal{T}(X, Y, X')) \tag{5}$$

and the **Parallel composition**, written $S_1 \mid S_2$ can be defined by

$$S_1 \mid S_2 \stackrel{\text{def}}{=} (X^1 \cup X^2, X'^1 \cup X'^2, Y, \mathcal{T}_1 \oplus \mathcal{T}_2) \tag{6}$$

Note that one should take care of the variable names to represent states when parallel composition is considered. In order to match the Definition 2, we always assume that the two systems to be composed have distinct state variable sets.

3 Abstraction by state fusion

This section is devoted to the computation of a reduced system according to a fusion state criterion. Fisrt, we assume that the criterion is given by some symbolic canonical surjection. In this case, reducing the system is straightforward. Next, we explore other means such as equivalence relation between states (eg. bisimulation), partitions, ...

3.1 System reduction w.r.t. a fusion criterion

Providing we are given a canonical surjection (see Definition 5) from the set of states to a smaller set of states (ie. represented with less variables), we recall how one can define the associated quotient ILTS over the latter set of variables. This can be done symbolically, as long as the surjection is itself symbolically expressed; this leads us to consider a relation rather than a mapping. Due to the aim of the Section 3, we shall use the terminology of "state fusion criterion" instead of the far too general expression "canonical surjection".

Definition 5 *We assume given an (n, m)-ILTS $S = (X, X', Y, I, \mathcal{T})$ where $X = \{X_1, ..., X_n\}$. We say that ϕ is a fusion criterion in S w.r.t. the set of "l" fresh variables $Z = \{Z_1, ..., Z_l\}$ whenever:*

1. *$l \leq n$,*
2. *$\phi \in \mathbb{Z}/_{p\mathbb{Z}}[X, Z]$, and*
3. *for all $x \in (\mathbb{Z}/_{3\mathbb{Z}})^n$, $\phi(x, Z) = 0$ has a solution and it is unique.*

By Definition 5, $Sol(\phi) \subseteq (\mathbb{Z}/_{p\mathbb{Z}^n}) \times (\mathbb{Z}/_{p\mathbb{Z}^l})$, and because of clause 3, $Sol(\phi)$ can be understood as a mapping from $(\mathbb{Z}/_{p\mathbb{Z}})^n$ to $(\mathbb{Z}/_{p\mathbb{Z}})^l$. Now, it is possible to define a transition system whose states are obtained by gluing those x's mapped onto the same element of $(\mathbb{Z}/_{p\mathbb{Z}})^l$.

Assuming a state fusion criterion ϕ exists, we can define the reduced ILTS of S w.r.t. ϕ by $S_\phi \stackrel{\text{def}}{=} (Z, Z', Y, \mathcal{T}_\phi)$, where

$$\mathcal{T}_\phi(Z, Y, Z') = \exists X \exists X' (\phi(X, Z) \oplus \mathcal{T}(X, Y, X') \oplus \phi(X', Z')) \tag{7}$$

Informally, $\mathcal{T}_\phi(z, y, z') = 0$ whenever there exists one x in the class encoded by its representative z and one x' in the class encoded by its representative z' such that $x \stackrel{y}{\rightarrow} x'$ holds in the original system S.

Note that when initial states are taken into account in the system, thus defined by some polynomial $I(X)$, the corresponding $I'(Z)$ in the reduced model can be computed by :

$$I'(Z) = \exists X \{\phi(X, Z) \oplus I(X)\}. \tag{8}$$

In general, S_ϕ will have more behaviors than S, but when the state fusion criterion ϕ is derived from a bisimulation equivalence, then S_ϕ preserves exactly the same behavioral properties than S. The next section explains how a state fusion criterion can be computed according to a given symbolic state equivalence.

3.2 System reduction modulo an equivalence relation

We assume given an (n, m)-ILTS $S = (X, X', Y, I, \mathcal{T})$ and an equivalence relation ρ over the states of $Ext(S)$, which can be symbolically represented by some $R \in \mathbb{Z}/_{p\mathbb{Z}}[X, X_d]$. Here $X_d = \{X_{d_1}, ..., X_{d_n}\}$ is a set of n variables aiming to represent copies of the variables X_i of set X: then, two states $(x, x_d) \in \rho$ whenever the expression $R(x, x_d) = 0$.

Now the natural state fusion criterion ϕ associated to ρ we would like to compute would satisfy the following:

- it is a fusion criterion,
- $(x, x_d) \in \rho$ iff the unique solution of $\phi(x, Z) = 0$ is equal to the unique solution of $\phi(x_d, Z) = 0$.

In other words, for a given x, the solution of $\phi(x, Z) = 0$ represents $[x]_R$, ie. the R-class of x.

The state fusion criterion construction:

Our construction of the state fusion criterion ϕ is not entirely made symbolically. To our knowledge, no algorithm avoiding the equivalence classes enumeration has been proposed.

Assuming the number of R-classes is k, and $p^{r-1} < k \leq p^r$ (for some $r \geq 1$), we show how to compute a TDD, say Φ, over variables $Z = \{Z_1, ..., Z_r\}$ and $X = \{X_1, ..., X_n\}$ which denotes the state fusion criterion ϕ associated to R. To do so, we directly work on the data structures, namely the p-Decision Diagrams (p-DD). However, it might be possible to adapt without difficulty this algorithm to any kind of data structures, without avoiding the enumerative feature of the method.

Intuitively, we start from the p-DD representation of relation $R(X, X_d) \subseteq (\mathbb{Z}/_{p\mathbb{Z}})^n \times (\mathbb{Z}/_{p\mathbb{Z}})^n$, with the particular reorder of the variables $X_i \prec X_{d_j}$, $\forall i, j$. Call θ this p-DD.

Property 2 *At the end of every path x (over variables X's) in θ, the remaining structure is a p-DD in variables X_d's and characterizes the R-class of x.* ◇

Therefore, a traversal of all paths x in θ leads us to compute "on the fly" the number of R-classes, namely k ; also, during this enumerative phase (in the worse case we explore the whole state space), we incrementally achieve the computation of Φ by introducing one by one the r variables Z_i when necessary.

Remark 2 *We have experimented, as expected, the fact that the reduction algorithm requirement for the state variables precedence $X_i \prec X_{d_j}$ has to be carefully implemented ; for example, implementing $\prec$ by the trivial order $X_1 \prec X_2 \prec ... \prec X_{d_1} \prec X_{d_2} \prec ...$ in the structure generally leads to a blow up. Therefore, we have considered an algorithm delivering a total order $\prec_{tot} \subseteq \prec$ based on a "blocks reordering" : $\prec_{tot}$ is a good ordering over the "variable blocks" $\{X_1, ...X_n\}$ and $\{X_{d_1}, ...X_{d_n}\}$.*

The idea of the algorithm is the following: from the root of θ (variable X or the leaf 0 if R is trivial), we recursively go down along a path until a variable X_{d_j} is reached. Call θ' the remaining sub-tdd below X_{d_j} in θ. By Property 2, θ' is an R-class. Provided we know this R-class has not been encountered yet, we attach θ to the structure Φ as follows: either an available hanging branch in Φ is available. In this case θ' is plugged at this available place. Otherwise, we introduce a fresh variable Z_i at the top of Φ and wait for the complete p-DD in the variables $\{Z_1, ..., Z_{i-1}\}$ containg θ' to be achieved, then plug it as a second son of Z_i.

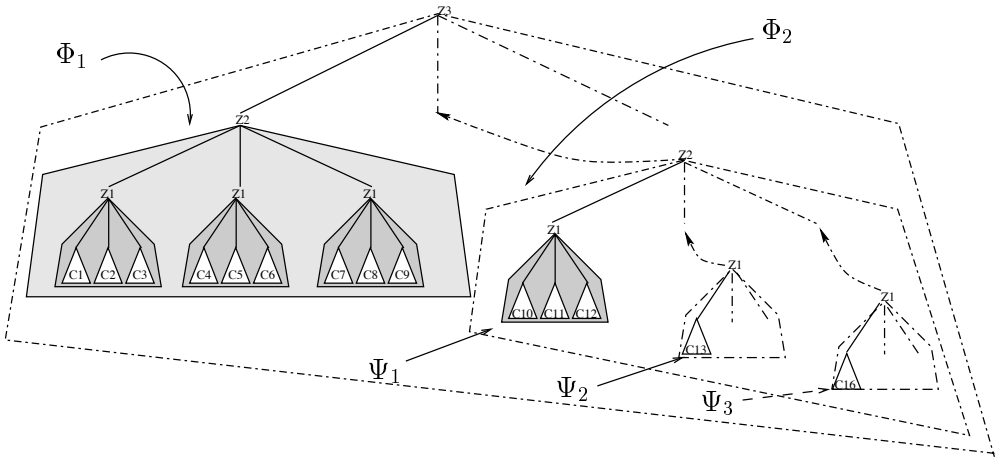


Figure 2: Construction of $\phi(X, Z)$

The reader can refer to Figure 2, for the case $p = 3$. We shall say "TDD" (for Ternary Decision Diagram) instead of 3-DD. In this example, already built TDDs are drawn as triangles with solid bold lines, whereas not already built ones are drawn in dashed bold lines.

Assume we already have completed a TDD containing already treated classes, say from C_1 upto C_9 ; call Φ_1 this structure. Suppose now that a new class C_{10} is encountered. Then a new variable Z_3 needs beeing introduced at the top Φ_1 and awaits for other sub-structures to be complete before plugging them underneath. For example, after structures Ψ_1 (containing C_{10} but also C_{11} and C_{12}), Ψ_2 and Ψ_3 have been completed, then Φ_2 can be completed and then plugged under Z_3.

The final stage of the algorithm is to run a completion for the remaining non allocated branches (which exists when the number of classes k is not of the form p^r) by pointing them to the leaf **1**. In our example, suppose there is only 16 R-classes, i.e. the last class is C_{16}, then the remaining hanging branches of structure Ψ_3 will point to leaf **1** as show in Figure 3.

Remark 3 *We can also handle relations $S(X_1, X_2)$ that are not equivalence relations (e.g. not reflexive, not symmetric, ...) by closing S such that it becomes an equivalence (using a classical fix point computation in the same spirit as the one described in (10)). Next algorithms described below can be applied.* ∘

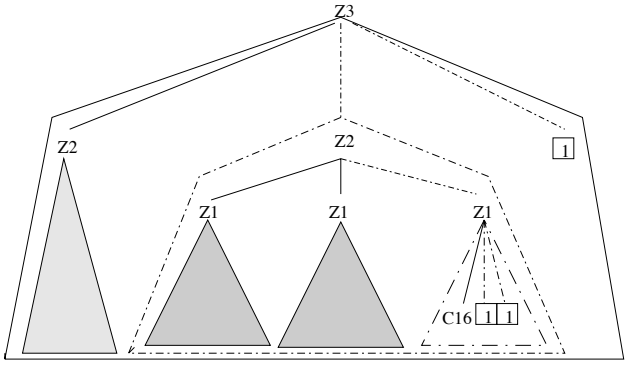


Figure 3: Final stage of the construction

3.3 Particular cases of equivalences: some bisimulations

Bisimulation relations [27, 28, 30, 31] are used to compare *labeled transition systems* (LTS) from the behavioral point of view: a bisimulation equivalence between states of the systems enables to perform interesting state fusion abstractions, in terms of the properties that are preserved.

3.3.1 The strong bisimulation case

As we aim to develop symbolic methods, we first explain how the classical strong bisimulation [28, 27] can be handled symbolically. The definition is inspired from De Simone's symbolic bisimulation over reactive automata [7]. We first define the *symbolic bisimulation* over iLTS. We then give a symbolic algorithm in the ILTS framework.

Definition 6 (The symbolic bisimulation) *Let $T_1 = (Q_1, Y, \mathcal{I}_1, \rightarrow_1)$ and $T_2 = (Q_2, Y, \mathcal{I}_2, \rightarrow_2)$ be two iLTSs. A symbolic bisimulation of T_1 and T_2 is a binary relation $\mathcal{R} \subseteq Q_1 \times Q_2$ s.t. $q_1 \mathcal{R} q_2$ whenever*

1. *for all $q_1 \stackrel{P}{\rightarrow}_1 q_1'$ there exists a finite set of transitions $(q_2 \stackrel{P_i}{\rightarrow}_2 q_2^i)_{i \in I}$ with*
   - *(a) $(P \Rightarrow \Pi_{i \in I} P_i) \equiv 0$, and*
   - *(b) $q_1' \mathcal{R} q_2^i$, for all $i \in I$, and*
2. *vice versa.* •

Definition 6 can be intuitively explained as follows: this equivalence aims to capture exactly classic strong bisimulation [28, 27] over the underlying explicit models. However, it takes advantage of the intensional feature by imposing that, for the first system, each outgoing transition of a given set (represented intensionally by P) can be mimicked in the second system, i.e. belongs to at least one outgoing "set" of transitions (one of the $Sol(P_i)$'s). Point (a) of Definition 6 can be rephrased as $Sol(P) \subseteq \bigcup_i Sol(P_i)$ (see Property 1), then retrieving the original definition of De Simone [7].

Theorem 1 [12] (Compositionality) *Symbolic bisimulation is a congruence w.r.t. parallel composition and events hiding.* ◇

Since bisimulations are closed under arbitrary unions, we can talk about the greatest bisimulation, written $\stackrel{\sim}{=}$ in the following, and say that "q_1 and q_2 are bisimilar" whenever $q_1 \stackrel{\sim}{=} q_2$.

From the expressiveness point of view, symbolic bisimulation is an alternative view of strong bisimulation (see Definition 7) when intensional models are considered (see Theorem 2). Also, symbolic bisimulation over iLTS can be expressed in terms of nested "projective" equivalences in the same spirit of [26, 27].

Definition 7 [28, 27] *Given two LTSs* $t_1 = (Q_1, \Sigma, \mathcal{I}_1, \rightarrow_1)$ *and* $t_2 = (Q_2, \Sigma, \mathcal{I}_2, \rightarrow_2)$*, a* **strong bisimulation** *of* t_1 *and* t_2 *is a binary relation* $\rho \subseteq Q_1 \times Q_2$ *s.t.* $(q_1, q_2) \in \rho$ *whenever*

1. *for all* $\sigma \in \Sigma$*, for all transition* $q_1 \xrightarrow{\sigma}_1 q_1'$ *there exists a state* q_2' *s.t.* $q_2 \xrightarrow{\sigma}_2 q_2'$ *and* $(q_1', q_2') \in \rho$*.*
2. *vice-versa* •

Definition 8 (Symbolic projective equivalences) *Let* $T_1 = (Q_1, Y, \rightarrow_1)$ *and* $T_2 = (Q_2, Y, \rightarrow_2)$ *be two iLTSs. We define a family of relations* $\precsim_i \subseteq Q_1 \times Q_2$ *by induction over* $j \in \mathbb{N}$*:*

- $\precsim_0 \stackrel{def}{=} Q_1 \times Q_2$,
- $q_1 \precsim_{j+1} q_2$ *iff*
  1. *for all* $q_1 \xrightarrow{P} q_1'$*, there exists a finite set of transitions* $(q_2 \xrightarrow{P_i}_2 q_2^i)_{i \in I}$ *with*
     (a) $(P \Rightarrow \Pi_{i \in I} P_i) \equiv 0$*, and*
     (b) $q_1' \precsim_j q_2^i$*, for all* $i \in I$
  2. *and reciprocally.* •

We can now state the following theorem:

Theorem 2 (Expressiveness)

1. *[12] Let* T_1 *and* T_2 *be two iLTSs. Then there exists a symbolic bisimulation between* T_1 *and* T_2 *iff there exists a strong bisimulation between* $Ext(T_1)$ *and* $Ext(T_2)$*.*
2. *Let* T_1 *and* T_2 *be two iLTS. Then for all* $q_1 \in Q_1$ *and* $q_2 \in Q_2$*,* $q_1 \precsim q_2$ *if and only if* $q_1 \bigcap_{n \in \mathbb{N}} \precsim_n q_2$

Proof : The proof of *(1)* can be found in [12]. Point *(2)* is due to the fact that symbolic bisimulation over finite iLTS (therefore finitely branching) is the limit of nested projective equivalences, in the same spirit of [26] for strong bisimulation. We omit the proof. ◇

Theorem 2 gives an iterative algorithm to compute symbolic bisimulation. Assume given two ILTSs $S_U = (U, U', Y, I_U, \mathcal{T}_U)$ and $S_V = (V, V', Y, I_V, \mathcal{T}_V)$. Algorithm 1 computes the greatest symbolic bisimulation of S_U and S_V.

Algorithm 1

1. *Define the polynomial* $\mathcal{R}_0(U, V) \stackrel{def}{=} 0$*.*
2. *Compute iteratively until stabilization* $(\mathcal{R}_j(U, V))_j$ *defined by:*

$$\mathcal{R}_{j+1}(U, V) \text{ is the canonical generator of the } \equiv\text{-class of } \left\{ \begin{array}{l} \mathcal{R}_j(U, V) \\ \oplus\ \forall U' \forall Y [(\mathcal{T}_U(U, Y, U') \Rightarrow \exists V' (\mathcal{T}_V(V, Y, V') \oplus \mathcal{R}_j(U', V'))] \\ \oplus\ \forall V' \forall Y [\mathcal{T}_V(V, Y, V') \Rightarrow \exists U' (\mathcal{T}_U(U, Y, U') \oplus \mathcal{R}_j(U', V'))] \end{array} \right.$$

3. *Call* $\mathcal{R}(U, V)$ *the result.*

Proposition 2 *For all* $j \in \mathbb{N}$*,* $R_j(u, v) = 0$ *if and only if* $u \precsim_j v$*.*

Proof : $\Rightarrow$) We use an inductive reasoning over j. The propsition clearly holds for the case $j = 0$. Now, assume u and v are s.t. $R_{j+1}(u, v) = 0$ and let $u \xrightarrow{P} u'$ be a (symbolic) transition in S_U. Clearly $P(Y) \equiv \mathcal{T}_U(u, Y, u')$. The polynomial $Q(Y) \stackrel{\text{def}}{=} \exists v' \mathcal{T}_V(v, Y, v') \oplus R_j(u', v')$ which is computed in the

algorithm exactly captures the set $\{y|\exists v \stackrel{P_i}{\rightarrow} v_i^2$ with $P_i(y)=0$ and additionally $u' \underline{\succeq}_j v_i^2\}$. By definition of R_{j+1}, the y's in $Sol(\mathcal{T}_U(u,Y,u'))$, i.e. in $Sol(P(Y))$, are also requested to be solutions of $\bigcup_i Sol(P_i)$, which entails $u \underline{\succeq}_{j+1} v$.

$\Leftarrow$) We can use again an induction over j and an argument similar to the proof of Theorem 2 1. We omit it. ◇

Theorem 3 *Algorithm 1 terminates and at the end, $R(u,v)=0$ if and only if $u \underline{\succeq} v$.*

Proof : Termination is guaranteed by the fact that relations R_j are finite and nested. The second statement is a corollary of Proposition 2 and Theorem 2. ◇

3.3.2 The τ-bisimulation case

In general, the behavior of a system is defined according to an observability criterion of its moves. Classically, the alphabet of actions is enriched with a new element, often written τ, to represent all invisible transitions and to consider appropriate bisimulations in this framework, such as τ-bisimulation (also known as the *observational equivalence* or *weak bisimulation*) [27, 34], but also variants like "delay" or "branching" bisimulations [31, 33].

Assume given an alphabet of actions $\Sigma_\tau = \Sigma \cup \{\tau\}$, with $\tau \notin \Sigma$. Symbol μ will denote a typical element of Σ_τ, whereas symbols a, b will be kept for elements of Σ. Let $t=(Q, \Sigma_\tau, \mathcal{I}, \rightarrow)$ be a LTS over Σ_τ. We introduce the *observational* transition relation $\Longrightarrow \subseteq Q \times \Sigma_\tau \times Q$ defined by $q \stackrel{\mu}{\Longrightarrow} q'$ if:

case $\mu \neq \tau$: there exists a sequence of the form $q = q_0 \stackrel{\tau}{\rightarrow} q_1 \stackrel{\tau}{\rightarrow} .. \stackrel{\tau}{\rightarrow} q_k \stackrel{\mu}{\rightarrow} q_{k+1} \stackrel{\tau}{\rightarrow} .. \stackrel{\tau}{\rightarrow} q_n = q'$.

case $\mu = \tau$: there exists a sequence of the form $q = q_0 \stackrel{\tau}{\rightarrow} q_1 \stackrel{\tau}{\rightarrow} .. \stackrel{\tau}{\rightarrow} q_k$ with $k \geq 0$.

Let us now define the τ-bisimulation. It only refers to observable moves of the systems. In the extensional framework, it is defined by:

Definition 9 (τ-bisimulation) [25, 27] *A τ-bisimulation between two LTSs $t_1 = (Q_1, \Sigma_\tau \mathcal{I}_1, \rightarrow_1)$ and $t_2 = (Q_2, \Sigma_\tau, \mathcal{I}_2, \rightarrow_2)$ is a binary relation $\mathcal{R} \subseteq Q_1 \times Q_2$ s.t. $(q_1, q_2) \in \mathcal{R}$ (or equivalently written $q_1 \mathcal{R} q_2$) whenever*

1. *for all $\mu \in \Sigma_\tau$, for all $q_1' \in Q_1, q_1 \stackrel{\mu}{\Longrightarrow} q_1'$, there exist a state $q_2' \in \mathcal{X}_2$ s.t. $q_2 \stackrel{\mu}{\Longrightarrow} q_2'$ and $q_1' \mathcal{R} q_2'$.*
2. *vice-versa* •

The following property makes the links between the τ-bisimulation and the strong bisimulation:

Property 3 *The τ-bisimulation and the strong bisimulation definitions are likewise but the former exploits relation $\Longrightarrow$ instead of $\rightarrow$.* ◇

Let $S=(X, X', Y, \mathcal{T})$ be an (n,m)-ILTS. In order to characterize unobservable transitions in S we assume given a particular polynomial, written $P_\tau(Y) \in \mathbb{Z}/_{p\mathbb{Z}}[Y]$. Therefore, $Ext(S)$ is a LTS over alphabet of visible actions $(\mathbb{Z}/_{p\mathbb{Z}})^m \setminus Sol(P_\tau)$ which transitions $T(x,y,x')=0$ mean $x \stackrel{y}{\rightarrow} x'$ whenever $P_\tau(y) \neq 0$, $x \stackrel{\tau}{\rightarrow} x'$ otherwise. In other words, silent moves in $Ext(S)$, i.e. states x and x' s.t. $x \stackrel{\tau}{\rightarrow} x'$ are characterized by the polynomial:

$$\mathcal{T}_\tau(X, X') \stackrel{\text{def}}{=} \exists Y \mathcal{T}(X, Y, X') \oplus P_\tau(Y). \tag{9}$$

The polynomial $\mathcal{T}_\tau^* \in \mathbb{Z}/_{p\mathbb{Z}}[X, X']$ representing the reflexive and transitive closure of the relation $\mathcal{T}_\tau$ can be computed using a standard iterative algorithm on the basis of the following:

$$\begin{cases} \mathcal{T}_\tau^0(X,X') = \mathcal{T}_\tau(X,X') * Id(X,X') \\ \mathcal{T}_\tau^{i+1}(X,X') \;=\; \exists X''[\mathcal{T}_\tau^i(X,X'') \oplus \mathcal{T}_\tau^i(X'',X')] \end{cases} \tag{10}$$

where $Id(X, X')$ is the canonical generator of the identity relation $[\bigoplus_{i=1}^{n}(X_i - X_i')]_{\equiv}$. The algorithm trivialy terminates (as the computation denotes a sequence of finite nested sets).

We can now express the symbolic representation for $\Longrightarrow$:

$$\mathcal{R}_\tau(X,Y,X') \stackrel{\text{def}}{=} (P_\tau(Y) \oplus \mathcal{T}_\tau^*(X,X')) * (\overline{P_\tau}(Y) \oplus \exists Z \exists Z' \{\mathcal{T}_\tau^*(X,Z) \oplus \mathcal{T}(Z,Y,Z') \oplus \mathcal{T}_\tau^*(Z',X')\}. \quad (11)$$

Proposition 3 *$x \stackrel{y}{\Longrightarrow} x'$ in $Ext(S)$ if and only if $\mathcal{R}_\tau(x,y,x') = 0$* $\diamond$

Given two ILTSs $S_U = (U, U', Y, I_U, \mathcal{T}_U)$ and $S_V = (V, V', Y, I_V, \mathcal{T}_V)$, the computation of the greatest τ-bisimulation between S_U and S_V reduces to the computation of the greatest strong bisimulation between $S_U = (U, U', Y, I_U, \mathcal{R}_{\tau_U})$ and $S_V = (V, V', Y, I_V, \mathcal{R}_{\tau_V})$ (by Property 3, Proposition 3 and Theorem 2).

3.3.3 Other kinds of bisimulations with τ

In the littterature, other kinds of bisimulations abstracting from τ's have been proposed. Among them, the *branching bisimulation* [32] enables to capture behaviors that abstract from τ loops and which turns out to be a congruence w.r.t. action refinement.

Definition 10 [32, 33] (Branching bisimulation) *Given two LTSs $t_1 = (Q_1, \Sigma_\tau, \rightarrow_1)$ and $t_2 = (Q_2, \Sigma_\tau, \rightarrow_2)$, a branching bisimulation between t_1 et t_2 is a binary relation $\rho \subseteq Q_1 \times Q_2$ such that $(q_1, q_2) \in \rho$ whenever*

1. *For all $q_1 \stackrel{\mu}{\rightarrow} q_1'$, if $\mu = \tau$ then $(q_1', q_2) \in \rho$, otherwise there exists $q_2', q_2'' \in Q_2$ s.t.*
   - *(a) $q_2 \stackrel{\tau}{\Longrightarrow} q_2''$, $q_2'' \stackrel{\mu}{\rightarrow} q_2'$, and*
   - *(b) $(q_1, q_2'') \in \rho$, and $(q_1', q_2') \in \rho$*
2. *vice-versa* •

Here follows the algorithm delivering the greatest branching bisimulation of two ILTSs $S_U = (U, U', Y, \mathcal{T}_U)$ and $S_V = (V, V', Y, \mathcal{T}_V)$.

Algorithm 2

1. *Define the polynomials $R_0(U,V) = 0$.*
2. *Compute $\mathcal{T}_{U_\tau}^*$ (resp. $\mathcal{T}_{V_\tau}^*$) of S_U (resp. S_V) as in previous section.*
3. *Compute iteratively until stabilization the sequence $(R_k(U,V))_k$ defined by:*

   $R_{k+1}(U,V)$ is the canonical generator of the $\equiv$-class of

$$\left\{\begin{array}{l} R_k(U,V) \\ \oplus \forall U' \forall Y [\mathcal{T}_U(U,Y,U') \Rightarrow \{(P_\tau(Y) \oplus R_k(U',V)) * \\ \qquad\qquad (\overline{P_\tau}(Y) \oplus \exists V' \exists V" (\mathcal{T}_{V_\tau}^*(V,V") \oplus \mathcal{T}_V(V",Y,V') \\ \qquad\qquad\qquad \oplus R_k(U,V") \oplus R_k(U',V')))\}] \\ \oplus \forall V' \forall Y [\mathcal{T}_V(V,Y,V') \Rightarrow \{(P_\tau(Y) \oplus R_k(V',U)) * \\ \qquad\qquad (\overline{P_\tau}(Y) \oplus \exists U' \exists U" (\mathcal{T}_{U_\tau}^*(U,U") \oplus \mathcal{T}_U(U",Y,U') \\ \qquad\qquad\qquad \oplus R_k(V,U") \oplus R_k(V',U')))\}] \end{array}\right.$$

   Call $R(U,V)$ the result.

The algorithm above should not fear the reader as it is an immediate translation of Definition 10, but in a symbolic style!

Also any other variant of bisimulation with τ can be proposed in the same spirit. We do not give here an exhaustive list, but refer to [33] for a complete classification.

3.4 Other kinds of state fusion criterion

In this section, we explore other means to express the state fusion criterion. For instance, it can be directly derived from either a partition, or a set of atomic propositions or any equivalence relation. In these cases, the abstract model computation could be simplified. Nevertheless, we insist in saying that it is up to the user to take care of the correction of the abstraction according to the kind of properties he aims to consider.

3.4.1 State Fusion induced by a partition

Let $[P_1, ...P_k]$ be a set of polynomials in $A[X_1, ...X_n]$ such that:

$$Sol(P_i) \cap Sol(P_j) = \emptyset \text{ and } \bigcup_{i \in [1..k]} Sol(P_i) = (\mathbb{Z}/_{p\mathbb{Z}})^n. \tag{12}$$

This set of polynomials constitute a partition of the state space of the system. The idea is then to perform a state fusion according to this partition. Compare to Section 3.2, we somehow already have the equivalence classes. Thus, we have neither to build the equivalence relation $R(X_1, X_2)$, nor to extract the equivalence classes from this relation. Moreover we do not have to enforce a particular order on the variables[2]. As k the number of classes is already known, the number of state variables Z_i we have to introduce in order to obtain the encoding function $\phi(Z, X)$ is statically determined. We then just have to read the set of polynomials (each of them representing a different class) and plug this class in the p-DD ϕ as illustrated in Section 3.2.

Example 1 *Assume we want to perform the fusion of all the states that have the same value for variable X_i (such a variable could represent a bounded counter). Computing the set of polynomials $P_j(X) = (X_i - j)^{p-1}$, $j = [0..p-1]$, we obtain a partition of the state space. In this particular case, the corresponding p-DDs only have 3 nodes. By computing ϕ and the reduced model according to (7), we represent by state z of the reduced model all states x's agreeing on variable X_i.* ◦

From a practical point of view, the user might rather define the partition according to a set of l atomic propositions $[Q_1, ...Q_l]$ that states have to agree on (see example above). Each element of the partition, say α_{IJ} can be computed as : for $I \cup J = [1..l]$, $I \cap J = \emptyset$,

$$\alpha_{IJ}(X) = \bigoplus_{i \in I} Q_i(X) \oplus \bigoplus_{j \in J} (1 - Q_j^{p-1}(X)) \tag{13}$$

Once obtained the set of polynomials α_{IJ}, we fall into the partition case, as explained previously. Finally, as 2^l corresponds to the maximal number of equivalent classes, we will need at most $\lceil l * log_p(2) \rceil$ state variables Z_i to define the abstract model.

3.4.2 Improvement of state fusion abstraction

Because in our systems initial states are considered (with polynomial $I(X)$), we have refined the approach by considering only reachable R-classes, ie. containing reachable states. Such a refinement makes sens only if the set of reachable states (or the orbit) has already been computed. In this case, a class is taken into account only if it contains a reachable state, with minor changes in the algorithms.

[2] This part constitutes a bottle neck (in terms of memory, computation time,etc) of the algorithm presented in Section 3.2

Moreover, taking into account this information leads to a simple method for minimizing the number of state variables of the model: indeed, a given ILST $S = (X, X', Y, I, \mathcal{T})$, $\mid X \mid = n$ might have in theory p^n reachable states. However, most of the time, the cardinal of the orbit, say m, is far smaller than p^n, especially when this system is obtained by synchronized product of sub-systems, even if these sub-systems have themselves already been abstracted. Therefore, it is relevant to try to decrease the number of state variables that are needed to encode the global system.

The state variable set minimization is performed by considering the identity relation: $Id(X_1, X_2) = \bigoplus_{i=1}^{n}(X_{1_i} - X_{2_i})$. The algorithm to compute the fusion criterion (as in Section 3.2) with additional reachability condition delivers a "reduced" model S' with less state variables, namelly at most $log_p(m)$. Of course, reachable graphs of $Ext(S')$ and $Ext(S)$ are isomorphic.

4 Abstraction by Restriction

This abstraction aims to simplify the model by disallowing some behaviors. A naive approach would consist in modifying the structure of the automata by removing either a set of events or a set of states.

In our framework, abstraction by restriction w.r.t. a set of events, is performed on the basis of a polynomial $A(Y)$, which denotes the events y that are kept in the desired behavior (i.e. all transitions labeled by the others are then removed). The abstracted system is simply obtained by intersecting the transition relation T with the polynomial A:

$$T'(X, Y, X') = T(X, Y, X') \oplus A(Y). \tag{14}$$

In the case of an abstraction by restriction w.r.t. a set of states $O(X)$, which can possibly be deduced from a property (e.g. invariant property,...), the idea is to intersect the transition relation of the ILTS with this polynomial in order to consider only the states that satisfy O[3]. The new transition relation of the system is then inductively given by:

$$\begin{cases} I_0(X) &= I \oplus O(X) \\ T'(X, Y, X') &= T(X, Y, X') \oplus O(X') \end{cases} \tag{15}$$

Also, more general systems and restriction specifications/objectives can be considered: for example, observing automata can be composed with the original one in order to get more subtle restrictions of the behaviors. Both techniques have been considered, and for the latter one, we have explored particular kind of restrictions where the status of events is twofold : some events, called *uncontrollable*, cannot be prevented from occurring (e.g. think of an open system in an environment), whereas the others, though *controllable* can be disabled. When we restrict the behavior of the system to a given set of states, we are possibly disabling some events that are not controllable (situation that is not allowed in this new framework). Hence, we propose a way to refine the abstraction by restriction which disables only controllable events. Also, some minimality features of the restriction will be taken into account : actually, the abstracted model will have a smaller behavior than the original one - to fulfill the restriction objectives - but as large as possible, otherwise, the abstracted model "doing nothing" would correspond most of the time.

The adopted principle is to over-constrain the initial system - by additional (boolean/polynomial) equations, called *the controller*. The resulting model, called *restricted/controlled system*, can be understood as a parallel composition of the original system and the controller. The "restriction synthesis" process, we describe now, is borrowed from the controller synthesis methodology as carried out in [8, 23].

[3] If the interest is to forbid some set of states $F(X)$, $O(X)$ will be equal to $\overline{F}(X)$

4.1 The restriction synthesis achievement

The starting point is the following: given a model for the system and the restriction objectives, a "controller" must be derived by various means such that the resulting behavior of the closed loop system meets the restriction objectives. The specification of the system is represented by an ILTS while the restrictions applied to the system are achieved by incorporating new algebraic equations in the original ILTS.

The model: From now on, an ILTS will be given by

$$S = (X, X', Y, K, \mathcal{T}(X, Y, K, X'), I) \tag{16}$$

where X and X' represents the state variables; Y and K are respectively the sets of *uncontrollable* and *controllable* event variables. Each event (y, k) contains an uncontrollable component y and a controllable one k. To distinguish the two components, the vector y is called an *event* and the vector k a *control*. We have no direct influence on the y part which depends only on the state x, but we can observe it. On the other hand, we have full control over k and we can choose any value of k provided it is admissible : k is *admissible* at (x, y) whenever $(\exists X' T)(x, y, k) = 0$.

Let us also introduce the polynomial Q defined over the variables X, Y, K by:

$$Q(X, Y, K) = \exists X' T(X, Y, K, X'). \tag{17}$$

This polynomial will be called the *constraint equation of* S. Intuitively, it denotes the set of events (y, k) that are possible in a particular states x.

The restriction policy: The behavior of an ILST S can then be restricted by first restricting initial states and then by choosing suitable values for $k_1, k_2, \ldots, k_n, \ldots$ to restrict the executions. Different strategies can be chosen to determine the value of the controls k_i's. We will here consider restriction policies where the value of the control k is statically computed from the value of x and y. Such a controller is called a *static controller*. It is of the form:

$$\begin{cases} C_0(X) = 0 \\ C(X, Y, K) = 0 \end{cases} \tag{18}$$

where the equation $C_0(X) = 0$ determines the restricted initial states and $C(X, Y, K)$ describes how to choose the controls.

The restricted system is then simply given by

$$S_R = (X, X', Y, K, \mathcal{T} \oplus C, I \oplus C_0). \tag{19}$$

When the restricted system is in state x, and when an event y occurs, any value k such that $Q(x, y, k) = 0$ and $C(x, y, k) = 0$ can be chosen; i.e. given a state x and an event y, choosing a k such that $C(x, y, k) = 0$ implies an evolution of the state in accordance with the restriction objective.

However, not every controller (C, C_0) is acceptable : first, the restricted system S_R must have initial states ; thus, the equations $I(X) = 0$ and $C_0(X) = 0$ must have common solutions. Furthermore, events y in S must have their counterpart in S_R. Hence the following definition of an *acceptable controller*:

Definition 11 *An acceptable controller for S is a controller as in (19) satisfying:*

1. *$Sol(I) \cap Sol(C_0) \neq \emptyset$, and*
2. *for each reachable state x in S_R, any event y that can occur in x for S can also occur in S_R.* •

4.2 Some examples of restriction objectives

Here is the a non-exhaustive list of the various restriction objectives (as well as their definitions) for which we are able to synthesize a controller [8, 23, 24]:

- **The invariance of a set of states.** A set of states E is *invariant* for an ILTS S if every trajectory initialized in x remains in E.
- **The (global) reachability of a set of states.** A set E is *(globally) reachable* in S , if starting from any possible state, there exists a trajectory that reaches E.
- **The attractivity of a set of states from another set of states.** Let E and F be two set of states. Then, F is *attractive* for E if every trajectory initialized in E reaches F.
- **The persistence of a set of states.** A set of states E is persistent if it is attractive from the initial states and if E is invariant.
- **The recurrence of a set of states.** A set of states E is recurrent if it is visited infinitely often.
- The invariance of a set of states + one of the above control objectives.

Finally note that some other restriction objectives, dealing with *quantitative criterions* can also be considered. In general, these restriction objectives are expressed as partial order relations. Such relations can, for example, be described by means of numerical cost functions (see [22, 21]).

Let us first introduce technical material, namelly (more or less) classical state predicate transformers that will be use to achieve restriction synthesis.

- The operator $\widetilde{\mathbf{pre}}$ defined by :

$$\widetilde{pre}\ (F) = \{x | \forall y, (\exists K Q)(x,y) = 0 \Rightarrow \exists k, Q(x,y,k) = 0 \text{ and } \{x' |\ T(x,y,k,x') = 0\} \subseteq F\} \quad (20)$$

  It is the set of states for which whatever event occurs, there exists a control which "forces" the system to evolve in F. In terms of polynomials, we obtain:

$$\widetilde{pre}\ (g_F) = \forall Y \{(\exists K Q)(X,Y) \Rightarrow \exists K \exists X' (T(X,Y,K,X') \oplus g_F(X'))\} \quad (21)$$

- The operator **pre** defined by :

$$pre(F) = \{x | \exists y, \exists k, \exists x', T(x,y,k,x') \in F\} \quad (22)$$

  A state x belongs to $pre(F)$ whenever it has a successor which belongs to F. Its polynomial version is given by:

$$pre(g_F) = \exists Y \exists K \exists X' T(X,Y,K,X') \oplus g_F(X') \quad (23)$$

The rest of the section explains in detail how reachability/safety restriction objectives are treated. Proofs and details could be found in [8, 24, 23][4]

[4] In [8, 24, 23], the ILST to be controlled is supposed to be deterministic (see Remark 1). However, the generalization to non-deterministic ILTS is not difficult.

Restriction of S to a set of states E. Let g_E be the principal generator of the set of states E. Consider now the following sequence of polynomials:

$$\left\{ \begin{array}{lcl} g_0 & = & g_E \\ g_{i+1} & = & g_i \oplus \overset{\approx}{pre}(g_i) \end{array} \right. \tag{24}$$

This sequence of polynomials $(g_i)_{(i\in I)}$ is converging whatever g_E is: indeed, each g_i denotes a finite set, and the sequence decreases. The limit of this fix-point computation is either the constant polynomial 1 and this restriction problem has no solution, or some polynomial g, with $Sol(g) \neq \emptyset$. In this case, we can state:

Theorem 4 [23] *Let $S = (X, X', Y, K, \mathcal{T}, I)$ be an ILTS and E (or equivalently g_E) be a set of states. With the preceding notations, consider the systems of equations (where g is obtained by iterating (24)):*

$$\left\{ \begin{array}{lcl} C_0(X) & = & g(X) \\ C(X,Y,K) & = & \forall X'\{(T(X,Y,K,X') \Rightarrow g(X')\} \end{array} \right. \tag{25}$$

If $C_0 \oplus I \not\equiv 1$, then (C_0, C) is acceptable for S and all reachable states of $S_R = (X, X', Y, K, \mathcal{T} \oplus C, I \oplus C_0)$ belong to E. Otherwise this restriction problem has no solution. ◇

Reachability of a set of states E. Consider the following sequence of polynomials:

$$\left\{ \begin{array}{lcl} g_0 & = & g_E \\ g_{i+1} & = & g_i * \overset{\approx}{pre}(g_i) \end{array} \right. \tag{26}$$

This sequence converges since increasing and bounded by the state space, which is finite.

Theorem 5 [23] *Let $S = (X, X', Y, K, \mathcal{T}, I)$ be an ILTS and E (or equivalently g_E) be a set of states. Consider the system of equations as defined by (25) (where g is obtained by iterating (26)). If $C_0 \oplus I \not\equiv 1$, then (C_0, C) is acceptable for S and ensures the (global) reachability of E in $S_R = (X, X', Y, K, \mathcal{T} \oplus C, I \oplus C_0)$. Otherwise this restriction problem has no solution.* ◇

Reachability of a set of states E (g_E) and invariance of F (g_F). For all set of states V, let us denote $Reach(V, E)$ the set of states which are origin of a trajectory leading to E and containing only states of V. Let g_V be the principal generator of V. Then the polynomial characterization of $Reach(V, E)$, written $Reach(g_V, g_E)$, is obtained as the result of the following fix-point computation.

$$\left\{ \begin{array}{lcl} h_0 & = & g_E \oplus g_V \\ h_{i+1} & = & h_i * (g_V \oplus pre(h_i)) \end{array} \right. \tag{27}$$

This (increasing and bounded) sequence converges whatever the set V is. Consider now the following sequence of polynomials:

$$\left\{ \begin{array}{lcl} g_0 & = & g_F \\ g_{i+1} & = & Reach(g_i \oplus (\overset{\approx}{pre}(g_i), g_E)) \end{array} \right. \tag{28}$$

Theorem 6 [23] *Let $S = (X, X', Y, K, \mathcal{T}, I)$ be an ILTS and E, F (or equivalently g_E, g_F) be two sets of states. Consider the systems of equations as defined by (25) (where g is obtained by iterating (28). If $C_0 \oplus I \not\equiv 1$, then (C_0, C) is acceptable and ensures both the (global) reachability of E and the invariance of F in $S_R = (X, X', Y, K, \mathcal{T} \oplus C, I \oplus C_0)$. Otherwise this restriction problem has no solution.* ◇

5 Supervisory Control Problem using Bisimulation Techniques

In the specifications of real-time reactive systems[4] area, many applications require high reliability and safety. Traditionally, these requirements are checked *a posteriori* using simulation techniques and/or property verification. Control theory of discrete event systems allows to use constructive methods, that ensure, *a priori*, required properties on the system behavior. In this approach, the validation phase is reduced to properties not guaranteed by the programming process. There exist different theories for the control of discrete event systems since the 80's [29, 11]. Usually, the starting point of these theories is: given a model for the system and the control objectives, a controller must be derived by various means such that the resulting behavior of the closed loop system meets the control objectives.

In the Ramadge and Wonham theory of discrete events systems (DES) [29], the behaviors of discrete event systems are modeled by infinite sequences of events over a finite alphabet that represents all the possible actions of a system. The plant is represented by some kind of automaton, a labeled transition system (LTS), that generates sequences of events (or actions) through its execution. The control of the plant is then performed by inhibiting some events in Σ_c the set of controllable events as opposed to the set Σ_{uc} (uc stands for uncontrollable) of events that cannot be prevented from occurring.

In [3], G. Barret and S. Lafortune solved the Supervisory Control Problem (SCP) by exploiting the relation between a given bisimulation and the controllability of a language. [3] provides algorithms allowing the automatic synthesis of controllers according to the plant and their specification based on the computation of a greatest bisimulation of the plant and the specification. However, the implementation of these algorithms is explicit. It makes the computation of the supervisory controllers not practical because of the size of the states space which is often too important when dealing with realistic applications (even if the proposed algorithms are polynomial in the number of states). *A contrario*, the proposed approach provides symbolic computations, based on an implicit representation of the plant, encoded by polynomials and at a lower level by p-nary decision diagrams (p-DD), a slight extension of the Binary Decision Diagrams (BDD) [6].

5.1 The Supervisory Control Problem

Assuming that the plant to be controlled is modeled as a Labeled Transition System (LTS) that is a 4-tuple $G = \langle \mathcal{X}, \Sigma, x_0, \delta \rangle$, where $\mathcal{X}$ is the (finite) set of states, Σ is the set of events, $x_0 \in \mathcal{X}$ is the initial state, and δ is the partial transition function from $\mathcal{X} \times \Sigma$ to $\mathcal{X}$. In the sequel, we simply write $x \in G$ instead of $x \in \mathcal{X}$.

The behavior of the system is denoted by the prefix-closed language $\mathcal{L}(G)$ [29], generated by G[5]. Some of the events in Σ are uncontrollable, i.e., their occurrence cannot be prevented by a controller, while the others are controllable. In this regard, Σ is partitioned as $\Sigma = \Sigma_c \cup \Sigma_{uc}$, where Σ_c represents the set of controllable events and Σ_{uc} the set of uncontrollable events.

5.1.1 Basic Results

We now present some preliminaries for the Supervisory Control Problem that can be found in [29]. We first recall the definition of controllability :

Definition 12 *Let H and G be two LTSs, s.t. $\mathcal{L}(H) \subseteq \mathcal{L}(G)$. $\mathcal{L}(H)$ is said to be* controllable *w.r.t. $\mathcal{L}(G)$ and Σ_{uc} if $\mathcal{L}(H)\Sigma_{uc} \cap \mathcal{L}(G) \subseteq \mathcal{L}(H)$.* •

[5] In the following, all the languages will be assumed to be prefix-closed.

In other words, $\mathcal{L}(H)$ is conditionally invariant w.r.t. $\mathcal{L}(G)$ under events of Σ_{uc} [29].

In the Ramadge and Wonham framework, a supervisor is given by a function $S : \mathcal{L}(G) \to \{\gamma \in 2^{\Sigma}; \Sigma_{uc} \subseteq \gamma\}$, delivering the set of actions that are allowed in G by the control after a trajectory $s \in \mathcal{L}(G)$. Write S/G for the closed loop system, consisting of the initial plant G controlled by the supervisor S.

Supervisory Control Problem [29]

Given a plant modeled by an LTS $G = \langle \mathcal{X}, \Sigma, x_0, \delta \rangle$, $\Sigma_{uc} \subseteq \Sigma$ and a desired language $K \subseteq \mathcal{L}(G)$, The problem is to build a supervisor S such that $\mathcal{L}(S/G) \subseteq K$ is controllable, and for any other supervisor S' s.t. $\mathcal{L}(S'/G) \subseteq K$, $\mathcal{L}(S'/G) \subseteq \mathcal{L}(S/G)$.

In the sequel, we will be more interested in the computation of S/G rather than in the computation of the supervisor S itself, since one can easily extract S from S/G.

The solution of the SCP is classically called the most permissive solution : using $K^{\uparrow}$ to denote the supremal controllable sub-language of K w.r.t. $\mathcal{L}(G)$ and Σ_{uc}, we have $\mathcal{L}(S/G) = K^{\uparrow}$ (see [29]). The standard algorithm used to compute this supremal language is an iterative algorithm starting with the automaton product $H \times G$, with $\mathcal{L}(H) = K$. The iterative procedure consists of (i) removing states that violate the controllability condition, and (ii) removing states that are not reachable [29].

Definition 13 *Assume given a plant $G = \langle \mathcal{X}, \Sigma, x_0, \delta \rangle$ and let a desired behavior be modeled by the LTS H. We denote by $[H \times G]^{\uparrow}$ the LST obtained by using the standard algorithm [29] to compute the supremal controllable sub-language of $\mathcal{L}(H)$ w.r.t. $\mathcal{L}(G)$ and Σ_{uc}. Hence $\mathcal{L}(H)^{\uparrow} = \mathcal{L}([H \times G]^{\uparrow})$* •

We now present another algorithm [3] relying on a relation between the controllability and a bisimulation equivalence approach.

5.1.2 Solution of the SCP using Bisimulation

We present now the Σ'-bisimulation, where Σ' is some subset of Σ ; it reduces to strong bisimulation [28] when $\Sigma' = \Sigma$.

Definition 14 *[3, 2] Given two LTSs $H = \langle \mathcal{X}_H, \Sigma, x_{H_0}, \delta_H \rangle$ and $G = \langle \mathcal{X}_G, \Sigma, x_{G_0}, \delta_G \rangle$, and $\Sigma' \subseteq \Sigma$. A Σ'-bisimulation of H and G is a binary relation $\psi \subseteq \mathcal{X}_H \times \mathcal{X}_G$ s.t. $(x_H, x_G) \in \psi$ whenever*

1. *for all $\sigma \in \Sigma'$, for all transition $x'_H = \delta_H(x_H, \sigma)$ there exists a state x'_G s.t. $x'_G = \delta_G(x_G, \sigma)$ and $(x'_H, x'_G) \in \psi$.*
2. *vice-versa* •

Σ'-bisimulations of H and G are closed under arbitrary unions [3]. So there exists a greatest one, noted $\underline{\leftrightarrow}_{\Sigma'}$. We now formalize the link between Σ'-bisimulation and controllability:

Theorem 7 [3] *Let H and G be as in Definition 13 with $\mathcal{L}(H) \subseteq \mathcal{L}(G)$. Denote by $S^{\uparrow}$ the accessible state space of $[H \times G]^{\uparrow}$. Let $\underline{\leftrightarrow}_{\Sigma_{uc}}$ be the greatest Σ_{uc}-bisimulation of H and G, then:*

1. *$(x_{H_0}, x_{G_0}) \in S^{\uparrow}$ iff $x_{H_0} \underline{\leftrightarrow}_{\Sigma_{uc}} x_{G_0}$, and*
2. *$(x_H, x_G) \in S^{\uparrow}$ iff $x_H \underline{\leftrightarrow}_{\Sigma_{uc}} x_G$ and (x_H, x_G) is reachable from (x_{H_0}, x_{G_0}) by a sequence of state transitions that never leave $\underline{\leftrightarrow}_{\Sigma_{uc}}$.* ◇

From this theorem, we can easily derive an algorithm providing $[H \times G]^{\uparrow}$. It first computes the Σ_{uc}-bisimulation $\underline{\leftrightarrow}_{\Sigma_{uc}}$ of the H and G (using the algorithm of [10]) and next builds some product model from H and G according to the $\underline{\leftrightarrow}_{\Sigma_{uc}}$ relation. So the following Algorithm [3]:

Algorithm 3

1. *Compute* $\underline{\leftrightarrow}_{\Sigma_{uc}}$ *the greatest* Σ_{uc}*-bisimulation of* H *and* G
2. *If* $x_{H_0} \not\underline{\leftrightarrow}_{\Sigma_{uc}} x_{G_0}$*, then let* R *be the empty LTS.*
   Otherwise, let $R = (\mathcal{X}_R, \Sigma, x_{0_R}, \delta_R)$*, where:*

$$\begin{aligned} \mathcal{X}_R &= \{(x_H, x_G) \in \mathcal{X}_H \times \mathcal{X}_G\} \\ x_{R_0} &= (x_{H_0}, x_{G_0}) \end{aligned}$$

$$\forall \sigma \in \Sigma : \quad \delta_R((x_H, x_G), \sigma) = \begin{cases} (\delta_H(x_H, \sigma), \delta_G(x_G, \sigma)) \; if & \delta_H(x_H, \sigma) \underline{\leftrightarrow}_{\Sigma_{uc}} \delta_G(x_G, \sigma) \\ Undefined \; otherwise. \end{cases}$$

3. *Let* $R^{\uparrow}$ *denotes the accessible LTS of* R

Theorem 8 [3]$\mathcal{L}(R^{\uparrow}) = \mathcal{L}(H)^{\uparrow} = \mathcal{L}([H \times G]^{\uparrow})$. ◇

5.2 The SCP in the ILTS framework

5.2.1 Σ'-Bisimulation in the ILTS framework

Let us introduce the notion of *symbolic A-Bisimulation*, where A is a polynomial that will characterize the set of events Σ' of Definition 14. The notion is strongly inspired from Definition 6.

Definition 15 *Consider two (n,m)-iLTSs* $T_{\mathcal{U}} = (\mathcal{U}, Y, \mathcal{I}_{\mathcal{U}}, \rightarrow_{\mathcal{U}})$*,* $T_{\mathcal{V}} = (\mathcal{V}, Y, \mathcal{I}_{\mathcal{V}}, \rightarrow_{\mathcal{V}})$*, and a polynomial* $A \in \mathbb{Z}/_{p\mathbb{Z}}[Y]$*. A symbolic* A-bisimulation *of* $T_{\mathcal{U}}$ *and* $T_{\mathcal{V}}$ *is a binary relation* $\mathcal{R} \subseteq \mathcal{U} \times \mathcal{V}$ *s.t.* $(u, v) \in \mathcal{R}$ *(or* $u\mathcal{R}v$*) whenever*

1. *for all* $u \xrightarrow{P}_{\mathcal{U}} u'$*, there exists a finite set of transitions* $(v \xrightarrow{P_i}_{\mathcal{V}} v^i)_{i \in I}$*, such that:*
   (a) $(1 - (P \oplus A)) * \Pi_i P_i = 0$*, and*
   (b) $u'\mathcal{R}v^i$*, for all* $i \in I$*.*
2. *vice versa.* •

In Definition 15, the clause $(1-(P \oplus A)) * \Pi_i P_i = 0$ can be reinterpreted as $Sol(P) \cap Sol(A) \subseteq \bigcup_i Sol(P_i)$, that is any transition from u to u' satisfying the "criterion" denoted by A can be mimicked from v. Since A-bisimulations are closed under arbitrary unions, we can talk about the greatest A-bisimulation, written $\underline{\asymp}_A$ in the following, and say that "u and v are A-bisimilar" whenever $u \underline{\asymp}_A v$.

Symbolic A-bisimulation between iLTSs corresponds to $Sol(A)$-bisimulation (in the sense of Definition 14) between the underlying extensional LTSs:

Theorem 9 (Expressiveness) *Let* $T_{\mathcal{U}}$ *and* $T_{\mathcal{V}}$ *be two iLTSs.* $\mathcal{R}$ *is a symbolic A-bisimulation between* $T_{\mathcal{U}}$ *and* $T_{\mathcal{V}}$ *iff* $\mathcal{R}$ *is a* $Sol(A)$*-bisimulation between* $Ext(T_{\mathcal{U}})$ *and* $Ext(T_{\mathcal{V}})$*.*

Proof : $\Rightarrow$) Let $\mathcal{R}$ be a symbolic A-bisimulation between $T_{\mathcal{U}}$ and $T_{\mathcal{V}}$. Let $u\mathcal{R}v$ and let $u \xrightarrow{y}_{\mathcal{U}} u'$ in $Ext(T_{\mathcal{U}})$, with $y \in Sol(A)$. Then there exists $u \xrightarrow{P}_{\mathcal{U}} u'$ with $P(y) = 0$ and $A(y) = 0$. By definition of $\mathcal{R}$, there exists some indexes i such that $v \xrightarrow{P_i}_{\mathcal{V}} v^i$, $(1 - P \oplus A) * \Pi_i P_i = 0$ and $u'\mathcal{R}v^i$. Because $P(y) = 0$ and $(1 - P \oplus A) * \Pi_i P_i = 0$ when applied to y, $P_i(y) = 0$ for some i which proves that $v \xrightarrow{y}_{\mathcal{V}} v^i$, and we are done. Transition $v \xrightarrow{y}_{\mathcal{V}} v'$ is dealt similarly.
$\Leftarrow$) Let $\mathcal{R}$ be a $Sol(A)$-bisimulation of $Ext(T_{\mathcal{U}})$ and $Ext(T_{\mathcal{V}})$. Let $u\mathcal{R}v$ and let $u \xrightarrow{P}_{\mathcal{U}} u'$ in $T_{\mathcal{U}}$. As $u\mathcal{R}v$, for each $y_0 \in Sol(A) \cap Sol(P)$, there exists $v \xrightarrow{y_0}_{\mathcal{V}} v^{y_0}$ in $Ext(T_{\mathcal{V}})$ with $u'\mathcal{R}v^{y_0}$. Then, in $T_{\mathcal{V}}$ there

exists some P^{y_0} s.t. $v \stackrel{P^{y_0}}{\rightarrow}_{\mathcal{V}} v^{y_0}$ and $P^{y_0}(y_0) = 0$. Consider the polynomial $\Pi_{y \in Sol(P \oplus A)} P^y(Y)$. Clearly by construction, $Sol(P \oplus A) \subseteq Sol(\Pi_{y \in Sol(P \oplus A)} P^y)$, which entails $(1 - P \oplus A) * \Pi_{y \in Sol(P \oplus A)} P^y(Y) \equiv 0$. This shows that the v^y are the good candidates, which concludes the proof ◇

Assume now given two ILTSs $S_U = (U, U', Y, I_U, \mathcal{T}_U)$, $S_V = (V, V, Y, I_V, \mathcal{T}_V)$, and $A \in \mathbb{Z}/_{p\mathbb{Z}}[Y]$. Algorithm 4 computes the greatest symbolic A-bisimulation of S_U and S_V.

Algorithm 4

1. *Define the polynomial* $\psi_0(U, V) \stackrel{def}{=} 0$.
2. *Compute iteratively until stabilization the sequence* $(\psi_k(U, V))_k$ *defined by:*

$$\psi_{k+1}(U, V) \text{ is the canonical generator of the } \equiv\text{-class of}$$

$$\begin{cases} \psi_k(U, V) \\ \oplus\ \forall U' \forall Y [\mathcal{T}_U(U, Y, U') \oplus A(Y) \Rightarrow \exists V (\mathcal{T}_V(V, Y, V) \oplus \psi_k(U', V))] \\ \oplus\ \forall V \forall Y [\mathcal{T}_V(V, Y, V) \oplus A(Y) \Rightarrow \exists U' (\mathcal{T}_U(U, Y, U') \oplus \psi_k(U', V))] \end{cases}$$

3. *Call* $\psi_A(U, V)$ *the result.*

Theorem 10 *With the preceding notations,* $\psi_A(u, v) = 0 \Leftrightarrow u \underline{\asymp}_A v \Leftrightarrow u \underline{\leftrightarrow}_{Sol(A)} v$. ◇

The proof is similar to the one of Theorem 2.12 of [12].

5.2.2 Application to the SCP

In this section, we present the application of the techniques developed in the previous section to solve the SCP. The non-blocking aspect of the SCP is here discard, but would also fit within this framework.

Event-controlled ILTS.

In order to match Section 5.1, we need to introduce the notion of Event-controlled ILTS, which are assumed to be deterministic[6]. So the definition:

Definition 16 *An Event-controlled ILTS is given by a deterministic* (n, m)*-ILTS and two polynomials* P_c *and* P_{uc} *of* $\mathbb{Z}/_{p\mathbb{Z}}[Y]$ *s.t.*

$$Sol(P_c) \cap Sol(P_{uc}) = \emptyset \quad (29)$$

$$Sol(P_c) \cup Sol(P_{uc}) \subseteq (\mathbb{Z}/_{p\mathbb{Z}})^m \quad (30)$$

•

For such an Event-controlled ILTS and with the preceding notations, $Sol(P_c)$ (resp. $Sol(P_{uc})$) corresponds to the set of controllable (resp. uncontrollable) events as in Section 5.1, i.e. $Sol(P_c) = \Sigma_c$ and $Sol(P_{uc}) = \Sigma_{uc}$. Note that Equation (29) ensures that an event is either controllable or uncontrollable.

The Desired Behavior.

Let $T = (X, X', Y, I, \mathcal{T}, P_{uc}, P_c)$ be an Event-controlled $ILTS$ modeling the plant. Let $\mathcal{O}(X, Y, X')$ be a polynomial in $\mathbb{Z}/_{p\mathbb{Z}}[X, Y, X']$, called a *control objective* on T.

Consider now the Event-controlled (n,m)-ILST $T_{\mathcal{O}} = (X, X', Y, I, \mathcal{T} \oplus \mathcal{O}, P_{uc}, P_c)$. $T_{\mathcal{O}}$ models the desired behavior. Intuitively, we only consider in the explicit LTS $Ext(T)$ the set of states/events/transitions that satisfy $\mathcal{O}(X, Y, X') = 0$.

[6] i.e. $\forall x, y,\ Sol(T(x, y, X'))$ as at most one element

$\mathcal{O}$ is used to express control objectives as predicates over the states and/or the events: for example, if we want to ensure a *safety property* over a particular set of states of the plant (say $Sol(\alpha(X))$), the control objectives $\mathcal{O}$ will be defined as $\mathcal{O}(X, X') = \alpha(X) \oplus \alpha(X')$. In other words, the supervisor, we want to compute with this particular polynomial, has to ensure the invariance of the set of states $Sol(\alpha(X))$. Some other control objectives can also be considered: $\mathcal{O}$ can result from a previous SCP computation (e.g. *attractivity, reachability, optimal control* [22]).

The supervisor Computation.

Assume we have two Event-controlled ILTSs T and $T_{\mathcal{O}}$ as defined in the previous section. First of all, remark that $T_{\mathcal{O}}$ is the initial system T restricted by polynomial $\mathcal{O}$. Hence the proposition:

Proposition 4 $\mathcal{L}(Ext(T_{\mathcal{O}})) \subseteq \mathcal{L}(Ext(T))$ ◇

That is to say, the behavior of $T_{\mathcal{O}}$ is a sub-behavior of T. Or in other words, the language generated by $Ext(T_{\mathcal{O}})$ is included in the language generated by $Ext(T)$. In the sequel we rename the states variables (source and target) of T and $T_{\mathcal{O}}$ such that $T = (X, X', Y, I, \mathcal{T})$ and $T_{\mathcal{O}} = (X_{\mathcal{O}}, X'_{\mathcal{O}}, Y, I_{\mathcal{O}}, \mathcal{T}_{\mathcal{O}})$. Let $\psi_{P_{uc}}$ be the result of Algorithm 2 applied on T and $T_{\mathcal{O}}$, i.e. the greatest symbolic P_{uc}-bisimulation of T and $T_{\mathcal{O}}$.

Build now the ILTS $T_C = (X_C, X'_C, Y_C, I_C, \mathcal{T}_C)$, where

- $X_C = X \cup X_{\mathcal{O}}$, $X'_C = X' \cup X'_{\mathcal{O}}$, and $Y_C = Y$
- $I_C(X_C) = I(X) \oplus I_{\mathcal{O}}(X_{\mathcal{O}}) \oplus \psi_{P_{uc}}(X, X_{\mathcal{O}})$
- $\mathcal{T}_C(X_C, Y, X'_C) = \mathcal{T}(X, Y, X') \oplus \mathcal{T}_{\mathcal{O}}(X_{\mathcal{O}}, Y, X'_{\mathcal{O}}) \oplus \ \psi_{P_{uc}}(X', X'_{\mathcal{O}})$

By Algorithm 1, we have to consider the accessible set of states of the system T_C. To do so, we compute the orbit $\mathcal{O}rb$ of the system, as follows:

$$\begin{cases} \mathcal{O}rb_0(X_C) = I_C(X_C) \\ \mathcal{O}rb_{i+1}(X_C) = \Delta(\exists X_C((\exists Y \mathcal{T}_C(X_C, Y, X'_C)) \oplus \mathcal{O}rb_i(X_C))) \end{cases}$$

where Δ is a function which renames the variable X' in X. It is then sufficient to intersect the transition relation $\mathcal{T}_C$ with the orbit to obtain a new relation, where all the states are accessible.

$$\mathcal{T}'_C(X_C, Y, X'_C) = \mathcal{T}_C(X_C, Y, X'_C) \oplus \mathcal{O}rb(X'_C)$$

With the preceding notations, we can state the following theorem:

Theorem 11 *If $Sol(I_C)$ is non empty, then the ILTS $T'_C = (X_C, X'_C, Y, I_C, \mathcal{T}'_C)$ generates the supremal controllable sub-language of $\mathcal{L}(Ext(T_{\mathcal{O}}))$ w.r.t. $\mathcal{L}(Ext(T))$ and $Sol(P_{uc})$.*

Proof : First by Assumption T and $T_{\mathcal{O}}$ are deterministic and by Proposition 4, the behavior of $T_{\mathcal{O}}$ is included in the one of T. By Theorem 8, we have to show that $Ext(T'_C)$ is $R^{\uparrow}$ of Algorithm 1: consider $(x, x_{\mathcal{O}}, y)$, s.t. $\mathcal{T}'_C(x, x_{\mathcal{O}}, y, x', x'_{\mathcal{O}}) = 0$ for some x' and $x'_{\mathcal{O}}$. Because $\mathcal{T}(x, y, x') = 0$, $\mathcal{T}_{\mathcal{O}}(x_{\mathcal{O}}, y, x'_{\mathcal{O}}) = 0$ and $\psi_{P_{uc}}(x', x'_{\mathcal{O}}) = 0$, $(x, x_{\mathcal{O}}) \xrightarrow{y} (x', x'_{\mathcal{O}})$ holds in $Ext(T'_C)$ whenever $x \xrightarrow{y} x'$, $x_{\mathcal{O}} \xrightarrow{y}_{\mathcal{O}} x'_{\mathcal{O}}$ and $x' \underline{\leftrightarrow}_{Sol(P_{uc})} x'_{\mathcal{O}}$ (Theorem 10). We exactly retrieve the definition of the transition function δ_R of Algorithm 1 which, by construction of T'_C is restricted to accessible states. ◇

References

[1] T. Amagbegnon, P. Le Guernic, H. Marchand, and E. Rutten. Signal – the specification of a generic, verified production cell controller. *Formal Development of Reactive Systems – Case Study Production Cell,* volume 891 *of Lecture Notes in Computer Science,* Chapter VII, pages 115–129, January 1995.

[2] J. C. M. Baeten and W. P. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge Univ. Press, 1990.

[3] G. Barret and S. Lafortune. Bisimulation, the supervisory control problem and strong model matching for finite state machines. *Journal of Discrete Event Dynamic Systems*, 8(4):337–429, December 1998.

[4] A. Benveniste and G. Berry. Real-time systems designs and programming. *Proceedings of the IEEE*, 79(9):1270–1282, September 1991.

[5] A. Benveniste, T. Gautier, P. Le Guernic, and E. Rutten. Distributed code generation of dataflow synchronous programs: the sacres approach. In *Proceedings of The Eleventh International Symposium on Languages for Intensional Programming, ISLIP'98*, Sun Microsystems, Palo Alto, California (USA), May 1998.

[6] R.E. Bryant. Graph-based algorithms for boolean function manipulations. *IEEE Transaction on Computers*, C-45(8):677–691, August 1986.

[7] R. De Simone and A. Ressouche. Compositional semantics of esterel and verification by compositional reductions. *Proc. CAV'94, LNCS 818*, 1994.

[8] B. Dutertre and M. Le Borgne. Control of polynomial dynamic systems: an example. Research Report 798, IRISA, January 1994.

[9] B. Dutertre and M. Le Borgne. Sigali : un sytème de calcul formel pour la vérification de programme signal. *Manuel d'utilisation*, July 1996.

[10] J.-C. Fernandez. An implementation of an efficient algorithm for bisimulation equivalence. *Science of Computer Programming*, 13:219–236, May 1989.

[11] L. E. Holloway and B. H. Krogh. Controlled Petri nets: A tutorial survey. In G. Cohen and J. P. Quadrat, editors, *11th Int. Conf. on Analysis and Optimization of Systems, Discrete Event Systems*, volume 199 of *LNCS*, pages 158–168, Berlin, Germany, June 1994. Springer-Verlag.

[12] O. Kouchnarenko and S. Pinchinat. Intensional approachs for symbolic methods. *Electronic Notes in Theoretical Computer Science*, 18, 1998.

[13] O. Kuschnarenko and S. Pinchinat. Intensional approaches for symbolic methods. In *the 23rd International Symposium on Mathematical Foundations of Computer Science, MFCS'98, International Workshop on Concurrency*, Brno, Czech Republic, August 1998.

[14] O. Kuschnarenko and S. Pinchinat. Intensional approaches for symbolic methods. Technical Report 3448, Irisa / Inria-Rennes, July 1998.

[15] M. Le Borgne, A. Benveniste, and P. Le Guernic. Polynomial dynamical systems over finite fields. In *Algebraic Computing in Control*, volume 165, pages 212–222. LNCIS, G. Jacob et F. Lamnabhi-lagarrigue, March 1991.

[16] M. Le Borgne, H. Marchand, E. Rutten, and M. Samaan. Formal verification of signal programs: Application to a power transformer station controller. In *Proceedings of AMAST'96*, volume 1101 of *Lecture Notes in Computer Science*, pages 271–285, Munich, Germany, July 1996. Springer-Verlag.

[17] P. Le Guernic, B. Chéron, T. Gautier, and C. Le Maire. Développer en langage signal. *Annales des Télécommunications*, 46(1-2):13–24, January 1991.

[18] P. Le Guernic, T. Gautier, M. Le Borgne, and C. Le Maire. Programming real-time applications with signal. Technical Report 582, Irisa, April 1991.

[19] E. Marchand, E. Rutten, H. Marchand, and F. Chaumette. Specifying and verifying active vision-based robotic systems with the Signal environment. *Int. Journal of Robotics Research*, 17(4):418–432, April 1998.

[20] H. Marchand, P. Bournai, M. Le Borgne, and P. Le Guernic. A design environment for discrete-event controllers based on the Signal language. In *1998 IEEE International Conf. On Systems, Man, And Cybernetics*, pages 770–775, San Diego, California, USA, October 1998.

[21] H. Marchand and M. Le Borgne. On the optimal control of polynomial dynamical systems over z/pz. In *4th International Workshop on Discrete Event Systems*, pages 385–390, Cagliari, Italy, August 1998.

[22] H. Marchand and M. Le Borgne. Partial order control of discrete event systems modeled as polynomial dynamical systems. In *1998 IEEE International Conference On Control Applications*, Trieste, Italia, September 1998.

[23] H. Marchand and M. Le Borgne. The supervisory control problem of discrete event systems using polynomial methods. Research Report 1271, Irisa, October 1999.

[24] H. Marchand and M. Samaan. Incremental design of a power transformer station controller using controller synthesis methodology. In *World Congress on Formal Methods (FM'99)*, volume 1709 of *Lecture Notes in Computer Science*, pages 1605–1624, Toulouse, France, September 1999. Springer Verlag.

[25] R. Milner. A calculus of communicating systems. In *Lecture Notes in Computer Science*, volume 92. Springer Verlag, 1980.

[26] R. Milner. Operational and algebraic semantics of concurrent processes. Research Report ECS-LFCS-88-46, Lab. for Foundations of Computer Science, Edinburgh, February 1988.

[27] R. Milner. A complete axiomatisation for observational congruence of finite-state behaviours. *SIAM J. Comput.*, 81(2):227–247, 1989.

[28] D. Park. Concurrency and automata on infinite sequences. In *Proc. 5th GI Conf. on Th. Comp. Sci., LNCS 104*, pages 167–183. Springer-Verlag, March 1981.

[29] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems*, 77(1):81–98, 1989.

[30] R. J. Van Glabbeek. The linear time – branching time spectrum. In J. C. M. Baeten and J. W. Klop, editors, *Proceedings CONCUR 90,* Amsterdam, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer-Verlag, 1990.

[31] R. J. Van Glabbeek. The linear time–branching time spectrum II: The semantics of sequential systems with silent moves (extended abstract). In *CONCUR '93: 4th International Conference on Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81, Hildesheim, Germany, 23–26August 1993. Springer-Verlag.

[32] R. J. Van Glabbeek and F. Vaandrager. Modular specifications in process algebra. In *Proc. 9th IFIP WG 6.1 Int. Symp. Protocol Specification, Testing and Verification, Enschede, NL*, June 1989.

[33] Rob J. van Glabbeek and W. Peter Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, May 1996.

[34] W. P. Weijland. *Synchrony and Asynchrony in Process Algebra.* PhD thesis, Univ. Amsterdam, June 1989.

A The SIGALI Tool Box

The SIGALI tool box [9] offers algebraic polynomial computation functionalities. It relies on an implementation of polynomials by Ternary Decision Diagram (TDD) (for three valued logics) in the same spirit of BDD [6], but where the paths in the data structures are decorated by values in $\{-1, 0, 1\}$ instead of $\{0, 1\}$[7].

From a practical point of view, ILST can be obtain for "free", provided we have specified the system in the high level language SIGNAL [17, 5]. In fact, the equational nature of SIGNAL leads naturally to the use of a method based on polynomial dynamical equation systems over $\mathbb{Z}/3\mathbb{Z}$ as a formal model of programs behavior[16]. The model essentially expresses boolean data and synchronizations, i.e. the control part of the SIGNAL program. There exists a lot of examples using SIGALI for SIGNAL programs: among them, a production cell [1], a power transformer station controller [16], an experiment with reactive data-flow tasking in active robot vision [19],...

A.1 Some elements of the Sigali syntax

A.1.1 The Basic Syntax

We can write polynomial expressions, lists of polynomials, etc. All the usual polynomial operations are also available (+, -, *, ...). For example, the polynomial $a^2(-b - b^2)$ is written `a^2*(-b-b^2)`.

The list of variables, polynomials, equations, etc. are written as follows: `[a,b,c,d]` is a set of variables, `[a+b,c+d]` is a list of polynomials and `[a+b=x,a*d^2=b^2]` is an equation system. As for the polynomial operations, all the possible operations over lists have been defined (union, intersection, complementary, ...).

The function call is classically written `f(x,y,z)`. For example, in order to check the invariance of a set of states defined by a polynomial g w.r.t. an ILTS S, we write:

```
> Invariant(S,g);
```

The result of such a computation is either *true* or *false*.

The affectation is simply written as follows: `symbol:expression;` . For example,

```
g:  gen([a+b=x,a*d^2=b^2]);
```

[7] We can also deal with numerical aspects using Arithmetic Decision Diagrams (ADDs) [22]

will attached the name `g` to the principal generator of this equation system (which is then computed).

Fix point computation can also be performed. For example, given :

$$\begin{cases} p_0 &= 0 \\ p_{i+1} &= p^2 + 1 \end{cases}$$

the corresponding expression in SIGALI is

```
loop x=x^2+1 init 0;
```

Of course, such sequences do not always converge. This is not checked by the system.

A.1.2 Some useful functions

There exist more than 120 different functions that belong to the kernel of the SIGALI languages. We here just provides the one that we will use in Section A.2.

declare(var,var)→ Lvar	declaration of variables
union_lvar(Lvar,Lvar)→ Lvar	performs the union of variable lists
diff_lvar(Lvar,Lvar)→ Lvar	diff_lvar(L1,L2); returns the sublist of L1 from which the variables of L2 have been removed
implies(Poly,Poly)→ Poly	implies(P1,P2)=(1-P1^2)P2
complementary(Poly)→ Poly	complementary(P)=(1-P1^2)=$\overline{\text{P}}$
intersection(Poly,Poly,...)→ Poly	P : intersection(P1,P2)⇔ $Sol(P) = Sol(P1) \cap Sol(P2)$
union(Poly,Poly,...)→ Poly	P : union(P1,P2)⇔ $Sol(P) = Sol(P1) \cup Sol(P2)$
exist(LVAR,Poly)→ Poly	exist(L,P): Existential elimination over the polynomial P w.r.t. the variables of L
forall(LVAR,Poly)→ Poly	forall(L,P):Universal elimination over the polynomial P w.r.t. L
rename(Poly,Lvar1,Lvar2)→ Poly	rename(P,L1,L2): renaming of the variables L1 by the variables L2 in P

Moreover, starting from the existing functions, it is also possible to define new functions. The syntax is the following: `def f(x,y,z):expression;`.

For example, assume we want to compute the set of reachable states of a given ILTS S (i.e the orbit of S). S is formally given by two polynomials: the implicit transition system: `Rel(X,Y,X_nexts)=0` and the initial states `Ini(X)=0`, where `X` (resp. `X_nexts`) is the set of states variables at the current instant (resp. at the next instant) and `Y` is the set of event variables.

We first introduce the function `succ(P)` that basically computes the set of reachable states from Sol(P) in one step:

```
def succ(P): rename(exist(X,intersection(P,exist(Y,Rel)),X_nexts,X);
```

The orbit of the ILST S is then simply given by :

```
def Orbit(S): loop x = union(x,succ(x)) init Ini;
```

A.2 Quotient system computation w.r.t. the strong bisimulation

We here assume that the ILST has already been defined in SIGALI. It is formally given by two polynomials: `Rel(X,Y,X_nexts)=0` and `Ini(X)=0`. First we a copy of the ILST `Rel` into `Rel_d` by renaming the variables.

```
% copy of states variables at the current instant%
X_d : declare(X_nexts);
%  copy of states variables at the next instant%
X_d_nexts : declare(X_d);
```

```
% The copy of  Rel(X,Y,X_nexts)%
Rel_d : rename(Rel,union_lvar(X,X_nexts),union_lvar(X_d,X_d_nexts));

 % new set of variables  that will be used further %
X_X_d : union_lvar(X,X_d);
X_X_d_nexts : union_lvar(X_nexts,X_d_nexts);
```

We then compute the greatest auto-bisimulation according to Algorithm 1.

```
Bis: loop x =
intersection(x,
  forall(X_nexts, forall(Y,
      implies(Rel,exist(X_d_next,intersection(Rel_d,rename(x,X_X_d,X_X_d_nexts))))),
  forall(X_d_nexts, forall(Y,
      implies(Rel_d,exist(X_next,intersection(Rel, rename(x,X_X_d,X_X_d_nexts)))))
      init 0;
```

As a result, we obtain a relation $Bis(X, X_d)$, such that $Bis(x, x_d) = 0 \Leftrightarrow x \underline{\leftrightarrow} x_d$
We now compute the function ϕ describe in Section 3.2:

```
phi : classes(Bis,Orbit,X,X_d,Z);
```

As a result, we have a relation $\phi(Z, X)$, such that $\phi(z, x) = 0 \Leftrightarrow z$ is the representative of the bisimulation lass of x is. As explained in Section 3.2, some new variables Z have been added during this computation. We can recover their list as follows :

```
Z : diff_lvar(varof(phi),X);

%the set of state variables at the next instant%
Z_nexts : declare(Z);
```

Finally, the reduced model is computed as follows (according to Relation (7)):

```
%The implicit reduced transition relation Rel_Red(Z,Y,Z'0=0 is : %
Rel_red : exist(union_lvar(X,X_nexts),
       intersection(phi,Rel, rename(phi,union_lvar(Z,X),union_lvar(Z_nexts,X_nexts))));

% the polynomial encoding the new initial states is given by : %
Init_red : exist(X,intersection(phi,Ini));
```

A.3 Experimental results in $\mathbb{Z}/_{3\mathbb{Z}}$:

we here provide some experimental results dealing with the computation of the reduced model of a given model S according to (strong) bisimulation. T corresponds to the transition relation of S whereas T' corresponds to the one of the reduced system. In the X (resp. Y, Z) column, we write the number of variables (hence, the relation $T(X, Y, X')$ will have $2|X| + |Y|$ variables). We here make a comparison between the number of reachable states in the two systems as well as the size (in terms of TDD nodes) of their transition relations.

	Initial System $T(X,Y,X')$				Reduced System $T'(Z,Y,Z')$		
Name	X	Y	Reachable States	TDD Nodes	Z	Reachable States	TDD Nodes
CO.z3z	9	4	9	145	2	9	38
TREAT.z3z	7	14	80	6884	3	24	669
PRESS.z3z	11	4	19	438	1	1	7
LINK_CELL.z3z	11	16	60	3131	2	8	418
DEP_CELL.z3z	12	24	2114	122992	5	162	42872
SRI.z3z	17	24	1808	54516	1	1	28
POWER_TRSP_2..z3z	28	21	16384	65663	6	717	54768
POWER_TRSP_1.z3z	23	32	105452	754658	7	1202	44779

Contents