Project 22703: SYRF

Deliverable 3.3.1

Handling Numerical Properties

---

1 Introduction

Any synchronous program can be viewed as an interpreted automaton, i.e., a finite control structure, the transitions of which are augmented with conditions and actions on variables. Existing verification tools for synchronous programs are based on the control structure, and ignore the interpretation. This abstraction has been shown to be reasonable in many practical cases, since the critical properties of a reactive system often depend only on the control. However, there are cases where the behavior of datas must be taken into account.

Using the technique of synchronous observers, any *safety property* of a synchronous program can be expressed as the unreachability of some states in the interpreted automaton. Finite-state-based methods, which generally abstract away the interpretation, sometime wrongly conclude that these "bad" states can be reached. Task 3.3 of the project is devoted to obtaining more precise results by means of two approaches:

- A *static* method, whose goal is to cut, in the automaton, transitions that are guarded by unfeasible conditions. Interesting results can be obtained using a procedure checking the satisfiability of systems of linear constraints.
- A more ambitious *dynamic* method, taking into account the dynamic behavior of numerical variables. Based on abstract interpretation, this approximate method gives quite precise results, in particular for properties involving real-time delays [Hal93]. The analysis tool POLKA, developed by INRIA/Spectre, is based on this method. Annex 3.3.A is a paper [HPR97] on this method, applied to the verification of both synchronous programs and hybrid systems (Task 7).

In Section 2, we explain our approach by means of a simple example. Sections 3 and 4 present the work performed on Task 3.3 during the first year of the project.

2 Approach

2.1 Numerically unfeasible transitions

Let us consider a very simple example, in order to explain the general approach. The example is extracted from a subway speed regulation system. The program determines if a train is early or late in the following way: The train is able to detect and count

- beacons, that are placed along the track;

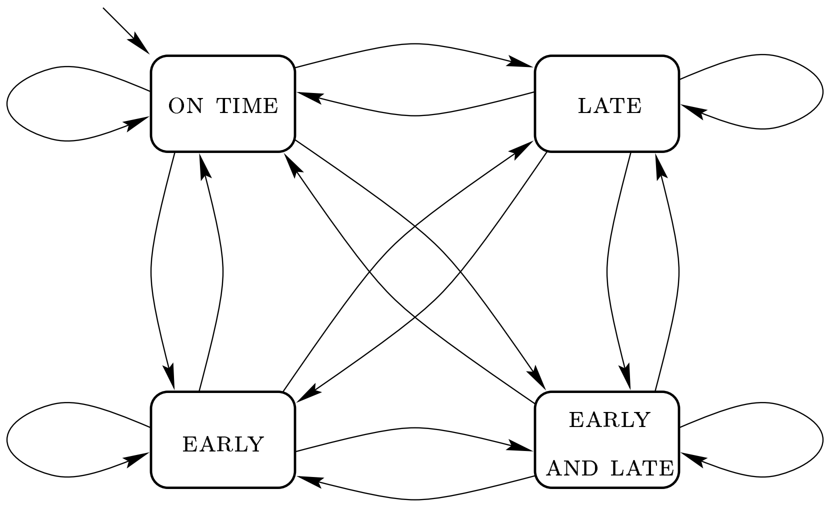

Figure 1: The interpreted automaton of the subway speed regulator

- radio signals broadcasted from a central clock.

Let Δ denote the difference between the number of detected beacons and the number of received signals. Ideally, a train should detect exactly one beacon between each two successive signals, so, it should be "in time" only when $0 \leq \Delta \leq 1$. Now, to avoid excessive shaking, an hysteresis is applied:

- Whenever Δ becomes greater than 10, the train is considered early until Δ becomes less than 0.
- Whenever Δ becomes less than -10, the train is considered late until Δ becomes greater than 0.

If we describe this system in our favourite synchronous language — assuming, for simplicity, that the inputs B (for "beacon") and S (for "signal") are exclusive —, and if we compile the resulting program into an interpreted automaton, we get the automaton represented by Fig. 1, where, for instance, the code of the state "ON TIME" looks like

```
ON_TIME: if S then delta := delta-1;
            if delta <= -10 then
               if delta < 10 then goto LATE
               else goto EARLY_AND_LATE
            else goto ON_TIME
         else if B then delta := delta+1;
            if delta >= 10 then
               if delta > -10 then goto EARLY
               else goto EARLY_AND_LATE
            else goto ON_TIME
         else goto ON_TIME
```

On this example, it is clear that some properties cannot be checked by considering only the control structure of the automaton. For instance, on Fig. 1, the state "EARLY AND LATE" seems to be reachable, though, obvioully, a train cannot be both early and late

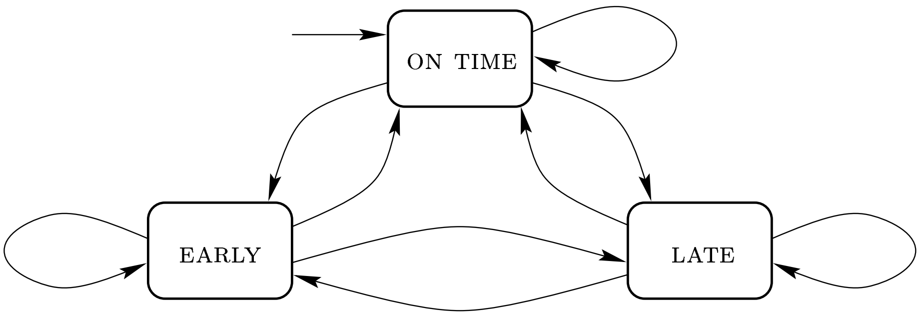

Figure 2: The subway automaton without statically unfeasible transitions

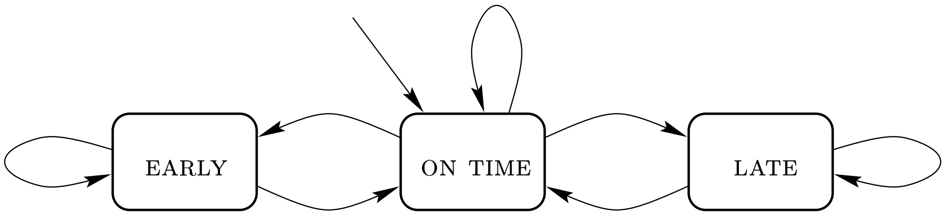

Figure 3: The subway automaton without dynamically unfeasible transitions

at the same time. To show that this state is not actually reachable, one has to look at the interpretation and to know something about numbers: For instance, in state "ON TIME" the transitions leading to state "EARLY AND LATE" are both guarded by the condition

$$(\Delta \leq -10) \land (\Delta \geq 10)$$

which is clearly numerically unfeasible.

Transitions guarded by unfeasible conditions are called *statically unfeasible*. If we remove statically unfeasible transitions from our example automaton, we get the automaton represented in Fig. 2.

On this simplified automaton, the unreachability of the state "EARLY AND LATE" is now apparent. Now, on this automaton, the state "EARLY" seems directly reachable from the state "LATE" and conversely. This is obviously not the case of the actual program, since in state "EARLY" we always have $0 < \Delta$, so, by decrementing Δ once, we cannot satisfy the guard $\Delta \leq -10$ of the transition leading to "LATE". Such transitions, whose guard cannot be satisfied because of invariant properties satisfied in their source state, are called *dynamically unfeasible*. Removing dynamically unfeasible unfeasible transition from the automaton of Fig. 2 results in the one drawn in Fig. 3.

Our goal is to detect numerically unfeasible transitions. Of course, the general problem is undecidable, so we have to consider restricted cases and heuristic methods. Our approach is based on a restriction to *linear algebra*, for obvious computational reasons, and because it is enough to deal with many practical problems. The basis of our tool, called POLKA, is a library of operations on *convex polyhedra*, i.e., sets of solutions of a system of *linear constraints* (equations and inequalities).

2.2 Operations on convex polyhedra

Let us briefly recall the basic definitions and classical results about convex polyhedra and their representations. Let $\mathbb{E} = \mathbb{R}$ or $\mathbb{Q}$ (our implementation works on rational numbers).

A *convex polyhedron* (or, simply, a polyhedron) in $\mathbb{E}^n$ is the set

$$P = \{X \in \mathbb{E}^n \mid AX \leq B\}$$

of solutions of a system of linear inequalities, where A is an $m \times n$ matrix, and B is an n-vector with coefficients in $\mathbb{E}$. The system $(AX \leq B)$ is the *system of constraints* of P. For any polyhedron P, there exists two finite sets $V = \{v_1, \ldots, v_k\}$ (set of *vertices*) and $R = \{r_1, \ldots, r_\ell\}$ (set of *rays*) of vectors in $\mathbb{E}^n$, such that

$$P = \left\{ \sum_{i=1}^{k} \lambda_i v_i \; + \; \sum_{j=1}^{\ell} \mu_j r_j \;\middle|\; \lambda_i \in [0,1], \; \mu_j \geq 0, \; \Sigma_i \lambda_i = 1 \right\}$$

i.e., a point belongs to P if and only if it is the sum of a *convex combination* of P's vertices and a *positive combination* of P's rays. The pair (V, R) is a *system of generators* of P.

Let P_1, P_2 be two polyhedra, $(A_i X \leq B_i)$, $(i = 1, 2)$ their systems of constraints, and (V_i, R_i), $(i = 1, 2)$ their systems of generators. Then

- Their intersection $P_1 \cap P_2$ is a polyhedron, the system of constraints of which is $(A_1 X \leq B_1) \wedge (A_2 X \leq B_2)$
- Their union $P_1 \cup P_2$ is not convex, in general. The *convex hull* $P_1 \sqcup P_2$ is the least polyhedron containing $P_1 \cup P_2$

  $$P_1 \sqcup P_2 \;=\; \{\lambda X_1 + (1 - \lambda) X_2 \mid X_1 \in P_1, \; X_2 \in P_2, \; \lambda \in [0,1]\}$$

  Then $(V_1 \cup V_2, R_1 \cup R_2)$ is a system of generators of $P_1 \sqcup P_2$.
- $P_1 \subseteq P_2$ if and only if $\quad A_2 v \leq B_2, \forall v \in V_1 \quad$ and $\quad A_2 r \leq 0, \forall r \in R_1$
- P_1 is empty if and only if V_1 is empty.
- Let C be an $n \times n$ matrix and D be be an n-vector. The *affine transformation* of P_1 by (C, D), is the polyhedron

  $$[C, D](P_1) \;=\; \{CX + D \mid X \in P_1\}$$

  Then $\quad V' = \{Cv + D \mid v \in V_1\} \quad , \quad R' = \{Cr \mid r \in R_1\} \quad$ is a system of generators of $[C, D](P_1)$

These classical results show that all the considered operations can be easily implemented provided both representations of polyhedra are available. The translation of each representation into the other is the only difficult and expensive operation. We use an algorithm proposed by [Che68] and improved by [LeV92]. An important feature of this translation is that it provides a *minimal* representation, thus allowing irrelevant constraints and generators to be removed.

2.3 Statically unfeasible transitions

To remove a statically unfeasible transition, one has to decide whether its guard is feasible. Thanks to the previous results, we can detect that a guard made of linear constraints is unfeasible (empty polyhedron). Notice that our emptyness condition is sometimes approximate, in case of integer variables, since we consider emptyness in $\mathbb{E}^n$ instead of $\mathbb{Z}^n$. Removing statically unfeasible transitions has been integrated in the model-checker LESAR [HLR92] in the following way: if, during the enumerative traversal of reachable states of the control automaton, a "bad" state (i.e., violating the wanted property) is encountered, the tool looks backward at the transitions leading to this state and its ancestors, trying to cut all the paths by an unfeasible transition.

2.4 Dynamically unfeasible transitions

Dynamically unfeasible transitions are much more difficult to detect. We use our "linear relation analysis" [HPR97] for that. Based on a particular version [CH78] of Cousot's "abstract interpretation" [CC77], this analysis computes, in each state s of the automaton, an upper approximation P_s (a polyhedron) of the set of possible valuations of numerical variables when the control is in that state. The system of constraints of P_s is then an invariant property of numerical variables when the control is in s. Once P_s is computed, one can check, for each transition starting form s, whether its guard intersects P_s.

The analysis is based on "forward collecting semantics" of the interpreted automaton: With each state s (resp., each transition $s \to s'$) we associate the set X_s (resp., $Y_{s \to s'}$) of all possible valuations of numerical variables when the control is in state s (resp., just after the transition $s \to s'$ has occurred). These sets can be characterized by means of a system of mutually recursive equations:

$$\forall s,\ X_s = \bigcup_{s' \to s} Y_{s' \to s} \qquad \forall t = (s \to s'),\ Y_t = a_t(X_s \cap g_t)$$

where

- g_t is the set of valuations satisfying the guard of transition t
- a_t is the action performed when the transition is fired
- $a(X) = \{a(x) \mid x \in X\}$

We restrict ourselves to the case where guards are linear constraints, and actions are assignments of linear expressions. The analysis iteratively computes upper approximations P_s of X_s and Q_t of Y_t, all of them being polyhedra, using two kinds of approximations:

- Set unions are approximated by convex hulls;
- To enforce the convergence of iterations, an extrapolation operator (Cousot's "widening") is applied, to "guess" the limit of possibly infinite sequences of polyhedra.

During the first year of the project, two topics have been studied:

- The library of operations on polyhedra has been improved, by allowing *Cartesian products of polyhedra* to be handled as such.

- A more fundamental work has been started, concerning the choice of a suitable control structure of the automaton with respect to the property to be proved.

3 Handling polyhedra as Cartesian products

The library of operations on polyhedra is reasonably efficient as long as these operations are applied to polyhedra of small dimensions (less than 10 variables). However, the computations become rapidly untractable when the dimension increases, because the size of polyhedra representations can grow exponentially with the dimension of the space (e.g., a hypercube of dimension n has 2^n vertices).

Our goal is to avoid this exponential explosion in a case that appears quite often in practice: When the set of variables can be partitioned into independent subsets, a polyhedron is in fact a *Cartesian product* of polyhedra of smaller dimensions. For instance, an n dimensional hypercube H, defined by the following system of constraints

$$H = (0 \leq x_i \leq 1 \ , \ i = 1 \dots n)$$

can be expressed as the Cartesian product of n polyhedra H_i of dimension 1:

$$\bigtimes_{i=1}^{n} H_i \quad \text{with} \quad H_i = (0 \leq x_i \leq 1)$$

If this representation of H as a Cartesian product can be preserved in operations, one has only to consider the $2n$ vertices of the H_i, instead of the 2^n vertices of H.

Such an optimization requires that

- Cartesian products be detected
- operations be performed on Cartesian products, without explicitly computing the product.

We propose below a solution to these problems, with specific emphasis on the difficult operation, which is the convex hull.

3.1 Cartesian products of polyhedra

If $P_1 \subset \mathbb{E}^{k_1}, P_2 \subset \mathbb{E}^{k_2}$ then $P_1 \times P_2 \in \mathbb{E}^{k_1+k_2}$. Let $(V_i, R_i), (i = 1, 2)$ be the system of generators of $P_i, (i = 1, 2)$. Then

$$V = \{(v_1, v_2) \mid v_i \in V_i\} \quad R = \{(r_1, 0) \mid r_1 \in R_1\} \cup \{(0, r_2) \mid r_2 \in R_2\}$$

is a system of generators of $P_1 \times P_2$.

Let ϕ be a permutation of $\{1, \dots, n\}$: If $X \in \mathbb{E}^n$ and if P is a polyhedron of $\mathbb{E}^n$, let us note $\phi(X)$ the vector Y, and $\phi(P)$ the polyhedron Q, such that

$$Y_i = X_{\phi(i)}, i = 1 \dots n \quad \text{and} \quad Q = \{\phi(X) \mid X \in P\}$$

A polyhedron P of $\mathbb{E}^n$ can be decomposed along two orthogonal subspaces E^{k_1} and E^{k_2} if and only if there exists a permutation ϕ of $\{1, \dots, n\}$, and there exists $P_1 \subset \mathbb{E}^{k_1}$ and $P_2 \subset \mathbb{E}^{k_2}$ such that $P_1 \times P_2 = \phi(P)$. Of course, P_1 and P_2 are polyhedra.

Decomposition: Let $(AX \leq B)$ be the system of constraints of a polyhedron $P \in \mathbb{E}^n$. Let $I = \{i_1, i_2, \ldots, i_{k_1}\}$ and $J = \{j_1, j_2, \ldots, j_{k_2}\}$ be two sets of integers, being a partition of $\{1, \ldots, n\}$, and such that

$$\forall i \in I, j \in J,\ \forall k \in \{1, \ldots, n\} \quad A_{ki} = 0 \ \vee \ A_{kj} = 0 \tag{1}$$

Then P can be decomposed along two orthogonal subspaces $\mathbb{E}^{k_1}$ and $\mathbb{E}^{k_2}$ according to the permutation

$$\phi : \ell \mapsto \begin{cases} i_\ell & \text{if } \ell \leq k_1 \\ j_{\ell - k_1} & \text{if } \ell > k_1 \end{cases}$$

To check whether a polyhedron P is a Cartesian product $P_1 \times P_2$, one only has to look at its system of constraints $(AX \leq B)$ and to check whether the set of variables can be partitioned into $I \oplus J$ satisfying the independence property (1). The set of constraints k such that there exists an index $i \in I$ (resp., $\in J$) with $A_{ki} \neq 0$, constitutes a system of constraints of P_1 (resp., of P_2).

Simple operations: Let $P = P_1 \times P_2$ and $Q = Q_1 \times Q_2$ be two polyhedra, decomposed along E^{k_1} and E^{k_2} according to the same permutation (Here, for simplicity, we consider it to be the identity permutation). Then

- $P \cap Q = (P_1 \cap Q_1) \times (P_2 \cap Q_2)$
- $P \subseteq Q$ if and only if $P_1 \subseteq Q_1$ and $P_2 \subseteq Q_2$
- P is empty if and only if either P_1 or P_2 is empty
- If $C_{ij} = C_{ji} = 0,\ \forall i \in I, j \in J$, then $[C, D](P) = [C, D](P_1) \times [C, D](P_2)$

3.2 Convex hull

The computation of a convex hull is less obvious: Given two polyhedra $P = P_1 \times P_2$ and $Q = Q_1 \times Q_2$, decomposed in the same way, one has

$$P \sqcup Q \ \subseteq\ (P_1 \sqcup Q_1) \times (P_2 \sqcup Q_2)$$

but, in general, the reverse inclusion does not hold. For instance

$$(x = 0 \wedge y = 0) \sqcup (x = 1 \wedge y = 1) \ = \ (0 \leq x \leq 1 \wedge x = y) \quad \text{(a segment)}$$

but

$$\begin{aligned} ((x = 0) \sqcup (x = 1)) \times ((y = 0) \sqcup (y = 1)) \ &= \ (0 \leq x \leq 1) \times (0 \leq y \leq 1) \\ &= \ (0 \leq x \leq 1 \wedge 0 \leq y \leq 1) \quad \text{(a square)} \end{aligned}$$

However,

$$\begin{aligned} (x = 0 \wedge y = 0) \sqcup (x = 1 \wedge y = 0) \ &= \ (0 \leq x \leq 1 \wedge y = 0) \\ &= \ (0 \leq x \leq 1) \times (y = 0) \end{aligned}$$

Our goal is to compute the convex hull of two decomposed polyhedra, while detecting whether the result is still decomposed, and, if so, without losing the advantage of the decomposition. Our idea is to introduce an auxiliary variable, whose role is to avoid losing information while separatedly applying the convex hull to components. Let v be a fresh variable, and let us define

$$\begin{array}{ll} P'_1 = P_1 \wedge (v=0) & P'_2 = P_2 \wedge (v=0) \\ Q'_1 = Q_1 \wedge (v=1) & Q'_2 = Q_2 \wedge (v=1) \end{array}$$

Let $P\downarrow_v$ denote the projection of P according to v (i.e., removing v from the representations of P. This operation is straightforward on system of generators, and can be performed on systems of constraints, using Fourier-Motzkin elimination procedure). Then

$$\begin{array}{rcl} P \sqcup Q & = & (P_1 \times P_2) \sqcup (Q_1 \times Q_2) \\ & = & (P'_1\downarrow_v \times P'_2\downarrow_v) \sqcup (Q'_1\downarrow_v \times Q'_2\downarrow_v) \\ & = & ((P'_1 \sqcup Q'_1) \times (P'_2 \sqcup Q'_2))\downarrow_v \end{array}$$

Let $R_i = P'_i \sqcup Q'_i$, $i = 1, 2$. R_1 and R_2 are *dependent* if either v is bounded from below in R_1 and from above in R_2 by non constant expressions, or conversely. If R_1 and R_2 are independent then

$$P \sqcup Q = R_1\downarrow_v \times R_2\downarrow_v$$

Otherwise $P \sqcup Q$ cannot be decomposed in the same way as P and Q, and

$$P \sqcup Q = (R_1 \wedge R_2)\downarrow_v$$

Examples:

- Let $P = (x=0) \times (y=0)$ and $Q = (x=1) \times (y=0)$. We get $R_1 = (0 \leq x = v \leq 1)$ and $R_2 = (y = 0 \wedge 0 \leq v \leq 1)$. In R_2, v is not bounded by any non constant expression, so R_1 and R_2 are independent, and

$$P \sqcup Q \;=\; R_1\downarrow_v \times R_2\downarrow_v = \;(0 \leq x \leq 1) \times (y=0)$$

- Now, let $P = (x=0) \times (y=0)$ as before, and $Q = (x=1) \times (y=1)$. Then $R_1 = (0 \leq x = v \leq 1)$ and $R_2 = (0 \leq y = v \leq 1)$. Now, R_1 and R_2 are non longer independent (e.g., $x \leq v$ in R_1 and $v \leq y$ in R_2), and

$$P \sqcup Q \;=\; (R_1 \wedge R_2)\downarrow_v = \;(0 \leq x = y \leq 1)$$

4 Refining the control structure

The research concerning the second topic is just starting. The goal is twofold:

- On one hand, a severe limitation of linear relation analysis is the size of the control structure it is based on. Obviously, if the interpreted automaton we start with, has several millions states and transitions — a size which is non uncommon in real applications —, the analysis will be untractable, since it will have to solve systems of millions equations in polyhedra. So, we are looking for a way of starting with a very rough control structure, which would be progressively refined according to the property to be proved. The goal is to obtain the minimal control structure allowing the property to be verified.

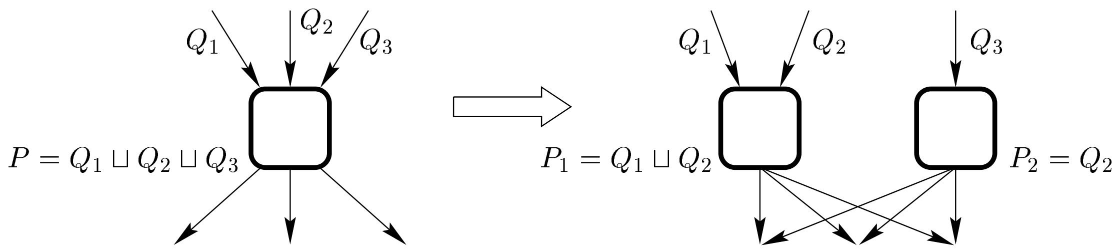

Figure 4: Splitting a control state

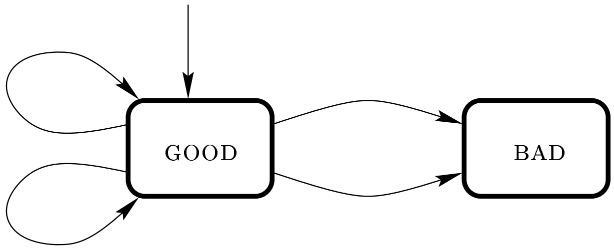

Figure 5: A minimal starting control structure

- On the other hand, the approximation by convex hull is often the main cause of failure of the method. Taking into account non-convex polyhedra raises algorithmic problems and, even if solved, would dramatically increase the complexity of the analysis. However, when the convex hull performed in a state appears to lose too much information (according to some criteria to be studied), one can split the state into several states, sharing the same successors (see Fig. 4).

To start tackling these problems, we first consider purely numerical programs. A minimal control structure to start with, is an automaton with two control states, distinguishing between "good" valuations (that satisfy the property) and "bad" valuations (see Fig. 5). Now,

- Surely, one has to distinguish, among "good" valuations, those which can directly lead to "bad" valuations, from others (starting a kind of "backward analysis" in the lines of [BFH$^+$92]).

- On the other hand, as soon as one has to perform a convex hull between polyhedra, all of which contain only "good" valuations, but the convex hull of which contains "bad" valuations, this convex hull will surely be too much a rough approximation.

More formally, we consider transition systems whose states, called *valuations*, are vectors in $\mathbb{E}^n$. $\mathbb{E}^n$ is supposed to be partitioned into G ("good") and B ("bad"). What we need is a kind of "convex bisimulation", say a binary relation $\approx \subseteq \mathbb{E}^n \times \mathbb{E}^n$, such that:

$$\begin{array}{rll} X \approx Y \iff & \text{either } X, Y \in B & \\ & \text{or } \;\; X, Y \in G & \\ & \quad\;\; \text{and} & \forall X' \text{ such that } X \to X', \exists Y' \text{ such that } Y \to Y' \\ & & \text{and } \forall \lambda \in [0,1],\ X' \approx (\lambda X' + (1-\lambda)Y')) \\ & \quad\;\; \text{and} & \forall Y' \text{ such that } Y \to Y', \exists X' \text{ such that } X \to X' \\ & & \text{and } \forall \lambda \in [0,1],\ X' \approx (\lambda X' + (1-\lambda)Y')) \end{array}$$

This definition raises several questions:

- Given G and B, is there a greatest convex bisimulation?
- In general, a convex bisimulation has an infinite number of equivalence classes. Now, if it happens that a finite convex bisimulation exists, is it always constructed by an iterative partitioning algorithm like the one proposed in [BFH+92]?

References

[BFH+92] A. Bouajjani, J.-C. Fernandez, N. Halbwachs, P. Raymond, and C. Ratel. Minimal state graph generation. *Science of Computer Programming*, 18:247–269, 1992.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th ACM Symposium on Principles of Programming Languages, POPL'77*, Los Angeles, January 1977.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th ACM Symposium on Principles of Programming Languages, POPL'78*, Tucson (Arizona), January 1978.

[Che68] N. V. Chernikova. Algorithm for discovering the set of all solutions of a linear programming problem. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 8(6):282–293, 1968.

[Hal93] N. Halbwachs. Delay analysis in synchronous programs. In *Fifth Conference on Computer-Aided Verification, CAV'93*, Elounda (Greece), July 1993. LNCS 697, Springer Verlag.

[HLR92] N. Halbwachs, F. Lagnier, and C. Ratel. Programming and verifying real-time systems by means of the synchronous data-flow programming language LUSTRE. *IEEE Transactions on Software Engineering, Special Issue on the Specification and Analysis of Real-Time Systems*, September 1992.

[HPR97] N. Halbwachs, Y.E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, August 1997.

[LeV92] H. LeVerge. A note on Chernikova's algorithm. Research Report 635, IRISA, February 1992.