MPRI – Course 2-8 - Part 1 - Verification of real-time systems

Eugene Asarin

Fall 2013 - draft - Cours 1

1 Introduction

1.1 2-8: Who, when, how?

There are three parts in this lecture course:

- Timed and hybrid automata - Eugene Asarin (LIAFA, Paris Diderot)- 8 lectures - mid-term exam.
- Temporal logic for the specification of real-time systems - Franois Laroussinie - 4 lectures - final exam.
- WModeling and verification of real-time distributed systems - Benedikt Bollig (LSV, ENS Cachan) - 4 lectures - final exam.

About us:

- `www.liafa.univ-paris-diderot.fr/~asarin`
- `www.liafa.univ-paris-diderot.fr/~francoisl`
- `http://www.lsv.ens-cachan.fr/~bollig`

1.2 2-8 and 2-9 : What is it about?

Verification beyond finite-state systems.

2-8 Quantitative and continuous models and/or properties.

2-9 Infinite-state discrete models and qualitative properties.

1.3 Reminder: verification paradigm

The problem Given a System S and a Property (Specification) P check whether the behaviour(s) of S satisfy the property P (see examples below).

The approach

- Build a model of S (usually a variant of automaton/transition system/Kripke structure of some class C)
- Express formally the specification P (usually in some logic L, or sometimes like reachability)
- Apply some algorithm (or semi-algorithm) to check whether S satisfies P

Ingredients Thus we need

- A class of models (+ a methodology how to represent real systems).

- A specification formalism (+ a methodology how to formalize real specifications).
- A verification algorithm (+ a software tool implementing it).

Ingredients In 2-8

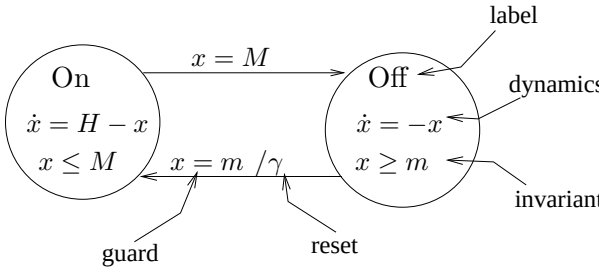
- Hybrid and timed automata and their variants (+ a methodology how to represent real systems).
- reachability problems and some logics (+ a methodology how to formalize real specifications).
- Generic techniques and concrete results concerning semi-decidability, undecidability, decidability of verification problems.
- A short discussion of practical verification issues.

2 Hybrid automata and their semantics

2.1 The first example

A thermostat. x - temperature in the room, the temperature outside is 0.

- When the heater is OFF, the room cools down : $\dot{x} = -x$
- When it is ON, the room heats: $\dot{x} = H - x$
- When $t > M$ it switches OFF
- When $t < m$ it switches ON



2.2 The meta-definition of hybrid automata

Many variants exist, most of them are similar to

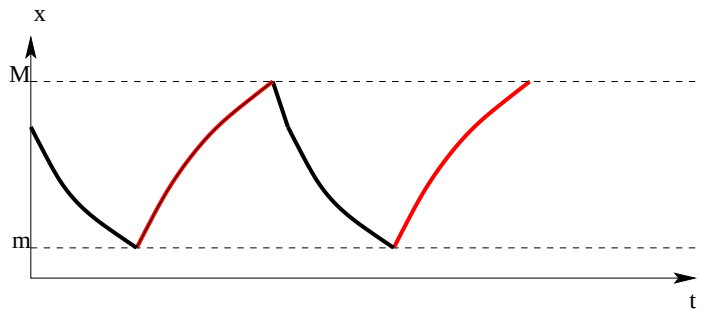
Definition 1 *A hybrid automaton is* $H = (Q, X, \Sigma, Dyn, I, \Delta)$ *with*

- Q *finite set of locations*
- $X = \mathbb{R}^n$*, continuous state space , a point in* X *= valuation of continuous variables* $\mathbf{x} = x_1, \ldots, x_n$
- Dyn*, dynamics on* X *for every* $q \in Q$*,* $Dyn(q) = f_q$*, whenever in location* q *the continuous state obeys* $\dot{\mathbf{x}} = f_q(\mathbf{x})$*.*
- I*, invariant, staying condition in* X*, whenever in location* q *the continuous state obeys* $\mathbf{x} \in I(q)$*.*
- Δ*, finite set of transitions* $\delta = (p, q, a, g, r)$
  - $p, q \in Q$*, from* p *to* q
  - $a \in \Sigma$ *a label*
  - g *a guard;* $g(\mathbf{x})$ *required to take* δ
  - r *a reset (or jump);* $\mathbf{x} := r(\mathbf{x})$ *when taking* δ

2.3 Trajectories and runs

Two semantics are used for Hybrid automata.

Trajectories A trajectory is a function $f : [0; T) \to Q \times \mathbb{R}^d$ (the domain can also be $[0, \infty)$, where $f(t)$ is the state at time t.



Runs A HA can be seen as a transition system:

Definition 2 (Transition system (S, T) of a HA)

- ***States:*** $S = Q \times \mathbb{R}^n$
- ***Transitions:*** $T = T_{flow} \cup T_{jump}$
  - $(q, \mathbf{x}_1) \overset{flow}{\to} (q, \mathbf{x}_2) \Leftrightarrow$ *we can go from* $\mathbf{x}_1$ *to* $\mathbf{x}_2$ *in ODE* $\dot{\mathbf{x}} = f_q(\mathbf{x})$
  - $(q_1, \mathbf{x}_1) \overset{jump}{\to} (q_2, \mathbf{x}_2) \Leftrightarrow$ *if we can jump.*

textbf Runs: sequences of states and transitions.

$$(\text{On}, 0) \overset{flow}{\to} (\text{On}, M) \overset{jump}{\to} (\text{Off}, M) \overset{flow}{\to} (\text{Off}, m) \overset{jump}{\to} (\text{On}, m) \cdots$$

2.4 Linear hybrid automata

A very popular subclass of Hybrid automata is the class of Linear Hibrid Automata (LHA). In two words those are HA whose dynamics, guards, invariants and updates are polyhedral sets. To be more precise we need some preliminary work.

2.4.1 Polyhedra

Consider the following types of constraints on the variables $(x_1, \ldots, x_d) = \vec{x}$ and corresponding sets in $\mathbb{R}^d$

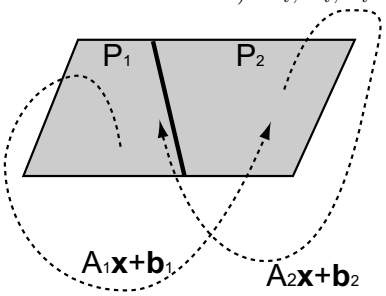
- A linear constraint: $\vec{a} \cdot \vec{x} < b$ (or $\leq$). It defines a half-space.
- A conjunction of linear constraints. It defines a convex polyhedron.
- A boolean combination of linear constraints can be rewritten as a disjunction of conjunctions of linear constraints: (prove this). It defines a polyhedron (it can be non-convex and even non-connected).

A polyhedron is rational if all the coefficients in the constraints are rational. In the sequel we only consider rational polyhedra.

The following two classes of models can be considered as subclasses of LHA

2.5 Piecewise Affine Maps (PAM)

A d-dimensional PAM is described by N polyhedra, N square matrices and N vectors (all d-dimensional and rational): $P_i, A_i, \mathbf{b}_i$. The discrete-time dynamics of such a PAM is as follows:



$$\mathbf{x} := f(\mathbf{x}) = A_i\mathbf{x} + \mathbf{b}_i \text{ for } \mathbf{x} \in P_i$$

A run of a PAM is a sequence $\mathbf{x}_1, \ldots, \mathbf{x}_n$ such that $\mathbf{x}_{i+1} = f(\mathbf{x}_i)$ for every i.

2.6 PCD

A d-dimensional Pcd is described by N (disjoint) polyhedra, and N vectors (all d-dimensional and rational): $P_i, \mathbf{c}_i$. The continuous-time dynamics of such a PCD is as follows[1]:

$$\dot{\mathbf{x}} = \mathbf{c}_i \text{ for } \mathbf{x} \in P_i$$

.

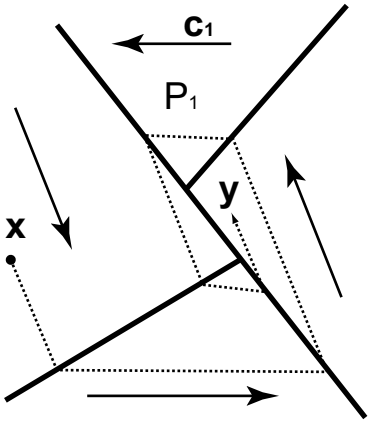


Figure 1: Example of a PCD

A PCD with N polyhedra can be seen as a linear hybrid automaton with N locations. The straightforward construction is illustrated by the picture:

[1]Some subtle details concerning the meaning of such a discontinuous differential equation are beyond the scope of these short notes.

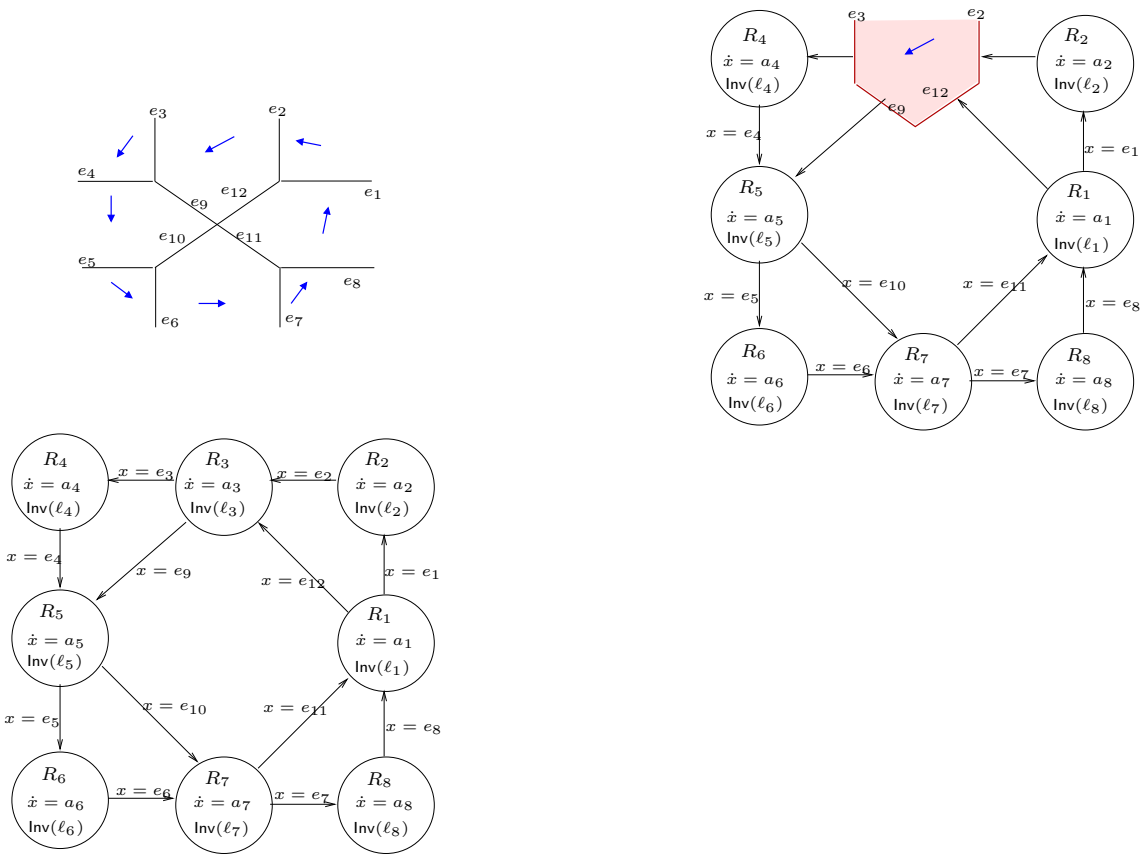


Figure 2: A PCD as an LHA

3 Semi-decidability of reachability problem

3.1 Reachability problem

The central verification problem that we explore for hybrid automata is the Reachability problem: given a system (hybrid automaton) H, and two subsets of its state space A and B find out whether there exists a run (or, equivalently, a trajectory) of H starting in A and arriving to B. We denote $R(H, A, B)$ the reachability predicate that is true iff such a run exists and false otherwise. Sometimes we consider similar problems, such as point-to-point reachability $R(H, x, y)$ (the initial and the target sets are singletons), or bounded reachability $R_n(H, A, B)$ (that is reachability in no more than n steps). Reachability problems are less general than model-checking of logics such as LTL or CTL, but they are interesting for many reasons:

- Reachability is mathematically natural.
- If H is a model of the system, A the set of its initial states, B the set of its bad states, then to verify that runs of H starting in an initial state never heat a bad state is the same as to check $\neg R(H, A, B)$
- Most safety properties on a system S can be rephrased as (non-)reachability on an augmented system S'.
- If reachability is undecidable for a class of systems, then more general model-checking problems are also undecidable for this class.

- If reachability is decidable, than one can use the ideas of the decision procedure and extend it to more general model-checking problems for the same class of systems.

3.2 Generic semi-algorithm

The following procedure P works for almost any class of systems.

function Reach(A,B)
M=A
repeat
    Mold=M
    M=M∪ Succ(M)
until M∩ B $\neq \emptyset \vee$ M=Mold
if M∩$B \neq \emptyset$ return true
    else return false

Here Succ(M) denotes the set of all the states reachable from M in one transition.

Lemma 1 (Invariant) *The main loop of the procedure P preserves the following invariant: "M is the set of all the states of the system that are reachable in no more than n transitions from the initial set A".*

Proposition 1 (Correctness) *The procedure P is correct in the following (weak) sense.*

- *If B is reachable from A, then P stops and returns true.*
- *If B is not reachable from A, then either P stops and returns false, or it never stops.*

In order to implement the procedure P we need a data structure to represent subsets of the state space, such that

- The initial and the target sets A and B can be represented by this data structure.
- If two sets M and N can be represented, than also $M \cup N$ and $M \cap N$ can be represented, and their representations can be algorithmically obtained from those of M and N
- Given some representations of M and N, there are algorithms deciding whether $M = \emptyset$ and whether $M = N$.
- Given a representation for M, a representation for Succ(M) exists and can be algorithmically obtained.

We will call such a data structure suitable.

We conclude the following generic fact concerning many verification problems:

Proposition 2 *If for a class of systems there exists a suitable data structure, then reachability is semi-decidable for this class(using the procedure P).*

Another simple but practically important remark is in order. The following variant of procedure P always stops and decides the bounded (n steps) reachability problem.

function ReachBounded(n,A,B)
count=0
M=A
repeat
    Mold=M
    M=M∪ Succ(M)
    coun=count+1 until M∩ B $\neq \emptyset \vee$ M=Mold $\vee$ count=n
if M∩$B \neq \emptyset$ return true
    else return false

We conclude:

Proposition 3 *If for a class of systems there exists a suitable data structure, then bounded reachability is decidable for this class.*

3.3 Linear constraints, polyhedra and some logic

Our aim is to apply the approach of the previous section to Linear Hybrid Automata, and the key is to come up with a suitable data structure to represent subsets of $Q \times \mathbb{R}^d$. This data structure is based on polyhedra, and we start with some definitions and technical tools.

3.3.1 Linear constraints and polyhedra

A linear constraint over $\mathbb{R}^d$ has a form $\sum_{i=1}^{d} a_i x_i < c$ or $\sum_{i=1}^{d} a_i x_i \leq c$. It defines a half-space in $\mathbb{R}^d$. We say that the constraint is rational if all the coefficients are rational number

A (finite) conjunction of linear constraints $\bigwedge_n C_n$, where each C_n is a linear constraint defines a convex polyhedron (maybe unbounded, maybe empty, maybe degenerate) in $\mathbb{R}^d$.

A finite disjunctions of conjunctions of linear constraints $\bigvee_m \bigwedge_n C_m n$ defines a finite union of convex polyhedra (maybe overlapping). We will call them polyhedral sets (or even polyhedra).

A boolean combination of linear constraints can be of course transformed to a DNF, and hence it reduces to the previous case.

In what follows we will use rational polyhedral sets (represented by DNFs of rational constraints).

3.3.2 First order logic of linear constraints

We introduce an apparently reacher logical formalism to describe subsets of $\mathbb{R}^d$, and next we prove that any set expressible in this logic is a polyhedral set.

FOLC is a first order logic whose atomic predicates are (rational) linear constraints. Formally, the syntax of FOLC is defined recursively by:

$$F ::== \sum a_i x_i < c | \neg F | F \vee F | \exists x F$$

(other operations $=, \leq, \wedge, \forall$ can be easily expressed).

We consider a natural interpretation of FOLC where the variables range over reals numbers. The semantics of a formula with n free variables is a subset of $\mathbb{R}^n$[2].

3.3.3 Fourier-Motzkin algorithm

In order to simplify formulas of FOLC, we start with removing one existential quantifier.

Lemma 2 (Fourier-Motzkin) *Any formula of the form*

$$\exists y (\bigwedge_j b_j y + \sum_j a_{ij} x_i < c_j)$$

is equivalent to a quantifier-free formula

$$\bigwedge_j \sum_j \hat{a}_{ij} x_i < c_j.$$

The latter can be obtained algorithmically from the former.

[2]if you do not know what it means please revise your course of logic

Geometrically this lemma means that the projection of a convex polyhedron is a convex polyhedron.

The proof (and algorithm) idea is as follows. We start from a formula $\exists y G$. We first partition the inequalities into 3 groups, G+, G-, G0 according to the sign of the y's coefficient b_j. Every inequation of G+ can be rewritten as an upper bound on y:

$$y < \sum a'_{ij} x_i + c'_j,$$

with $a'_{ij} = -a_{ij}/b_j$ and $c'_j = c_j/b_j$.

Every inequation of G- can be rewritten as a lower bound on y:

$$\sum a'_{ik} x_i + c'_k < y.$$

Every inequation of G0 is in fact a constraint on $\mathbf{x}$.

In order to project the variable y we build the new system of inequalities (without this variable). All the constraints of G0 are copied as is. For every upper bound (from G+) and every lower bound (from G-) we write that the lower bound is smaller than the upper bound:

$$\hat{G} = G0 \cup \{a'_{ik} x_i + c'_k < a'_{ij} x_i + c'_j \mid j \in G+; k \in G-\}.$$

We claim that

$$\exists y\ G \Leftrightarrow \hat{G}$$

$\Rightarrow$ Suppose $\exists y\ G(x, y)$. This means that for some y_0, the variables $\mathbf{x}$ satisfy $G(x, y_0)$. Hence, they satisfy G0 (there is no y there). On the other hand, for any $j \in G+$ we have $y_0 < \sum a'_{ij} x_i + c'_j$,, and for any $k \in G-$ we have that $\sum a'_{ik} x_i + c'_k < y$. By transitivity:

$$a'_{ik} x_i + c'_k < a'_{ij} x_i + c'_j.$$

Hence $\mathbf{x}$ satisfies all the constraints of $\hat{G}$, QED.

$\Leftarrow$ Suppose $\mathbf{x}$ satisfies all the constraints of $\hat{G}$. Let

$$m = \max\{a'_{ik} x_i + c'_k \mid k \in G-\},$$

i.e. the maximum of lower bounds on y. By definition of maximum m coincides with the expression in braces for some $k = k_0$. Symmetrically, let

$$M = \min\{a'_{ij} x_i + c'_j \mid j \in G+\},$$

the minimum of upper bounds, which coincides with the expression in braces for some $j = j_0$.

The inequality with indices k_0, j_0 in $\hat{G}$ ensures that $m < M$.

Let $y_0 = (m + M)/2$. This value of y is greater than all the lower bounds, and less than all the upper bounds. It is easy to see, that $\mathbf{x}, y_0$ satisfy G, and hence $\mathbf{x}$ satisfies $\exists y\ G$, QED.

The previous algorithm and lemma can be easily extended to any combination of strict and non-strict inequalities.

3.3.4 Eliminating quantifiers

Theorem 1 *Every formula of FOLC is equivalent to a quantifier-free formula.*

A logician would say: FOLC admits quantifier elimination.

What is important for us: every set that can be expressed in FOLC is a polyhedral set (and vice versa).

Sketch of proof We prove by structural induction over the formula F that it can be represented as a boolean combination of linear constraints.

$$\bigvee_i \bigwedge_j \vec{a}_{ij} \cdot \vec{x} \prec b_{ij},$$

According to definition of FOLC we have to consider the following cases.

F is a constraint Nothing to prove

$F = G \vee H$ and by inductive hypothesis $G \Leftrightarrow g$ and $H \Leftrightarrow h$, where g, h are boolean combinations of linear constraints. Hence $F \Leftrightarrow g \vee h$ is the required representation for F

$F = \neg G$ similar to the previous case

$F = \exists y\ G$ and by inductive hypothesis $G \Leftrightarrow g$ where g is a boolean combination of linear constraints. Put g in a DNF form:

$$G \Leftrightarrow \bigvee_i \bigwedge_j C_{ij},$$

where C_{ij} are linear constraints. Hence

$$F \Leftrightarrow \exists y\ \bigvee_i \bigwedge_j C_{ij} \Leftrightarrow \bigvee_i \exists y\ \bigwedge_j C_{ij}$$

By Fourier-Motzkin each subformula $\exists y\ \bigwedge_j C_{ij}$ is equivalent to some conjunction of constraints $\bigwedge_j C'_{ij}$. We conclude that

$$F \Leftrightarrow \bigvee_i \bigwedge_j C'_{ij}.$$

This concludes the proof.

Corollary 1 *Given a closed formula of FOLC (i.e. without free variables) it is decidable whether it is true or false.*

In logical slang "FOLC is decidable". The decision procedure consists in transforming the formula into a quantifier-free formula. The result will not contain any variable, only a boolean combination of inequalities between rational constants, which is easy to evaluate.

3.4 Implementing the semi-algorithm for LHA

3.4.1 Choosing a data structure

In order to implement the procedure P for LHA we need a suitable data structure to represent (some) subsets of $Q \times \mathbb{R}^d$ and associated algorithms to compute $\cup, \cap$, Succ, to decide emptiness and equality. We suggest as such a structure a list of couples (q_i, P_i) where $q_i \in Q$ and P_i is a convex rational polyhedron in $\mathbb{R}^d$ represented by a conjunction of linear constraints. Of course, such a list denotes the set $\bigcup_i \{q_i\} \times P_i$, that is a polyhedral set for each location.

3.4.2 Expressing operations

Let $M = \bigcup_i \{q_i\} \times P_i$ and $N = \bigcup_j \{r_j\} \times S_j$ be two sets represented by lists (q_i, P_i) and (r_j, S_j) respectively.

Let us perform all the useful operations on these sets.

Union To express $M \cup N$ it suffices to concatenate two lists (q_i, P_i) and (r_j, S_j).

Intersection Expressing $M \cap N$ is a bit more involved:

$$M \cap N = \bigcup_{ij|q_i=r_j} \{q_i\} \times P_i \cap S_j$$

The intersection of convex polyhedra $P_i \cap S_j$ is a convex polyhedron.

Equality Equality should be checked for every location. Namely for every $q \in Q$ we should check:

$$\bigcup_{q_i=q} P_i = \bigcup_{r_j=q} S_j,$$

that is

$$\forall \mathbf{x} \left(\bigvee (\mathbf{x} in P_i) \Leftrightarrow \bigvee (\mathbf{x} in S_j) \right).$$

We have expressed equality by a FOLC formula without free variables. Using the decision procedure for FOLC we can decide this formula, Hence equqlity is decidqble on our data structures.

Emptiness $m \neq \emptyset$ if and only if some of P_i is nonempty. To test emptyness of P_i it suffices to eliminate all variables using Fourier-Motzkin (at the end we get true if the polyhedron was nonempty, and false otherwise).

Successor

$$Succ(M) = \bigcup_{t \in \mathbb{R}+} Succ_t(M) \cup \bigcup_{\delta \in \Delta} Succ_\delta(M) = \bigcup_i \left(\bigcup_{t \in \mathbb{R}+} Succ_t(\{q_i\} \times P_i) \cup \bigcup_{\delta \in \Delta} Succ_\delta(q_i\} \times P_i) \right)$$

. Let us treat separately each of the terms of this union. For simplicity we omit the indices.

- Compute first the continuous successor: $\bigcup_{t \in \mathbb{R}+} Succ_t(\{q\} \times P) = \{q\} \times P'$, and we want to compute the set P'. Suppose the dynamics in q is $\dot{\mathbf{x}} = \mathbf{c}$ and the (convex) invariant $\mathbf{x} \in I$. Then:

  $$\mathbf{x}' \in P' \Leftrightarrow \exists \mathbf{x} \in P \exists t \in \mathbb{R}(t > 0 \wedge \mathbf{x}' - \mathbf{x} = ct \wedge \mathbf{x} \in I \wedge \mathbf{x}' \in I)$$

  We see that P' is defined by a FOLC formula, hence it is a polyhedral set, and it can be represented in the required form of a union of convex polyhedra (a DNF of linear constraints)

- Compute now a discrete successor of M by some $\delta = (q, q', a, g, u)$. As in the previous case we can consider each polyhedron of M separately. We have that $Succ_\delta(\{q\} \times P) == \{q\} \times P'$ with P' defined by:

  $$\mathbf{x}' \in P' \Leftrightarrow \exists \mathbf{x} \in P(g(\mathbf{x}) \wedge u(\mathbf{x}, \mathbf{x}')$$

  Again P' is defined by a FOLC formula, hence it is a polyhedral set, and it can be represented in the required form of a union of convex polyhedra (a DNF of linear constraints)

3.5 Assembling everything together

- The semi-algorithm A (if implemented) semi-decides reachability.
- In order to implement this semi-algorithm we need a data structure able to represent subsets of the state space. We should be able to compute on this data structure the operations $\cap, \cup, Succ$, and tests $=, \subset$.

- For LHA such a structure is provided by polyhedral sets, represented as DNFs of linear constraints.
- In order to compute necessary operations on these polyhedral sets we obtain first FOLC formulae, and next we retransform these formulae in the standard polyhedral form (that is DNFs of linear constraints). The core of this computation is Fourier-Motzkin algorithm (interesting by itself).
- We conclude with the following result:

Theorem 2 *Reachability in LHA is semi-decidable.*

And also

Theorem 3 *Bounded reachability in LHA is decidable.*

4 Undecidability of reachability problem for LHA

In the previous section we have established that reachability is semi-decidable for LHA, her we prove that it is undecidable. Moreover, to get undecidability one does not need many variables. In fact we prove that it is undecidable even for very restricted subclasses of LHA (see below):

- 2-dimensional PAMs, i.e. LHA with 2 continuous variables, trivial dynamics, and nontrivial updates.
- 4-dimensional PCDs. i.e. LHA with 2 continuous variables, nontrivial dynamics, and trivial (identity) updates.

These results are interesting (and sad) by themselves, but the proof method is more important.

4.1 Reminder: how to prove undecidability by reduction

Definition 3 *A predicate P is reducible to a predicate Q (denote $P \leq Q$, if there exists a total computable function f such that*

$$\forall x \ P(x) \Leftrightarrow Q(f(x)).$$

In fact there exist many variants of reducibility, this one is called m-reduction.

Proposition 4 *Suppose that a predicate P is undecidable, and $P \leq Q$. Then Q is also undecidable.*

Indeed, if Q were decidable, it would be possible to answer the question "$P(x)$?", because it is equivalent to $Q(f(x))$ (and f is computable and Q decidable).

This gives a nice method to prove undecidability of a problem Q: find a known undecidable problem P and to reduce it to Q (warning: this reduction of a known problem to the one under exploration is somewhat counter-intuitive). There are several well-known undecidable problems often used for such a reduction (as problem P):

- Post Correspondence Problem
- Diophantine Equations
- Halting (Reachability) for Turing Machines
- Halting (Reachability) for Minsky Machines
- ...

In this course we will use the last problem.

4.2 Minsky machines and several undecidable problems

A Minsky Machine can be seen as a finite automaton extended with two counters C and D that can store natural numbers, be incremented and decremented (by 1) and tested to be 0.

We can also see Minsky Machine as a program, with a finite number of lines, each labeled by a unique label q_i. Every line has one of six forms, three of them concern C:

q_i : $C++$; goto q_j
q_i : $C--$; goto q_j
q_i : if $C > 0$ then goto q_j else q_K
and three similar ones deal with D.

Exercice 1 *Give a formal definition of a MM, and of its run.*

A well-known fact is that MM can simulate Turing Machines, and hence many reachability problems concerning MM are undecidable. In particular:

- Given MM M, its two locations q, q', and four naturals m, n, m', n' is it possible to start at (q, m, n) and to reach (q', m', n')?
- Given MM M, its two locations q, q', and two naturals m, n is it possible to start at (q, m, n) and to reach (q', m', n') for any m', n'[3]?
- Given MM M, its two locations q, q', is it possible to start at $(q, 0, 0)$ and to reach (q', m', n') for any m', n'?
- The first two question are undecidable even for a particular MM U (a universal machine, that can be explicitely written on one page)

4.3 Our method for undecidability proofs

Suppose we want to prove for a class C of hybrid systems, that reachability is undecidable. The method that often works is the following:

- Find how to simulate any MM M by an automaton $H = F(M)$of the class C.
- Find a reduction of some reachability problem (listed above or similar) for MM to the reachability for the class H:

  $$R_M(M, \dots) \Leftrightarrow R_C(F(M), \dots)$$

  (don't forget to replace the dots by something more concrete, and to prove the equivalence).
- Deduce from the undecidability of R_M that R_C is also undecidable, as required.

The key step here is the first one, the two remaining are standard. For this reason let us give some hints to find a simulation:

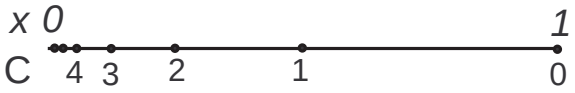
- Find a way to represent a counter value $C = n$ in a system of the class C (typically using its variables).
- Find a way to simulate the three operations on the counter (increment, decrement, test equality to 0). This gives 3 gadgets - small automata (or fragments of automata) of class C
- Extend the representation and the operations to two counters. Warning, operations on one counter should not affect the other one.
- Extend the representation and the operations to the full configuration of the MM: (q, m, n). Pay special attention to transitions (goto).

[3]it can be interpreted as a halting problem if we take as q' a special halting state

- Finalize the construction: given an MM how to build the automaton of class C (using gadgets invented on previous steps).
- State and prove a Lemma relating transitions of the given MM and of the automaton you have built.
- State and prove a Lemma relating runs of the given MM and of the automaton you have built.

4.4 Reachability is undecidable for PAMs

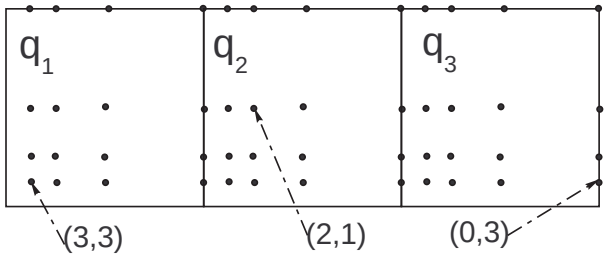
We will illustrate the methodology and prove undecidability for a very simple class of systems, PAM. We start by simulating a counter.



Counter	PAM
State space $\mathbb{N}$	State space $[0;1]$
State $C = n$	$x = 2^{-n}$
$C++$	$x := x/2$
$C--$	$x := 2x$
$C > 0$?	$x < 0.75$?

To simulate two counters we represent $C = n; D = m$ by a point $(x; y)$ on the unit square with $x = 2^{-n}; y = 2^{-m}$. (we skip the description of gadgets for operations, very similar to the previous case).

In order to simulate MM, we have to represent the full configuration of this machine (q_i, n, m). We will represent every control location by a square of unit size on the plane, the squares should be disjoint for different is. Values of the counters are represented by a point on this square as presented below.



Minsky Machine	PAM
State space $\{q_1, \ldots, q_k\} \times \mathbb{N} \times \mathbb{N}$	State space $[1; k+1] \times [0; 1]$
State $(q_i, C = m, D = n)$	$x = i + 2^{-m}, y = 2^{-n}$

The operations of the MM are realized by operations of PAM as follows.

Minsky Machine	PAM
State space $\{q_1, \ldots, q_k\} \times \mathbb{N} \times \mathbb{N}$	State space $[1; k+1] \times [0; 1]$
State $(q_i, C = m, D = n)$	$x = i + 2^{-m}, y = 2^{-n}$
q_1 : $D++$; goto q_2	$\begin{cases} x := x + 1 \\ y := y/2 \end{cases}$ if $1 < x \leq 2$
q_2 : $C--$; goto q_3	$\begin{cases} x := 2(x-2) + 3 \\ y := y \end{cases}$ if $2 < x \leq 3$
q_3 : if $C > 0$ then goto q_2 else q_1	$\begin{cases} x := x - 1 \\ y := y \end{cases}$ if $3 < x < 4$ $\begin{cases} x := x - 2 \\ y := y \end{cases}$ if $x = 4$

Using this construction we can compile any MM M into a PAM P=F(M) using the encoding above. Every instruction of the MM is translated int one or two lines in the description of PAM. Reachability in M reduces to reachability in P (Exercise: please provide details). Since reachability is undecidable for MM, we conclude with a theorem:

Theorem 4 (Koiran et al.) *Reachability is undecidable for 2-dimensional PAMs*

Remark 1 *An important open problem is whether reachability is decidable for 1-dimensional PAMs*

Corollary 2 *Reachability is undecidable for LHA.*