## Chapter 8

# Verification of Infinite-State Systems

### 8.1. Introduction

In this section, we briefly present several methods that are used for the verification of infinite-state systems. Furthermore, we show how the developments in counter systems are related to other techniques (exact methods or by approximation).

### 8.1.1. *From finite-state to infinite-state systems*

Model-checking is a well-known approach to verifying behavioral properties of computing systems; which has been very successful in the verification of finite-state systems, see e.g. [MCM 93, CLA 00b, BER 01]. The assumption that programs are finite-state is usually too restrictive; which is why, model-checking techniques for infinite-state systems have flourished over the last 20 years. However, dealing with infinity or unboundedness of computational structures has dramatic consequences computationally. The source of infinity of infinite-state systems is not unique and it can be caused by at least the following factors. Programs manipulate local or global variables interpreted in infinite domains such as the set of integers or the set of real numbers. Similarly, dynamic data structures are considered in programs which is another source of infinity. Programs contain procedure/method calls, leading to an unbounded context stack to handle the control. The size of the stack can be arbitrarily large depending on the number of nested calls, in which recursive calls may induce unbounded control structures. Similarly, process creation can be the source of infinity (see section 8.3). Furthermore, the behavior of programs depends on input data values (parameters).

Chapter written by Stéphane DEMRI and Denis POITRENAUD.

Similarly, systems may be parameterized by the number of subsystems that are synchronized, etc. Parameterized systems can therefore represent an infinite number of specific systems, depending on the parameter values. Note, for instance, to verify a property automatically regardless of the number of processes is a great challenge. Among the jungle of infinite-state systems, there are many interesting classes that have been used for modeling computer systems and for undertaking formal verification. Here are a few of them: Petri nets, see e.g. [REI 98]; timed systems (see Chapter 9); pushdown systems, see e.g. [FIN 97]; counter systems (see section 8.2) and systems with lossy channels [ABD 96].

### 8.1.2. *Decision problems for infinite-state systems*

Decision problems related to verification for infinite-state systems can be roughly divided into two categories. Firstly, numerous decision problems are essential for finite-state and for infinite-state systems such as reachability problems (including control state repeated reachability problem), model-checking temporal formulae or checking behavioral equivalences with respect to a finite-state system. Nevertheless, specific methods or adaptations of existing methods are required in order to deal with infinity. In this chapter, we shall mainly focus on reachability problems since their resolution often enables much more complex problems to be solved. Secondly, decision problems also exist that are more specific to infinite-state systems. This includes decision problems related to boundedness (for instance, checking whether a counter in a counter automaton takes a bounded amount of values) and those related to model-checking temporal formulae in which atomic formulae can state properties about unbounded values (a typical example is to replace control states in the temporal language by constraints on counter values).

### 8.1.3. *Techniques for verifying infinite-state systems*

Techniques for the verification of infinite-state systems stem from exact methods in which potentially infinite sets of configurations are finitely represented symbolically to semi-algorithms that are designed to behave well in practice (of course these latter procedures may not terminate). When exact methods provide decision procedures, this is mainly due to the identification of an underlying finite structure in the verification problem. For instance, the set of reachable configurations can be effectively represented symbolically, for instance by a formula in Presburger arithmetic for which satisfiability is known to be decidable [PRE 29] (see section 8.4). The use of Presburger arithmetic for formal verification has been advocated in [SUZ 80]. Finiteness can also occur in a more subtle way as in well-structured transition systems [FIN 01] for which termination is guaranteed thanks to underlying well quasi-orderings, see also [HEN 05] for a classification of symbolic transition systems. Section 8.4 presents exact methods to decide reachability problems for subclasses of counter systems by taking advantage of decision procedures for Presburger arithmetic. Similarly, section 8.3 introduces the class of recursive Petri nets, an extension of Petri nets with recursion, and it shows how standard proof techniques for Petri nets (that are already common and often studied class of infinite-state systems) can be extended in presence of recursion. As there are methods to verify finite systems that can be extended to infinite-state systems (for instance the automata-based approach), specific methods for infinite-state systems need to be developed too, for instance those based on well quasi-orderings.

### 8.1.4. *Automata-based symbolic representations*

A major problem for the verification of infinite-state systems consists of computing the set of configurations reachable from a set of configurations. This requires a well-suited symbolic representation for such (potentially infinite) sets and techniques to compute the transitive closure of transition relations. Regular model-checking is an approach that represents sets of configurations by regular sets of finite words (or infinite words, or trees) and transducers encode the transition relations of the systems. Regularity is typically captured by finite-state automata. This automata-based approach has been developped for several types of systems including systems for integers and reals [BOI 98], pushdown systems [FIN 97] or systems with lossy channels [ABD 96] (see also a similar approach by automatic structures in [BLU 00]); recent developments can be found in [LEG 08]. Regular sets of trees are for instance considered in [BOU 06b] in order to verify programs with dynamic data structures. In section 8.4, we shall illustrate how sets of reachable configurations can be represented by finite-state automata accepting finite words.

### 8.1.5. *Approximations*

An important difficulty in the approach with regular model-checking is the state-explosion problem since the number of states of the built automata (representing sets of configurations) can be huge. This is partly due to the fact that the automata are constructed regardless of the properties to be shown. By contrast, approximation methods may over-approximate the exact set of reachable configurations so that in case of termination, non-reachability can be deduced. The aim is to reduce the verification of such systems to the verification of finite-state systems with the hope using known and efficient methods. For instance, predicate abstraction produces Boolean programs (program variables are Boolean) but the crux of the method relies on the ability to automatically produce a precise enough abstraction that allows the desired property to be checked. Indeed, the inacurracy of the abstraction should not induce the production of spurious counterexamples. The method CEGAR (counter-example guided abstraction refinement) [CLA 00a] aims to automatically derive more and more refined abstractions in order to check the desired properties. Many tools successfully use this methods, including BLAST [HEN 03].

### 8.1.6. *Counter systems*

Despite numerous symbolic representations having been proposed to deal with infinite-state systems (see e.g. timed automata [ALU 94] in Chapter 9), their formal verification remains a difficult problem. Many general formalisms referring to infinite-state systems have an undecidable model-checking problem. Sometimes, decidability can be regained by considering subproblems of the general problem. The class of counter systems is an example of such a formalism. Counter systems have many applications in formal verification. Their ubiquity stems from their use as operational models of numerous infinite-state systems, including for instance broadcast protocols [FIN 02], programs with pointer variables (see [BOU 06a]) and logics for data words. However, numerous model-checking problems for counter systems, such as reachability, are known to be undecidable. Many subclasses of counter systems admit a decidable reachability problem such as reversal-bounded counter automata [IBA 78] and flat counter automata [BOI 98, COM 98, FIN 02]. These two classes of systems admit reachability sets effectively definable in Presburger arithmetic (assuming some additional conditions, unspecified herein). In general, computing the transitive closures of integer relations is a key step to solve verification problems on counter systems, see e.g. [BOZ 10].

In this chapter, we consider

– the class of sequential recursive Petri nets in order to illustrate how recursion can be handled by adapting adequately techniques for Petri nets;

– subclasses of counter systems in order to illustrate the use of Presburger arithmetic to solve verification problems on such systems.

### 8.1.7. *Structure of the chapter*

In section 8.2, we present the class of counter systems that are essentially finite-state automata equipped with program variables (counters) interpreted by non-negative integers. To do so, we first present Presburger arithmetic, since the update functions on counters are governed by constraints expressed in Presburger arithmetic. Several subclasses of counter systems are also introduced, including the vector addition systems with states that are known to be equivalent to Petri nets. The subsequent sections are dedicated to subclasses of counter systems in which verification tasks can be done effectively. In section 8.3, we present the class of recursive Petri nets that extend Petri nets by adding recursion in a controlled way. Verification techniques for this enriched computational model are described by emphasizing how the proof techniques for Petri nets can be indeed extended adequately to this more expressive model. This includes the resolution of the reachability problem as well as the computation of linear invariants. In section 8.4, we introduce subclasses of counter systems for which reachability questions can be solved in Presburger arithmetic viewed as a means to symbolically represent sets of tuples of natural numbers. Unlike section 8.3, the new feature is not recursion but rather the possibility to perform zero-tests (and more sophisticated updates that can be expressed in Presburger arithmetic) but at the cost of making further restrictions, for example on the control graphs. Concluding remarks can be found in section 8.5.

## 8.2. Counter systems

In this section, we present Presburger arithmetic, the class of counter systems as well as remarkable subclasses, including VASS that are known to be equivalent to Petri nets (models of greater practical appeal).

### 8.2.1. *Presburger arithmetic in a nutshell*

Roughly speaking, Presburger arithmetic is the first-order theory of the structure $(\mathbb{N}, +)$ shown decidable in [PRE 29] (which contrasts with Peano arithmetic that also admits multiplication). This logical formalism is used to define sets of tuples of natural numbers. Moreover, it will serve several purposes. Firstly, in the definition of counter systems, Presburger arithmetic is used as a language to define guards and actions (updates on counter values) on transitions. Secondly, each formula from Presburger arithmetic defines a set of tuples (related to the set of assignments that make true the formula) and Presburger arithmetic is therefore a means to represent and symbolically manipulate infinite sets of tuples of natural numbers. Thirdly, formulae from Presburger arithmetic will serve as symbolic representations for semilinear sets (see section 8.3). This section is dedicated to the basics of Presburger arithmetic and to the main properties we shall use in the chapter.

*Basics on tuples of natural numbers*

We write $\mathbb{N}$ [resp. $\mathbb{Z}$] for the set of natural numbers [resp. integers] and $[m, m']$ with $m, m' \in \mathbb{Z}$ to denote the set $\{j \in \mathbb{Z} : m \leq j \leq m'\}$. Given a dimension $n \geq 1$ and $a \in \mathbb{Z}$, we write $\vec{a}$ to denote the vector with all values equal to $a$. For $\vec{x} \in \mathbb{Z}^n$, we write $\vec{x}(1), \ldots, \vec{x}(n)$ for the entries of $\vec{x}$. For $\vec{x}, \vec{y} \in \mathbb{Z}^n$, $\vec{x} \preceq \vec{y} \overset{\text{def}}{\Leftrightarrow}$ for $i \in [1, n]$, we have $\vec{x}(i) \leq \vec{y}(i)$. We also write $\vec{x} \prec \vec{y}$ when $\vec{x} \preceq \vec{y}$ and $\vec{x} \neq \vec{y}$.

*Definition*

Let $\mathrm{VAR} = \{\mathsf{x}, \mathsf{y}, \mathsf{z}, \ldots\}$ be a countably infinite set of *variables*. *Terms* are defined by the grammar $t ::= 0 \mid 1 \mid \mathsf{x} \mid t{+}t$, where $\mathsf{x} \in \mathrm{VAR}$ and 0 and 1 are distinguished constants (interpreted by zero and one respectively). For $k \geq 1$, we write $k\mathsf{x}$ instead of $\mathsf{x} + \cdots + \mathsf{x}$ ($k$ times). Similarly, for $k \geq 1$, we write $k$ instead of $1 + \cdots + 1$ ($k$ times). *Presburger formulae* are defined by the grammar $\varphi ::= t \equiv_k t \mid t < t \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists \mathsf{x}\ \varphi \mid \forall \mathsf{x}\ \varphi$, where $k \geq 2$. The atomic formula $\mathsf{x} \equiv_2 \mathsf{y}$ holds true whenever the difference between $\mathsf{x}$ and $\mathsf{y}$ is even (equality modulo 2). As usual, an occurrence of the variable $\mathsf{x}$ in the formula $\varphi$ is *free* if it does not occur in the scope of either $\exists\mathsf{x}$ or $\forall\mathsf{x}$. Otherwise, the occurrence is *bound*. For instance, in $\mathsf{x}_1 < \mathsf{x}_2$, all the occurrences of the variables are free.

A *valuation* $\mathbf{v}$ is a map $\mathrm{VAR} \to \mathbb{N}$ and it can be extended to the set of all terms as follows: $\mathbf{v}(0) = 0, \mathbf{v}(1) = 1$ and $\mathbf{v}(t + t') = \mathbf{v}(t) + \mathbf{v}(t')$. The satisfaction relation for Presburger arithmetic is equipped with a valuation witnessing that Presburger formulae are interpreted over the structure $(\mathbb{N}, +)$.

– $\mathbf{v} \models t \equiv_k t' \overset{\text{def}}{\Leftrightarrow}$ there is $n \in \mathbb{Z}$ such that $kn + \mathbf{v}(t) = \mathbf{v}(t')$;

– $\mathbf{v} \models t < t' \overset{\text{def}}{\Leftrightarrow} \mathbf{v}(t) < \mathbf{v}(t')$;

– $\mathbf{v} \models \neg\varphi \overset{\text{def}}{\Leftrightarrow} \mathbf{v} \not\models \varphi$;

– $\mathbf{v} \models \varphi \wedge \varphi' \overset{\text{def}}{\Leftrightarrow} \mathbf{v} \models \varphi$ and $\mathbf{v} \models \varphi'$;

– $\mathbf{v} \models \exists\mathsf{x}\ \varphi \overset{\text{def}}{\Leftrightarrow}$ there is $n \in \mathbb{N}$ such that $\mathbf{v}[\mathsf{x} \mapsto n] \models \varphi$ where $\mathbf{v}[\mathsf{x} \mapsto n]$ is equal to $\mathbf{v}$ except that $\mathsf{x}$ is mapped to $n$;

– $\mathbf{v} \models \forall\mathsf{x}\ \varphi \overset{\text{def}}{\Leftrightarrow}$ for every $n \in \mathbb{N}$, we have $\mathbf{v}[\mathsf{x} \mapsto n] \models \varphi$.

As usual, the Boolean connectives $\vee$ (disjunction) and $\Rightarrow$ (implication) can be defined from negation and conjunction in the standard way. Equality between two terms, written $t = t'$, can be expressed by $\neg(t < t' \vee t' < t)$. Similarly, we write $t \leq t'$ to denote the formula $(t = t') \vee (t < t')$. Observe also that $t \equiv_k t'$ is equivalent to $\exists\ \mathsf{x}\ (t = k\mathsf{x} + t' \vee t' = k\mathsf{x} + t)$ ($\mathsf{x}$ is a variable that does not occur in $t$ and $t'$). We invite the reader to check that $0$ and $1$ can be removed from the above definitions without changing the expressive power of the formulae.

In the chapter, we assume that the variables in $\mathrm{VAR}$ are linearly ordered by their indices. So, any valuation restricted to $n \geq 1$ variables can be viewed as a tuple in $\mathbb{N}^n$. Any formula with $n \geq 1$ free variables $\mathsf{x}_1, \ldots, \mathsf{x}_n$ defines a set of $n$-tuples ($n$-ary relation) as follows:

$$\mathrm{REL}(\varphi) \overset{\text{def}}{=} \{(\mathbf{v}(\mathsf{x}_1), \ldots, \mathbf{v}(\mathsf{x}_n)) \in \mathbb{N}^n : \mathbf{v} \models \varphi\}.$$

For instance, $\mathrm{REL}(\mathsf{x}_1 < \mathsf{x}_2) = \{(n, n') \in \mathbb{N}^2 : n < n'\}$. Similarly, the set of odd natural numbers can be defined by the formula $\exists\mathsf{y}\ \mathsf{x} = \mathsf{y} + \mathsf{y} + 1$. A set $X \subseteq \mathbb{N}^n$ is said to be *Presburger-definable* iff there is a Presburger formula $\varphi$ such that $X = \mathrm{REL}(\varphi)$. Sets that are Presburger-definable are known to correspond exactly to semilinear sets [GIN 66].

A formula $\varphi$ is *satisfiable* (in Presburger arithmetic) whenever there is a valuation $\mathbf{v}$ such that $\mathbf{v} \models \varphi$. Similarly, a formula $\varphi$ is *valid* (in Presburger arithmetic) when for all valuations $\mathbf{v}$, we have $\mathbf{v} \models \varphi$. When $\varphi$ has no free variables, satisfiability and validity are equivalent notions. Two formulae are *equivalent* (in Presburger arithmetic) whenever they define the same set of tuples.

**THEOREM 8.1** *[PRE 29] (I) The satisfiability problem for Presburger arithmetic is decidable. (II) Every Presburger formula is equivalent to a Presburger formula without first-order quantification.*

Theorem 8.1(II) takes advantage of atomic formulae of the form $t \equiv_k t'$ that contain an implicit quantification. Removing atomic formulae of the form $t \equiv_k t'$ does not change the expressive power but the equivalence in theorem 8.1(II) would not hold in that case. Moreover, in (II) above, the equivalent formula can be effectively built witnessing the quantifier elimination property. In subsequent developments, we mainly consider quantifier-free Presburger formulae, which does not restrict the expressive power but may modify complexity issues. It is worth noting that the first-order theory of $(\mathbb{N}, \times)$ is decidable too (known as *Skolem arithmetic*) whereas the first-order theory of $(\mathbb{N}, \times, +)$ is undecidable (see e.g. [TAR 53]). Observe also that $(\mathbb{Z}, <, +)$ is decidable [PRE 29] as well as the first-order theory of $(\mathbb{R}, \times, +)$ [TAR 51].

Satisfiability problem for Presburger arithmetic can be solved in triple exponential time [OPP 78] by analyzing the quantifier elimination procedure described in [COO 72]. Besides, satisfiability problem for Presburger arithmetic is shown 2EXPTIME-hard in [FIS 74] and in 2EXPSPACE in [FER 79]. An exact complexity charaterization is provided in [BER 80] (double exponential time on alternating Turing machines with linear amount of alternations). Due to the wide range of applications for Presburger arithmetic, computational complexity of numerous fragments has been also characterized, see e.g. [GRÄ 88]. Moreover, its restriction to quantifier-free formulae is NP-complete [PAP 81] (see also [BOR 76]).

### 8.2.2. *Classes of counter systems*

A counter system $\mathcal{S}$ is defined below as a finite-state automaton equipped with counters, i.e. program variables interpreted by non-negative integers. In full generality, the counters are governed by constraints that can be expressed by Presburger formulae (this generality is mainly useful for section 8.4.1). Minsky machines [MIN 67] form a special class of counter systems and therefore most interesting problems on counter systems happen to be undecidable since Minsky machines can simulate Turing machines [MIN 67]. However, we shall study important subclasses of counter systems for which decidability can be regained for various decision problems.

**DEFINITION 8.2** *A* counter system $\mathcal{S} = (Q, n, \delta)$ *is a structure such that*

*– $Q$ is a nonempty finite set of* control states *(a.k.a.* locations*);*

*– $n \geq 1$ is the* dimension *of the system, i.e. the number of counters; we assume that the counters are represented by the variables $\mathsf{x}_1, \ldots, \mathsf{x}_n$;*

– $\delta$ *is the* transition relation *defined as a finite set of triples of the form* $(q, \varphi, q')$*, where* $q, q'$ *are control states and* $\varphi$ *is a Presburger formula whose free variables are among* $\mathsf{x}_1, \ldots, \mathsf{x}_n, \mathsf{x}'_1, \ldots, \mathsf{x}'_n$.

Elements $t = (q, \varphi, q')$ are called *transitions* and are often represented by $q \xrightarrow{\varphi} q'$. As usual, by convention, prime variables are intended to be interpreted as the next values of the unprimed variables. Moreover, observe that a counter system has no initial control state and no final control state but in the chapter we shall introduce such control states on demand. It is certainly possible to propose an alternative definition without control states and to encode them by new counters, for instance. However, when infinite-state transition systems arise in the modeling of computational processes, there is often a natural factoring of each system state into a control component and a memory component, where the set of control states is typically finite.

Figure 8.1 contains a counter system (augmented with an initial control state). It is related to the famous Collatz problem. The role of control state $q_0$ is to compute an arbitrary counter value before reaching the control state $q_1$. At the control state $q_1$, if the counter value is even, then divide by two the counter value. Otherwise, multiply by 3 and add 1. It is open whether whenever the system enters in the control state $q_1$, eventually it reaches the counter value 1.

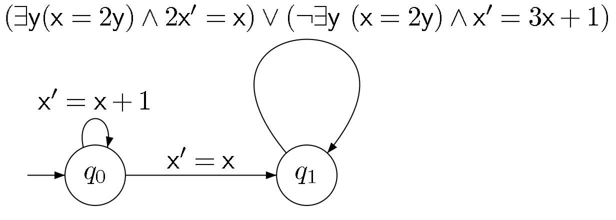


**Figure 8.1.** *An example of counter system*

The class of counter systems is quite general and it very often makes sense to label the transitions by Presburger formulae that can be decomposed by a *guard* (constraints on the current counter values) and an *update function* (constraints on the way the new counter values are computed from the previous ones).

A *configuration* of the counter system $\mathcal{S} = (Q, n, \delta)$ is defined as a pair $(q, \vec{x}) \in Q \times \mathbb{N}^n$. Given two configurations $(q, \vec{x})$, $(q', \vec{x'})$ and a transition $t = q \xrightarrow{\varphi} q'$, we write $(q, \vec{x}) \xrightarrow{t} (q', \vec{x'})$ whenever $\mathbf{v}_{\vec{x},\vec{x'}} \models \varphi$ and for $i \in [1, n]$, $\mathbf{v}_{\vec{x},\vec{x'}}(\mathsf{x}_i) \overset{\text{def}}{=} \vec{x}(i)$ and $\mathbf{v}_{\vec{x},\vec{x'}}(\mathsf{x}'_i) \overset{\text{def}}{=} \vec{x'}(i)$. The operational semantics of counter systems updates configurations, and runs of such systems are essentially sequences of configurations. Every counter system $\mathcal{S} = (Q, n, \delta)$ induces a (possibly infinite) graph made of configurations. Indeed, all the interesting problems on counter systems can be formulated on its *transition system*.

**DEFINITION 8.3** *Given a counter system $\mathcal{S} = (Q, n, \delta)$, its* transition system $T(\mathcal{S}) = (S, \rightarrow)$ *is a graph such that $S = Q \times \mathbb{N}^n$ and $\rightarrow \subseteq S \times S$ such that $((q, \vec{x}), (q', \vec{x'})) \in \rightarrow$ $\overset{\text{def}}{\Leftrightarrow}$ there exists a transition $t \in \delta$ such that $(q, \vec{x}) \xrightarrow{t} (q', \vec{x'})$. As usual, $\xrightarrow{*}$ denotes the reflexive and transitive closure of the binary relation $\rightarrow$.*

Given a counter system $\mathcal{S}$, a *run* $\rho$ is a non-empty (possibly infinite) sequence $\rho = (q_0, \vec{x_0}), \ldots, (q_k, \vec{x_k}), \ldots$ of configurations such that two consecutive configurations are in the relation $\rightarrow$ from $T(\mathcal{S})$. $(q_0, \vec{x_0})$ is called the *initial* configuration of $\rho$.

*Standard decision problems*

Below, we enumerate a list of standard decision problems for counter systems. They are mainly related to reachability questions. The list is certainly not exhaustive but it contains the main problems related to verification and model-checking.

Reachability problem:

– *Input*: a counter system $\mathcal{S}$ and two configurations $(q, \vec{x})$ and $(q', \vec{x'})$;

– *Question*: is there a finite run with initial configuration $(q, \vec{x})$ and final configuration $(q', \vec{x'})$?

Control state reachability problem:

– *Input*: a counter system $\mathcal{S}$, a configuration $(q, \vec{x})$ and a control state $q_f$;

– *Question*: is there a finite run with initial configuration $(q, \vec{x})$ and whose final configuration has control state $q_f$?

Covering problem:

– *Input*: a counter system $\mathcal{S}$ and two configurations $(q, \vec{x})$ and $(q', \vec{x'})$;

– *Question*: is there a finite run with initial configuration $(q, \vec{x})$ and whose final configuration is $(q', \vec{x''})$ with $\vec{x'} \preceq \vec{x''}$?

Boundedness problem:

– *Input*: a counter system $\mathcal{S}$ and a configuration $(q, \vec{x})$;

– *Question*: is the set $\{(q', \vec{x'}) \in Q \times \mathbb{N}^n : (q, \vec{x}) \xrightarrow{*} (q', \vec{x'})\}$ finite?

Termination problem:

– *Input*: a counter system $\mathcal{S}$ and a configuration $(q, \vec{x})$;

– *Question*: is there an infinite run with initial configuration $(q, \vec{x})$?

Control state repeated reachability problem:

– *Input*: a counter system $\mathcal{S}$, a configuration $(q, \vec{x})$ and a control state $q_f$;

– *Question*: is there an infinite run with initial configuration $(q, \vec{x})$ such that the control state $q_f$ is repeated infinitely often?

Most standard verification problems on counter systems reduce to one of the above mentioned decision problems. For instance, model-checking over linear-time temporal logic (LTL) in which atomic formulae are restricted to control states amounts to questions on control state repeated reachability problem. Hence, designing algorithms for such decision problems can be helpful for instance to verify computer systems such as programs with pointers [BOU 06a], broadcast protocols [ESP 99], or replicated finite-state programs [KAI 10] (Boolean programs with a finite set of configurations but can be executed by an unknown number of threads), to quote but a few.

In the forthcoming subsections, we introduce several subclasses of counter systems by restricting the general definition provided above. Additional requirements can be of distinct nature: restriction on syntactic ressources (number of counters, Presburger formulae etc.), restriction on the control graph (e.g. flatness), and semantical restrictions (reversal-boundedness, etc.)

*Succinct counter automata*

In the chapter, we adopt the convention that a counter automaton is a counter system in which the instructions are either zero-tests, increments, or decrements, possibly encoded succinctly. A *succinct counter automaton* is a counter system $(Q, n, \delta)$ in which the transitions are of the form either $q \xrightarrow{\mathrm{inc}(\vec{b})} q'$ with $\vec{b} \in \mathbb{Z}^n$ or $q \xrightarrow{\mathrm{zero}(\vec{b'})} q'$ with $\vec{b'} \in \{0,1\}^n$ where:

– $\mathrm{inc}(\vec{b})$ is a shortcut for $\bigwedge_{i \in [1,n]} \mathsf{x}'_i = \mathsf{x}_i + \vec{b}(i)$;

– $\mathrm{zero}(\vec{b'})$ is a shortcut for $\bigwedge_{i \in [1,n] \text{ s.t. } \vec{b'}(i)=1} \mathsf{x}_i = 0 \wedge \bigwedge_{i \in [1,n]} \mathsf{x}'_i = \mathsf{x}_i$ (as usual, empty conjunction is understood as $\top$).

In succinct counter automaton, each transition either performs zero-tests on a subset of counters or updates counters by adding a vector in $\mathbb{Z}^n$. All the counters are tested or updated simultaneously.

*Standard counter automata*

A *standard counter automaton* is a counter system $(Q, n, \delta)$ in which the transitions are of the form either $q \xrightarrow{\mathrm{inc}(i)} q'$ or $q \xrightarrow{\mathrm{dec}(i)} q'$ or $q \xrightarrow{\mathrm{zero}(i)} q'$ $(i \in [1, n])$ where

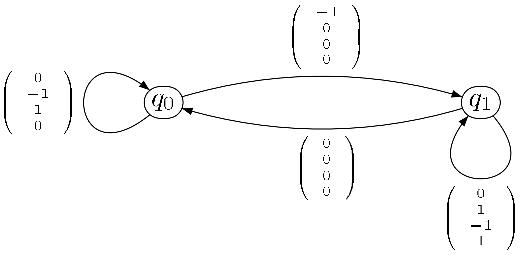
– $\mathrm{inc}(i)$ is a shortcut for $(\mathsf{x}'_i = \mathsf{x}_i + 1) \wedge (\bigwedge_{j \neq i} \mathsf{x}'_j = \mathsf{x}_j)$ (also written $\mathsf{x}_i$++);

– $\mathrm{dec}(i)$ is a shortcut for $(\mathsf{x}'_i = \mathsf{x}_i - 1) \wedge (\bigwedge_{j \neq i} \mathsf{x}'_j = \mathsf{x}_j)$ (also written $\mathsf{x}_i$- -);
– $\mathrm{zero}(i)$ is a shortcut for $(\mathsf{x}_i = 0) \wedge (\bigwedge_j \mathsf{x}'_j = \mathsf{x}_j)$ (also written $\mathsf{x}_i = 0?$).

By contrast to succinct counter automata, transitions in standard counter automata can perform a simple operation at once (otherwise, a succession of transitions is needed). Indeed, standard counter automata and succinct counter automata are very similar but when it comes to complexity issues, exponential blow-up may occur when passing from one model to another. In the sequel, unless otherwise stated, by a counter automaton we mean a standard one. It is easy to check that Minsky machines (with two counters) form a subclass of standard counter automata.

8.2.2.1. *Vector addition systems with states*

A *vector addition system with states* (VASS) [KAR 69] is a succinct counter automaton without zero-tests, i.e. all the transitions are of the form $q \xrightarrow{\mathrm{inc}(\vec{b})} q'$ with $\vec{b} \in \mathbb{Z}^n$. In the sequel, a VASS is represented by a tuple $\mathcal{V} = (Q, n, \delta)$ where $Q$ is the finite set of control states and $\delta$ is a finite subset of $Q \times \mathbb{Z}^n \times Q$. Standard counter automata can be naturally viewed as VASS augmented with zero-tests by simulating transitions of the form $q \xrightarrow{\vec{b}} q'$ by sequences of increments and decrements. Additionally, VASS are known to be equivalent to Petri nets that are models of greater practical appeal. Figure 8.2 presents an example of VASS. One can show that for all $\vec{x} \in \mathbb{N}^4$, the set $\{\vec{y} \in \mathbb{N}^4 : (q_0, \vec{x}) \xrightarrow{*} (q_0, \vec{y})\}$ is finite. Moreover, for $\vec{x} \in \mathbb{N}^2$, $\{k \in \mathbb{N} : k \leq \vec{x}(1) \times \vec{x}(2)\} = \{\vec{y}(4) : (q_0, \vec{y_0}) \xrightarrow{*} (q_0, \vec{y}),\ \vec{y_0}(1) = \vec{x}(1),\ \vec{y_0}(2) = \vec{x}(2),\ \vec{y_0}(3) = \vec{y_0}(4) = 0\}$.



$\begin{pmatrix} -1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$
$\begin{pmatrix} 0 \\ -1 \\ 1 \\ 0 \end{pmatrix}$
$q_0$
$q_1$
$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$
$\begin{pmatrix} 0 \\ 1 \\ -1 \\ 1 \end{pmatrix}$

**Figure 8.2.** *A VASS weakly computing multiplication*

**THEOREM 8.4** *(see e.g. [MAY 84, KOS 82]) The reachability problem for VASS is decidable.*

The exact complexity of the reachability problem is open: we know it is EXPSPACE-hard [LIP 76] and no primitive recursive upper bound exists. By contrast, the covering problem and boundedness problems seem easier since they are EXPSPACE-complete [LIP 76, RAC 78]. Observe also that the covering problem can express the thread-state reachability problem for replicated finite-state programs, see e.g. [KAI 10]. Similarly, the boundedness problem for asynchronous programs has been considered in [GAN 09].

8.2.2.2. *Relationships with Petri nets*

In this section, we briefly recall how Petri nets (see e.g. [REI 98]) are related to VASS . First, let us recall that a *Petri net* $N$ is a structure $(P, T, W^-, W^+, m_I)$ such that $P$ is a finite set of *places*, $T$ is a finite set of *transitions*, $W^- : (P \times T) \to \mathbb{N}$ and $W^+ : (P \times T) \to \mathbb{N}$ are *weight functions*. A *marking* $m$ is a map of the form $P \to \mathbb{N}$: for each place, we specify a number of *tokens* (possibly none). In the Petri net $N$, $m_I : P \to \mathbb{N}$ is the initial marking (initial distribution of tokens). We assume that the reader is familiar with the semantics of this model (otherwise see e.g. [PET 81, REI 98]). We just recall below a few definitions. A transition $t \in T$ is $m$*-enabled*, written $m \xrightarrow{t}$, whenever for all places $p \in P$, $m(p) \geq W^-(p,t)$. An $m$-enabled transition $t$ may fire and produce the marking $m'$, written $m \xrightarrow{t} m'$, with for all places $p \in P$, $m'(p) = m(p) - W^-(p,t) + W^+(p,t)$. A marking $m'$ is *reachable* from $m$ whenever there is a sequence of the form $m_0 \xrightarrow{t_0} m_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{k-1}} m_k$ with $m_0 = m$ and $m_k = m'$ (also written $m \xrightarrow{t_0 \cdots t_{k-1}} m'$).

Here are standard problems for Petri nets.

Reachability problem for Petri nets:

– ***Input***: a Petri net $(P, T, W^-, W^+, m_I)$ and a marking $m$;

– ***Question***: is $m$ reachable from $m_I$?

Covering problem for Petri nets:

– ***Input***: a Petri net $(P, T, W^-, W^+, m_I)$ and a marking $m$;

– ***Question***: is there a marking $m'$ reachable from $m_I$ such that for all $p \in P$, we have $m'(p) \geq m(p)$?

Boundedness problem for Petri nets:

– ***Input***: a Petri net $(P, T, W^-, W^+, m_I)$;

– ***Question***: is the set of markings reachable from $m_I$ infinite?

Petri nets and VASS are known to be equivalent models as far as the reachability problem, the covering problem, and the boundedness problem are concerned, see e.g. [REU 90]. By way of example, let us show how to simulate a VASS using a Petri net. Let $\mathcal{V}$ be a VASS $(Q, n, \delta)$ and $(q_I, \vec{x_I})$ be a configuration. We can build a Petri net $N_{\mathcal{V}}$ that simulates $\mathcal{V}$, by using a standard translation from VASS to Petri nets.

For every control state $q$ in $\mathcal{V}$, we introduce a place $p_q$ in $N_{\mathcal{V}}$ and for $i \in [1, n]$, we introduce a place $p_i$. An initial marking $m_I$ contains one token in the place $p_{q_I}$ and for $i \in [1, n]$, $m_I(p_i) = \vec{x_I}(i)$. From this marking, we only obtain markings where a unique token belongs to a place of the form $p_q$ ($q \in Q$) which means that a unique control state is active for every marking. For every transition in $\mathcal{V}$, say $t = q \xrightarrow{\vec{b}} q'$, we consider a transition $\mathbf{t}$ in $N_{\mathcal{V}}$ that consumes a token in $p_q$, produces a token in $p_{q'}$ and produces [respectively consumes] $\vec{b}(i)$ tokens in the place $p_i$ when $\vec{b}(i) \geq 0$ [respectively when $\vec{b}(i) < 0$].

In the rest of this chapter, we shall consider two types of extensions of Petri nets, namely recursive Petri nets in section 8.3; which adds recursion to Petri nets and subclasses of counter systems in which zero-tests are allowed in section 8.4 (presented as extensions of VASS with zero-tests or with more complex Presburger-definable update functions). Methods to handle verification problems for such extensions will be of a different nature. As far as recursive Petri nets are concerned, extensions of techniques for standard Petri nets are considered. In contrast, in section 8.4, the decidability of verification problems is established by reduction into the satisfiability problem for Presburger arithmetic.

## 8.3. Recursive Petri nets

The recursive Petri net formalism (RPN) has been introduced to model dynamic systems for which the creation of processes (or threads) is required [El 95, El 96]. In this formalism, a process is characterized by a Petri net in which particular transitions (called *abstract*) allow the creation of processes and sets of markings from which the process is allowed to terminate. More precisely, termination of a process is possible when the current marking belongs to a given semilinear set, i.e. to a set defined by a Presburger formula. All the processes share the same control structure (i.e. the same RPN) but the initial marking of each process depends on the abstract transition which has created it. In consequence, the state of a process is completely characterized by a marking (a distribution of tokens over places) of the RPN and by the abstract transition that has created the process. Parallelism, which is a fundamental feature of Petri nets, is allowed between a process and its father. Termination of a process induces the update of the current marking of the father process (mainly by adding tokens to places). In this way, the relationships between processes and their fathers is encoded in the global state of the RPN. Hence, the global state of a recursive Petri net can be viewed as a finite tree whose nodes are markings (in the usual sense for Petri nets) and each edge is labeled by the abstract transition that has created the child process.

It has been shown in different papers [HAD 99, HAD 00, HAD 01] that RPN form a strict extension of Petri nets in terms of expressive power. It has also been shown that the reachability problem remains decidable for RPN. It is important to note that this property is essential for the design of verification tools. However, in [HAD ], the authors have established that the verification of linear-time temporal formulae is an undecidable problem whereas it is decidable for the standard class of Petri nets (limited to event-based formulae).

In this section, we focus on a particular subclass of RPN called *sequential recursive Petri nets* (SRPN) [HAD 01]. In such a net, a father process creating a child process is blocked until the child process terminates. Consequently, parallelism is only allowed within the unique active process, if any. The global state of a sequential recursive Petri net can be viewed as a finite stack of processes, the topmost process being the active one. Moreover, each process can be represented by a marking (in the usual sense for Petri nets) and by the abstract transition that has created it (if any), the father process being just below on the stack. This formalism is less expressive than RPN but it is a strict extension of Petri nets. Moreover, the verification of (event-based) linear-time temporal formulae remains a decidable problem for SRPN.

### 8.3.1. *Definitions*

Similar to an ordinary Petri net, a SRPN has *places* and *transitions*. The transitions are split into two categories: *elementary transitions* and *abstract transitions*.

The semantics of such a net may be informally explained as follows. In an ordinary net, a process plays the token game by firing a transition and by updating the current marking (its internal state). In an SRPN, there is a stack of processes (each one with its current marking) where the only active process is on top of the stack. A *step* of an SRPN is thus a step of this process. The enabling rule of the transitions is specified by the *backward incidence matrix*.

When a process fires an elementary transition, it consumes the tokens specified by the backward incidence matrix and produces tokens defined by the *forward incidence matrix* (as in ordinary Petri nets).

When a process fires an abstract transition, it consumes the tokens specified by the backward incidence matrix and creates a new process (called its son) put on top of the stack which consequently becomes the active process. Such a process begins its token game with an *initial marking* that depends on the abstract transition.

Termination of an active process is possible when the current marking belongs to a given set of markings that is effectively semilinear. Herein, effective semilinearity is guaranteed by using Presburger formulae whose free variables refer to places of the net (see section 8.2.1). So, a family of effective representations of semilinear sets of *final markings* is defined in order to describe the termination of processes. This family is indexed by a finite set whose items are called *termination indices*. When a process reaches a final marking, it may terminate its token game (i.e. it is popped out of the stack). Then, it produces in the token game of its father (the new top of the stack) and for the abstract transition which created it, the tokens specified by the forward incidence matrix. Unlike ordinary Petri nets, this matrix depends also on the termination index of the semilinear set which the final marking belongs to. Such a firing is called a *cut step* (or equivalently a *return step*). When a cut step occurs in a stack reduced to a single process, this results to the empty stack.

The next definitions are helpful to define what are the configurations (global states) of an SRPN. Below, such configurations are called *extended markings*.

**DEFINITION 8.5 (SRPN)** *An SRPN is defined by a tuple* $N = \langle P, T, I, W^-, W^+, \mathtt{Init}, \Upsilon\rangle$ *where:*

– $P = \{p_1, \ldots, p_\alpha\}$ *is a finite set of places;*

– $T$ *is a finite set of transitions such that* $P \cap T = \emptyset$*;*

– *A transition of* $T$ *can be either elementary or abstract. The sets of elementary and abstract transitions are respectively denoted by* $T_{el}$ *and* $T_{ab}$*;*

– $I$ *is a finite set of indices;*

– $W^-$ *is the pre function defined from* $P \times T$ *to* $\mathbb{N}$*;*

– $W^+$ *is the post function defined from* $P \times [T_{el} \cup (T_{ab} \times I)]$ *to* $\mathbb{N}$*;*

– $\mathtt{Init}$ *is a labeling function* $T_{ab} \to (P \to \mathbb{N})$ *which associates an ordinary marking called the starting marking of* $t$ *with each abstract transition;*

– $\Upsilon = (\varphi_i)_{i \in I}$ *is a family of Presburger formulae with free variables among* $\mathsf{x}_1$, $\ldots$, $\mathsf{x}_\alpha$. *Each formula* $\varphi_i$ *defines the set of ordinary markings* $\{m : P \to \mathbb{N} \mid \mathbf{v} \models \varphi_i, \text{ for } 1 \leq j \leq \alpha, \ \mathbf{v}(\mathsf{x}_j) = m(p_j)\}$. *By abuse of notation, we say that a marking* $m$ *belongs to* $\mathrm{REL}(\varphi_i)$ *to mean that it belongs to the above set. For instance, '*$\mathsf{x}_1 = \mathsf{x}_2$*' would be interpreted as a symbolic way to represent the set of markings such that the number of tokens in the place* $p_1$ *is equal to the number of tokens in the place* $p_2$.

In the chapter, we distinguish two kinds of markings. As usual for Petri nets, an *ordinary marking* is a map from $P$ to $\mathbb{N}$. By contrast, an *extended marking* corresponds to a global state of the SRPN that can be viewed as a finite stack of ordinary markings augmented with abstract transitions. When no confusion is possible, we simply use the term "marking".

**DEFINITION 8.6 (Extended marking)** *An* extended marking $em$ *for a SRPN* $N = \langle P, T, I, W^-, W^+, \mathtt{Init}, \Upsilon\rangle$ *is either the empty stack* $\perp$ *or a sequence* $(m_d, t_d), \ldots, (m_2, t_2), m_1$ *for some* $d \geq 1$ *such that*

– $d$ *is the* depth *of* $em$ *and* $\{1, \ldots, d\}$ *is the set of* levels*;*

*– $m_1, \ldots, m_d$ are ordinary markings;*

*– $t_2, \ldots, t_d$ are abstract transitions. The intention is that a process $(m_i, t_i)$ has been created by the abstract transition $t_i$ and its current marking is $m_i$. Whenever $d \geq 2$, the* active *process is $(m_d, t_d)$. When $d = 1$, the* active *process is $m_1$.*

Obviously, an extended marking different from $\perp$ can be viewed as a tree reduced to a single path where nodes are indexed by levels and labeled by ordinary markings and edges are labeled by abstract transitions. The root of this tree corresponds to the bottom of the stack and the single leaf to its top (for instance, see Figure 8.6). A *marked* SRPN $(N, em_0)$ is an SRPN $N$ together with an initial extended marking $em_0$. According to the presentation, the size of the stack corresponding to an extended marking $em$ is $d$ and the ordinary markings associated with the processes of the stack are $m_1, \ldots, m_d$. Since the effect of cut steps depends on the abstract transition which created a process, the transitions $t_2, \ldots, t_d$ are stored in the extended marking.

Given a SRPN, we can define its corresponding *reachability graph* whose nodes are made of extended markings and edges are labelled by *actions*. Three types of actions can be distinguished:

1) firing an elementary transition $t \in T \setminus T_{ab}$, corresponding to an *internal step*;

2) firing an abstract transition $t \in T_{ab}$, corresponding to a *gosub step*;

3) terminating the active process with the current marking in $\mathrm{REL}(\varphi_i)$ for some $i \in I$ and possibly returning to the father process, corresponding to a *return step* (also called *cut step*).

An *elementary step* corresponds to performing a single action. Hence, the semantics of a SRPN, completely determined by its corresponding reachability graph, is based on the notions of *enabled actions* and *firing of steps*.

**DEFINITION 8.7** *Let $em = (m_d, t_d), \ldots, (m_2, t_2), m_1$ be an extended marking (different from $\perp$) for the SRPN $N$.*

*1) A transition $t \in T$ is* enabled *in $em$, denoted by $em \xrightarrow{t}$ iff for all $p \in P$, we have $m_d(p) \geq W^-(p, t)$;*

*2) Return step $\tau_i$ with $i \in I$ is* enabled *in $em$, denoted by $em \xrightarrow{\tau_i}$, iff $m_d \in \mathrm{REL}(\varphi_i)$.*

**DEFINITION 8.8** *Let $em = (m_d, t_d), \ldots, (m_2, t_2), m_1$ and $em' = (m'_{d'}, t'_{d'}), \ldots, (m'_2, t'_2), m'_1$ be two extended markings for the SRPN $N$.*

*– For $t \in T \setminus T_{ab}$, $em \xrightarrow{t} em'$ iff*

*1) $t$ is enabled in $em$,*

*2) $d = d'$ and, $em$ and $em'$ differ at most on their active process,*
*3) $t_d = t'_d$ and for all $p \in P$, $m'_d(p) = m'_d(p) - W^-(p,t) + W^+(p,t)$;*

– *For $t \in T_{ab}$, $em \xrightarrow{t} em'$ iff*
*1) $t$ is enabled in $em$,*
*2) $d' = d + 1$,*
*3) $(m'_{d'}, t'_{d'}) = (\mathtt{Init}(t), t)$,*
*4) for all $p \in P$, $m'_d(p) = m_d(p) - W^-(p,t)$,*
*5) $t_d, (m_{d-1}, t_{d-1}), \ldots, (m_2, t_2), m_1 = t'_d, (m'_{d-1}, t'_{d-1}), \ldots, (m'_2, t'_2), m'_1$;*

– *For $i \in I$, $em \xrightarrow{\tau_i} em'$ iff*
*1) $\tau_i$ is enabled in $em$,*
*2) $d' = d - 1$,*
*3) if $d' \geq 1$, then for all $p \in P$, $m'_{d'}(p) = m_{d'}(p) + W^+(p, t_d, i)$,*
*4) We have the equality below:*

$$t_{d'}, (m_{d'-1}, t_{d'-1}), \ldots, (m_2, t_2), m_1 = t'_{d'}, (m'_{d'-1}, t'_{d'-1}), \ldots, (m'_2, t'_2), m'_1.$$

A *firing sequence* $\sigma$ is a sequence of the form $em_1 \xrightarrow{a_1} em_2 \xrightarrow{a_2} em_3 \cdots \xrightarrow{a_{n-1}} em_n$ such that for every $i \in [1, n-1]$, $em_i \xrightarrow{a_i} em_{i+1}$ is a single step. In the sequel, for the sake of simplicity, $\sigma$ will be often denoted by $\sigma = a_1 \ldots a_{n-1}$. When multiple SRPNs are involved, we denote by $em_1 \xrightarrow{\sigma}_N em_n$ a firing sequence $\sigma$ in an SRPN $N$. In a marked SRPN $(N, em_0)$, an extended marking $em$ is *reachable* iff there exists a firing sequence $em_0 \xrightarrow{\sigma}_N em$. The *reachability graph* of $(N, em_0)$ is defined as follows: the nodes are the reachable extended markings and the edges are labeled by actions.

### 8.3.2. *Expressive power*

This section illustrates both the syntax and the semantics of SRPN with the help of relevant examples. Furthermore, we simultaneously demonstrate the expressive power of the model and its suitability with respect to standard discrete event system patterns [CAS 99].

*Modeling of interrupts and exceptions.* SRPNs are illustrated by integrating an interrupt mechanism step by step in an existing net. Then, the treatment of exceptions is added. The SRPN modeling the abstraction of the original system is given in Figure 8.3. This net can be understood as a standard Petri net and it can realize a unique behavior where the transition $t_{correct}$ is systematically fired. It is important to note that any Petri net can be interpreted as a SRPN without abstract transitions and extended markings are reduced to ordinary markings.

When an (software or hardware) interrupt is handled, the context of the system must be saved and the control given to the interrupt handler. If different levels of

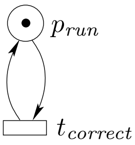


**Figure 8.3.** *An abstraction of a system*

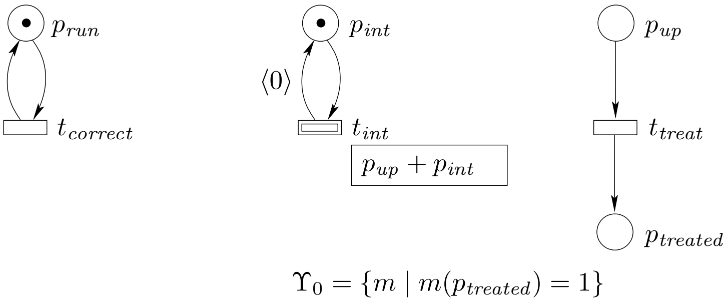


**Figure 8.4.** *Integration of an interrupt mechanism*

interrupts have to be taken into account, an interrupt can mask some others. Moreover, when the handler returns, the execution context must be restored.

The SRPN presented in Figure 8.4 adds an interrupt mechanism to the net from Figure 8.3. Contrary to ordinary nets, SRPNs are often disconnected since each connected component may be activated by the firing of different abstract transitions. As the initial extended marking is reduced to a single node, we have directly described the ordinary marking associated with this node by putting tokens in places. We will follow the same convention in the sequel. The new elements are:

– the transition $t_{int}$: this transition is an abstract one (represented by a double line rectangle) and represents the trigger point of the interrupt. Here, we consider that the interrupt can always occur and in consequence, the unique input place ($p_{int}$) of the abstract transition is initially marked. The firing of $t_{int}$ creates a new process on top of the stack and freezes the execution of the father process until the child process terminates. The child process starts its execution with the marking $p_{up} + p_{int}$ (indicated in the frame near the abstract transition). Because, $p_{int}$ will be still marked, a new interrupt may occur during its execution. The firing of $t_{int}$ consumes also a token from its input place $p_{int}$ but produces no token. This step is delayed until the end of the new process and depends on its type (index) of termination;

– the set $\Upsilon_0$: In this example, a unique set characterizes the final markings from which a return step is possible. $\Upsilon_0$ represents the set of markings where the interrupt has been treated (the place $p_{treated}$ is marked). The index 0 associated with this set is used to indicate the type of termination (here, there is a unique type);

– the valuation $\langle 0 \rangle$ labels the arc linking the transition $t_{int}$ to the place $p_{int}$: when a process terminates (its current marking should belong to $\Upsilon_0$), the marking of the father process is modified depending on the index of the semilinear sets used for termination of the active child process. Then, the post-condition of an abstract transition is conditioned by the index associated to the termination set. In this example, the firing of a cut step will always produce a token in the place $p_{int}$.

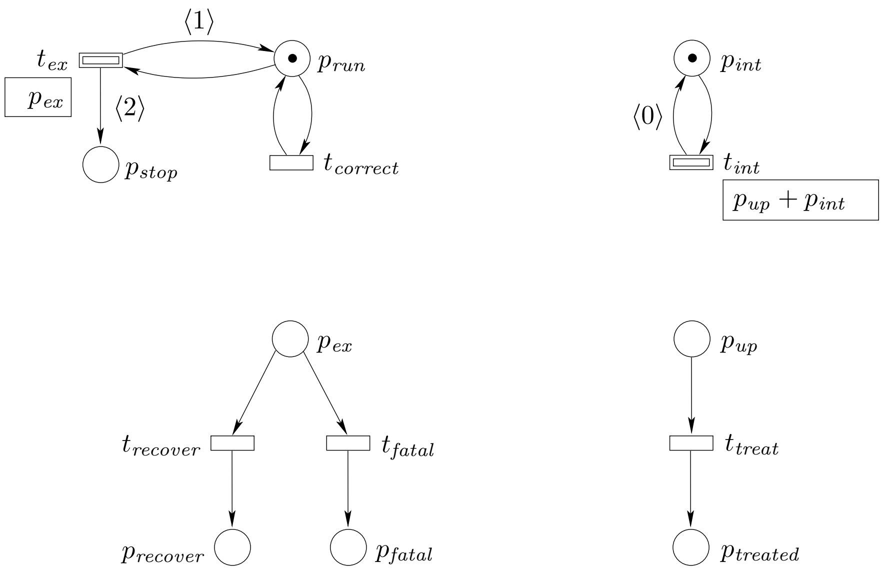
The initial extended marking of the SRPN of Figure 8.4 is composed of a unique process in which only the places $p_{run}$ and $p_{int}$ both contain a unique token. It is clear that this net can reach an infinite number of configurations. Indeed, taking into account an interrupt does not mask those that can occur during its handling (the place $p_{int}$ is marked in the starting marking of $t_{int}$). We can remark that this infinite state space can occur whatever the boundedness of the places (all the places may contain at most a unique token).

The integration of a mechanism for exceptions is illustrated in Figure 8.5. In this example, an exception can occur (the firing of the abstract transition $t_{ex}$) only if the main process runs (witnessed by the presence of a token on the place $p_{run}$). Its treatment can recover the error (place $p_{recover}$) or leads to a fatal situation (place $p_{fatal}$). The two termination sets $\Upsilon_1$ and $\Upsilon_2$ allow to distinguish both cases and the output arcs of the abstract transition $t_{ex}$ are labeled consecutively.

Modularity is a natural feature of SRPNs. Indeed, if the system does not include the interrupt mechanism, the designer simply deletes the corresponding elements in the figure.

Figure 8.6 presents a subgraph of the reachability graph of the SRPN. The right part of the figure illustrates the interrupts and the left the exceptions. Interrupt cannot occur during the treatment of exception. On the contrary, if an exception leads to a fatal situation and when this error has been transmitted to the main process, the behavior of this last process will be limited to the treatment of interrupts (the unique enabled transition – not represented in the figure – from the state where the main process reaches the marking $p_{stop} + p_{int}$, is $t_{int}$).

The example of Figure 8.5 shows the ability of SRPN to implicitly keep the context of suspended processes. This kind of formalization in terms of Petri net requires an explicit representation of each context. By using this feature, it has been shown in [HAD 00] that SRPN include the family of algebraic languages. On the other hand, it has been also shown that the language of palindromes (a well-known algebraic language) cannot be recognized by a (labeled) Petri net [JAN 79]. Thus, the family of



$$\Upsilon_1 = \{m \mid m(p_{recover}) = 1\} \qquad \Upsilon_0 = \{m \mid m(p_{treated}) = 1\}$$
$$\Upsilon_2 = \{m \mid m(p_{fatal}) = 1\}$$

**Figure 8.5.** *An exception mechanism*

languages recognized by SRPNs strictly includes the family of languages recognized by ordinary Petri nets.

*Modeling of fault tolerance.* In order to analyse fault-tolerant systems, the engineer may start by a nominal system and then introduces the faulty behavior as well as repairing mechanisms. We limit ourselves to an abstract view for such a system since this pattern can be simply generalized. This abstraction is given in the right part of Figure 8.7. The nominal system infinitely executes instructions (elementary transition $t_{count}$). The marking of place $p_{count}$ represents the number of instruction executions.

The complete SRPN is obtained by adding the left part of Figure 8.7. Its behavior can be described as follows. There are only two reachable extended markings reduced to a single node: the initial global state ($em_{start}$) where a token in $p_{start}$ indicates that the system is ready to start and the repairing state ($em_{repair}$) where a token in the place $p_{repair}$ indicates that the system is being repaired. Starting from the initial state, the abstract transition $t_{start}$ is fired and the execution of instructions is “played” by the new process. If this process terminates, meaning that a crash occurs, the repairing state

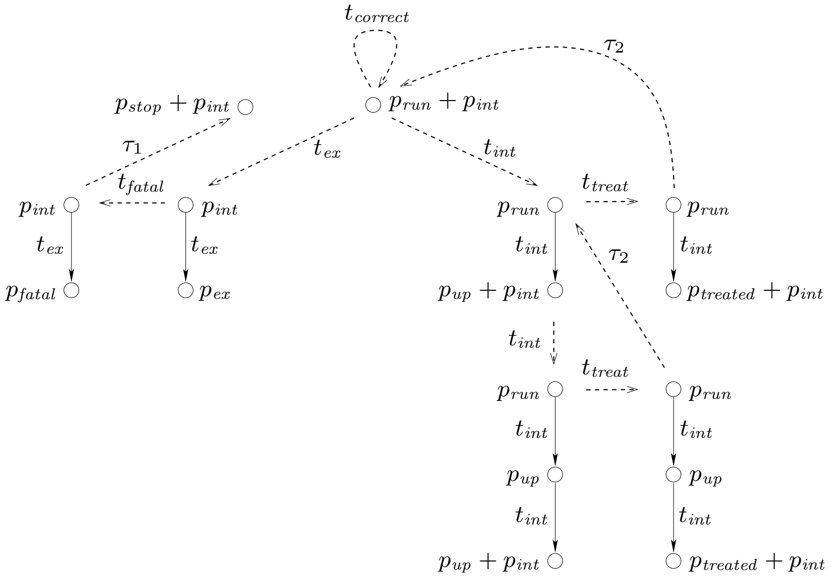
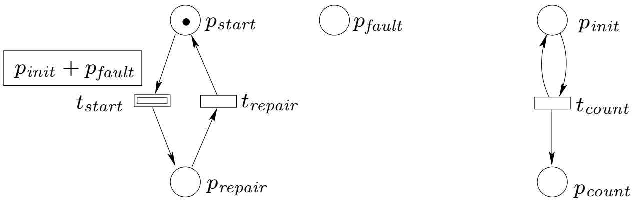


**Figure 8.6.** *Subgraph of the reachability graph of the SRPN of Figure 8.5*



$$\Upsilon_0 = \{m \mid m(p_{fault}) > 0\}$$

**Figure 8.7.** *A fault-tolerant system*

is reached. The place $p_{fault}$ represents the possibility of a crash. As $p_{fault}$ is always marked in the correct system and from the very definition of $\Upsilon_0$, the occurrence of a fault is always possible. We assume that no crash occurs during the repairing stage. With additional places and by modifying $\Upsilon_0$, we could model more complex fault occurrences (e.g. conditioned by software execution).

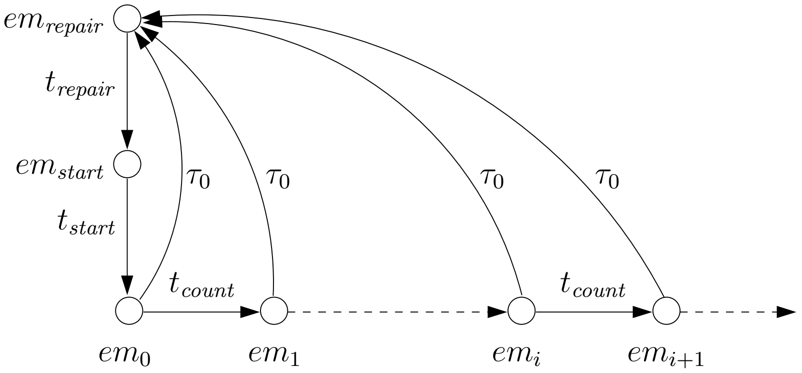


**Figure 8.8.** *(Infinite) reachability graph of the SRPN of the figure 8.7*

The reachable extended markings either consist of a single node or an initial node and its son. However, the number of reachable markings in this latter node is infinite (the place $p_{count}$ is unbounded). In other words, the repairing state can be reached from an infinite number of extended marking which means that *the transition system associated with an SRPN may have some states with an infinite in-degree*. This situation cannot occur with standard Petri nets or with process algebras. In particular, in a reachability graph obtained from a Petri net, for each marking/node, its in-degree is bounded by the number of transitions. Moreover allowing unobservable transitions does not solve the problem. Indeed, it has been proven that the transition system shown in Figure 8.8 cannot be generated by a standard Petri net with unobservable transitions.

Otherwise stated, the modeling of crash for a nominal system with an infinite number of states is impossible with Petri nets. In the restricted case where the nominal system has a finite number of reachable configurations, theoretically it is possible to model it with a standard Petri net. However, the modeling of a crash requires a number of transitions proportional to the number of reachable configurations, which leads to an intricate net. The SRPN designed herein does not depend on this number and leads to a quite compact representation.

### 8.3.3. *Verification*

A distinguished feature of Petri nets is the decidability of the reachability problem, even though the best known decision procedure does not lead to a primitive recursive complexity, see e.g. [REU 90]. Many safety properties (specifying that nothing bad will happen) can be reduced to instances of the reachability problem. Then, a decision procedure for the reachability problem can be viewed as a building block in larger verification tools.

It is possible to decide whether a Petri net is bounded. Because, the reachability graph of a bounded net is finite (by definition), the methods presented in Chapter 7 can be applied. Moreover, the membership problem with a language specified by a labeled Petri net is also a decidable problem. This allows verification that an expected behavior can be effectively realized by the system. Finally, the verification of a temporal property is possible but limited to the event-based linear-time temporal formula (an LTL formula can express the firing of transitions but not constraints on reached markings). The work in [ESP 94] gives a large synthesis of the decidability results concerning Petri nets.

These problems remain decidable for SRPN. Hence, even if it is a strict extension of Petri nets, the verification of numerous behavioral properties is still possible. In this section, we limit ourselves to the study of the reachability problem.

Another important feature of Petri net concerns the structural verification technique presented in Chapter 7. We present an algorithm for the computation of linear invariants of SRPN.

*Behavioral verification.* Given a marked SRPN, the reachability problem consists of checking whether an extended marking can be reached from the initial extended marking (through a sequence of successive firings). The decision procedure, informally presented below, is inspired from a more complex one that is dedicated to the larger class of RPNs. The idea consists of reducing an instance of the problem to several instances of the reachability problem for ordinary Petri nets.

The decision procedure can be divided into different stages:

– the first one is independent of the initial and final extended markings. It consists of determining whether a process created by a given abstract transition can reach a marking belonging to a termination set (those termination sets are specified by $\Upsilon$ in a SRPN). Such an abstract transition is said to be *closable* with respect to this termination set. This verification should be done for each abstract transition and for each termination set; thus it should be done at most $\mathrm{card}(T_{ab}) \times \mathrm{card}(I)$ times;

– the second stage aims to predict the behavior of processes composing the initial extended marking and, hence, the nature of the processes composing the extended marking to be reached. Indeed, a process of the initial extended marking can either be terminated during the firing sequence or can persist modifying its marking. The processes of the final extended marking, which do not correspond to processes of the initial extended marking, must be created by the firing sequence. Even if each prediction must satisfy some constraints (e.g. if a child process must be created by the firing sequence, its child processes must be created as well), many different predictions are generally possible. Figure 8.9 presents such a prediction. The two processes

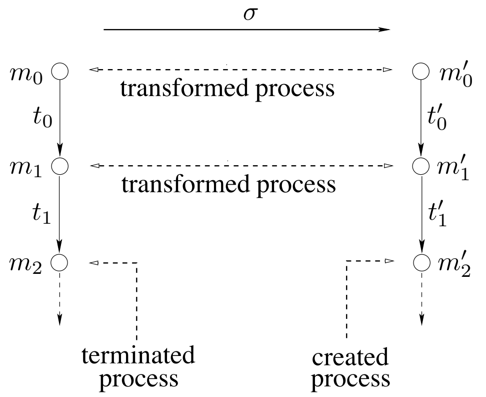


**Figure 8.9.** *A coherent prediction of the behavior of processes*

of the initial extended marking labeled by the markings $m_0$ and $m_1$ must evolve to the respective markings $m'_0$ and $m'_1$. On the contrary, the process labeled by $m_2$ must terminate during the sequence and, consequently, its child processes must also terminate. Finally, it is expected that the process labeled $m'_2$ in the reached extended marking will be created by the sequence. It is clear that the number of distinct and coherent predictions is bounded by the number of processes composing the initial and final extended markings.

All the steps of the decision procedure are based on the same principle: the construction of an ordinary Petri net and a reachability problem equivalent to the elementary problem to be decided. We now focus on the determination of *closable* abstract transitions.

Note that it is decidable to determine whether a marking from a semilinear set can be reached in a Petri net equipped with an initial marking (even if the semilinear set is infinite).

An abstract transition is said *closable* in a termination set if a process (created by this abstract transition) can realize a return step from this termination set. This amounts to checking an instance of the reachability problem and, when an abstract transition is closable, it induces a corresponding firing sequence. When an abstract transition is fired in a given process, the execution of this process is suspended until the child process created by the abstract transition terminates. Consequently, in the associated firing sequence, the firing corresponding to the root level can only involve some closable abstract transitions.

This leads to an iterative computation of the set $F \subseteq T_{ab} \times I$ of closable abstract transitions. $F_0$ is the subset of $F$ where the associated firing sequence contains no firing of abstract transitions. When new elements are added to $F$, this indicates that these transitions can be used.

The elements of $F_0$ are determined by testing if the ordinary net obtained by suppressing all the abstract transitions can reach a marking from a termination set from the initial marking of a given abstract transition (specified by `Init` in a SRPN). In order to compute the set $F_1$, the abstract transitions belonging to $F_0$ are now simulated by ordinary transitions having the same pre-conditions and post-conditions (for a given termination set). The aim of these transitions is to simulate the creation of child processes that are able to terminate (in this termination set).

Assuming that $F_i$ is defined, $F_{i+1}$ is defined as the set of closable abstract transitions, excluding those from $F_i$, such that we consider the SRPN in which the abstract transitions belonging to $F_i$ are now simulated by ordinary transitions having the same pre-conditions and post-conditions. Observe that there is $J \leq (\mathrm{card}(T_{ab}) \times \mathrm{card}(I)) + 1$ such that $F_J = \emptyset$ and whenever $F_i$ is empty, this implies that for all $j > i$, we have that $F_j$ is empty too. Knowing that the set of abstract transitions is finite as well as the number of final marking sets demonstrate the termination of the decision procedure. Hence, $F \overset{\text{def}}{=} \bigcup_i F_i$.

We are now in position to describe the procedure for the decision of the reachability problem. Let $src$ be the initial extended marking and $dest$ be the destination extended marking. If $src =\perp$ (the empty state) then it is sufficient to test if $dest =\perp$. Assume that $src \neq\perp$ and $dest =\perp$. This problem is similar to decide whether an abstract transition is closable except that the initial state is not necessarily composed by a unique process. Hence, the child processes must also be successively closed. This leads to a set of termination problems.

Finally, when $src \neq\perp$ and $dest \neq\perp$, the principle consists of testing the possible predictions one by one. The main difference comes from the fact that some processes of $dest$ can be predicted as created. Here again, the decision can be reduced to instances of the reachability problem in an ordinary Petri net. By way of example, let us consider the two extended markings below:

$$em_1 = (m_{d+d_1}, t_{d+d_1}), \ldots, (m_{d+1}, t_{d+1}), (m_d, t_d), \ldots, (m_2, t_2), m_1$$

$$em_2 = (m'_{d+d_2}, t'_{d+d_2}), \ldots, (m'_{d+1}, t'_{d+1}), (m_d, t_d), \ldots, (m_2, t_2), m_1$$

with $d_1, d_2 \geq 1$ and $d \geq 2$ (they share a common bottom of the stack). Assuming that these extended markings are defined with respect to the SRPN $N$, let $N^-$ be the standard Petri net obtained from $N$ by deleting the non-closable abstract transitions, and each closable abstract transition in $F$ is replaced by an ordinary transition having the same pre-condition and post-condition. We can show that $em_2$ is reachable from $em_1$ in $N$ iff there exists $i_1, \ldots, i_{d_1} \in I$ such that:

1) there exists a marking $m$ in $\mathrm{REL}(\varphi_{i_{d_1}})$ such that $m_{d+d_1} \xrightarrow{*} m$ in $N^-$;

2) for $j \in [1, d_1 - 1]$, there exists a marking $m$ in $\mathrm{REL}(\varphi_{i_j})$ such that $m_{d+j} + W^+(\cdot, t_{d+j+1}, i_{j+1}) \xrightarrow{*} m$ in $N^-$. As usual, $m_{d+j} + W^+(\cdot, t_{d+j+1}, i_{j+1})$ denotes the marking $m'$ such that for every place $p \in P$, $m'(p) = m_{d+j}(p) + W^+(p, t_{d+j+1}, i_{j+1})$. A similar notation is used below;

3) $\mathtt{Init}(t'_{d+d_2}) \xrightarrow{*} m'_{d+d_2}$ in $N^-$;

4) For $j \in [1, d_2 - 1]$, $\mathtt{Init}(t'_{d+j}) \xrightarrow{*} m'_{d+j} + W^-(\cdot, t'_{d+j+1})$.

Predictions (indeed non-determinism) are witnessed by the sequence of termination indices $i_1, \ldots, i_{d_1}$. Moreover, observe that (3) and (4) are instances of the reachability problem for Petri nets whereas (1) and (2) are instances of the reachability problem into a semilinear set, this latter problem being easily reducible to plain reachability.

*Structural verification*. Among the structural method presented in Chapter 7, we focus on the computation of linear invariants that we adapt to SRPN. The computation shown here is an adaptation of the one dedicated to RPN.

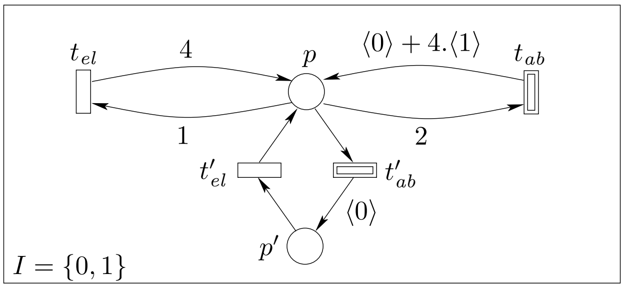
The incidence matrix $W$ of a SRPN is defined as follow. Rows are indexed by the places and by the abstract transitions. The intended meaning of a variable indexed by a place is its number of tokens while the interpretation of a variable indexed by an abstract transition is the current number of processes created by its firing.

The columns of the matrix are indexed by the transitions and by the pairs $(t, i)$ where $t$ is an abstract transition and $i \in I$ is an index corresponding to a termination set. A column indexed by a transition represents its firing while a pair $(t, i)$ indicates the firing of a cut step related to the termination set $\mathrm{REL}(\varphi_i)$ in a child process initiated by the abstract transition $t$.

The incidence matrix is defined by:

– for all $p \in P$, for all $t \in T_{el}$ and for all $t' \in T_{ab}$,
$W(p, t) = W^+(p, t) - W^-(p, t)$ and $W(t', t) = 0$;
– for all $p \in P$, for all $t, t' \in T_{ab}$ with $t' \neq t$,
$W(p, t) = -W^-(p, t)$ and $W(t, t) = 1$ and $W(t', t) = 0$;
– for all $p \in P$, for all $t, t' \in T_{ab}$, for all $i \in I$ with $t' \neq t$,
$W(p, (t, i)) = W^+(p, t, i)$ and $W(t, (t, i)) = -1$ and $W(t', (t, i)) = 0$.

Figure 8.10 illustrates the definition of the matrix $W$. The matrix is divided into six blocks depending on the type of rows and columns. Let us look at some items of the row indexed by place $p$: the elementary transition $t_{el}$ consumes one token from $p$ and produces 4 tokens for it, thus the corresponding value in the matrix is 3; firing the abstract transition $t_{ab}$ consumes 2 tokens thus the corresponding item is $-2$ and the cut step associated with $t_{ab}$ and index 0 (resp. 1) produces 1 token (resp. 4 tokens)



(a) A SRPN (only relevant elements are represented)

$$\begin{array}{c|cc|cc|cccc|l}
 & t_{el} & t'_{el} & t_{ab} & t'_{ab} & (t_{ab},0) & (t_{ab},1) & (t'_{ab},0) & (t'_{ab},1) & \\
\hline
p & 3 & 1 & -2 & -1 & 1 & 4 & 0 & 0 & \Big\} P \\
p' & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & \\
\hline
t_{ab} & 0 & 0 & 1 & 0 & -1 & -1 & 0 & 0 & \Big\} T_{ab} \\
t'_{ab} & 0 & 0 & 0 & 1 & 0 & 0 & -1 & -1 & \\
\hline
\multicolumn{1}{c}{} & \multicolumn{2}{c}{\underbrace{\qquad\quad}_{T_{el}}} & \multicolumn{2}{c}{\underbrace{\qquad\quad}_{T_{ab}}} & \multicolumn{4}{c}{\underbrace{\qquad\qquad\qquad\qquad}_{T_{ab} \times I}} &
\end{array}$$

(b) Its incidence matrix $W$

**Figure 8.10.** *A SRPN and its incidence matrix*

thus the corresponding item is 1 (resp. 4). Let us have a look at the row indexed by the abstract transition $t_{ab}$: firing $t_{ab}$ creates one more process initiated by $t_{ab}$, thus the corresponding item is 1 while firing any of the two cut steps corresponding to $t_{ab}$ terminates one such process yielding an item $-1$.

Given a process $pr$ in an extended marking $em$, we denote by $fire(pr)^{em}$ the vector, indexed on $T_{ab}$, such that for any abstract transition $t$, $fire(pr)^{em}(t)$ is equal to one if a child process initiated by $t$ has been created by $pr$ and is not already terminated and equal to zero otherwise. We are now in position to justify the choice of $W$ as incidence matrix.

Let $em'$ be an extended marking of an SRPN; which is reachable from a given state $em$ via a firing sequence $\sigma$. Assume the existence of a process $pr$ in both $em$ and $em'$ (and then in all the extended markings visited by $\sigma$). We note $m(pr)^{em}$ the marking of the process $pr$ in the state $em$. Let $x$ be a solution of $x \cdot W = 0$, then:

$$x \cdot (m(pr)^{em'}, fire(pr)^{em'}) = x \cdot (m(pr)^{em}, fire(pr)^{em})$$

As for ordinary Petri nets, when $em$ is the initial state of the SRPN, this equation is called a linear invariant. In order to obtain linear invariants, we can compute a generative family of solutions $\{x_1, \ldots, x_n\}$ of this equation. For a process $pr$ of the initial extended marking $em_0$, we obtain a superset of the reachable "states" of this process by the set of equations: for $i \in [1, n]$, $x_i \cdot (m(pr)^{em}, fire(pr)^{em}) = x_i \cdot (m(pr)^{em_0}, fire(pr)^{em_0})$.

The same overestimation can be done for the reachable ordinary marking space of a process $pr$ dynamically created by the firing of an abstract transition $t$: for all $i \in [1, n], x_i \cdot (m(pr)^{em}, fire(pr)^{em}) = x_i \cdot (\mathtt{Init}(t), \vec{0})$ (where $\mathtt{Init}(t)$ corresponds to the starting marking associated with $t$).

The fact that depending on the initial marking $\mathtt{Init}(t)$ some transitions are dead may lead to additional sources of overestimation. Since this last factor often happens in practical cases, we describe now an iterative method that tackles this problem. In fact, the method is also applicable to Petri nets but it is of limited interest in this case since usually the transitions of a Petri net are not dead.

Algorithm 4 simultaneously computes a set of linear invariants fulfilled by markings reachable from $m$, an ordinary marking, and a superset of the transitions enabled at least once from $m$. More precisely, it initialises $T_{live}$ as the empty set. Then it computes the positive invariants for the recursive Petri net whose transitions are reduced to $T_{live}$. In the algorithm 4, the function *Invariant* returns a generative family of invariants (see [COL 90] for efficient computation of such families). For each transition that does not belong to $T_{live}$, it builds a linear problem with the invariants and the firing conditions of this transition. If this problem admits a solution, it is possibly enabled and so, it adds it to $T_{live}$. This process is iterated until $T_{live}$ is saturated.

Finally, we describe how the linear invariants can be used to obtain information about the structure of the reachable extended markings. We build a graph whose nodes are the abstract transitions. There is an edge from $t$ to $t'$ if starting from the marking $\mathtt{Init}(t)$ a process may fire $t'$. In order to determine such an edge, we compute the invariants associated with $\mathtt{Init}(t)$ by a call to *structReach*. If $t'$ belongs to the set returned by this call then an edge is added. This graph is a skeleton for the dynamic structure of the extended markings. For instance, if it is acyclic, then any reachable extended marking has a bounded depth.

### 8.3.4. *Related work*

We have seen that Petri nets can be extended while preserving the decidability status of numerous problems (reachability problems, etc.). However, it is important to note that minor extensions can lead to undecidability results. For instance, the reachability problem is undecidable for Petri nets with two inhibitor arcs (a computational

```
input  : an ordinary marking $m$
output: a set of invariants and a set of transitions
$T_{live} = \emptyset$;
$New = \emptyset$;
$In = \emptyset$;
repeat
    $New = \emptyset$;
    $In = Invariant(N, m, T_{live})$;
    foreach $t \in T \setminus T_{live}$ do
        Build a linear problem $Pb$ in $\mathbb{N}^P$ with $In$ and $W^-(-, t)$;
        if $Pb$ has a solution then
            $New = New \cup \{t\}$;
        end
    end
    $T_{live} = T_{live} \cup New$;
until ($New == \emptyset$) ;
return $\langle In, T_{live} \rangle$;
```

**Algorithm 4**: *structReach*

model similar to Minsky machines) while it becomes decidable with one inhibitor arc (or a particular nested structure of inhibitor arcs). The self-modifying nets introduced by R. Valk have (like Petri nets with inhibitor arcs) the power of Turing machines and thus many properties including reachability are undecidable [VAL 78a, VAL 78b]. Moreover, these extensions do not offer a practical way to model the dynamic creation of objects.

In order to tackle this problem, A. Kiehn introduced a model called net systems [KIE 89] that are Petri nets with special transitions whose firing starts a new token game of one of these nets. A *call* to a Petri net, triggered by such a firing, may return if this net reaches a final marking. All the nets are required to be safe and the constraints associated with the final marking ensure that a net may not return if it has pending calls. It is straightforward to simulate a net system by an RPN. Moreover, as the class of languages recognized by Petri nets is not included in the class of languages recognized by net systems, the class of languages recognized by net systems is strictly included in the family of RPN languages.

Process algebra nets (PANs), introduced by R. Mayr [MAY 97], are a model of process algebra including the sequential composition operator as well as the parallel operator. The left term of any rule of a PAN may use only the parallel composition of variables whereas the right side is a general term. This model includes Petri nets and context-free grammars. In [HAD 00], the authors demonstrate that RPNs also include PANs. However, it is not known whether the inclusion of the PAN languages by the

RPN languages is strict. Moreover, PANs as well as process rewrite systems (a more expressive model) cannot represent a transition system with an infinite in-degree.

A verification technique, which is not treated in this section, concerns the equivalence relations between two nets. This approach is essential when the design is realized by successive refinements. In [HAD 07], it has been shown that checking bisimulation between an SRPN (satisfying some additional constraints) and a finite automaton is a decidable problem.

### 8.4. Presburger arithmetic as symbolic representation

In section 8.3, we have seen how verification problems for SRPNs can be solved by using techniques for standard Petri nets, at the cost of adequately adapting standard methods. Typically, an instance of the reachability problem for SRPNs (involving extended markings) is transformed into a finite number of instances of the reachability problem for Petri nets. By contrast, in this section, we consider other extensions of VASS by allowing for instance zero-tests, but in a controlled manner. In this section, decidability of the reachability problem is obtained by reduction into instances of the satisfiability problem for Presburger arithmetic. More precisely, in this section, we consider subclasses of counter systems for which the reachability sets of the form $\{\vec{x} \in \mathbb{N}^n : (q_0, \vec{x_0}) \xrightarrow{*} (q, \vec{x})\}$ are effectively Presburger-definable ($(q_0, \vec{x_0})$ and $q$ are fixed). By decidability of Presburger arithmetic, this allows us to solve problems restricted to such counter systems such as the reachability problem, the control state reachability problem, the boundedness problem, or the covering problem. Indeed, suppose that given $(q_0, \vec{x_0})$ and $q$, we can effectively build a Presburger formula $\varphi_q$ such that $\mathrm{REL}(\varphi_q) = \{\vec{x} \in \mathbb{N}^n : (q_0, \vec{x_0}) \xrightarrow{*} (q, \vec{x})\}$. We can then easily show the properties below:

1) $\{\vec{x} \in \mathbb{N}^n : (q_0, \vec{x_0}) \xrightarrow{*} (q, \vec{x})\}$ is infinite iff the formula below is satisfiable:

$$\neg \exists \mathsf{y} \, \forall \mathsf{x}_1, \ldots, \mathsf{x}_n \; \varphi_q(\mathsf{x}_1, \ldots, \mathsf{x}_n) \Rightarrow (\mathsf{x}_1 \leq \mathsf{y} \wedge \cdots \wedge \mathsf{x}_n \leq \mathsf{y});$$

2) $(q_0, \vec{x_0}) \xrightarrow{*} (q, \vec{z})$ iff the formula below is satisfiable:

$$\varphi_q(\mathsf{x}_1, \ldots, \mathsf{x}_n) \wedge \mathsf{x}_1 = \vec{z}(1) \wedge \cdots \wedge \mathsf{x}_n = \vec{z}(n),$$

where any constant $k > 0$ is encoded by the term $\overbrace{1 + \cdots + 1}^{k \text{ times}}$;

3) control state $q$ can be reached from $(q_0, \vec{x_0})$ iff the Presburger formula $\varphi_q(\mathsf{x}_1, \ldots, \mathsf{x}_n)$ is satisfiable.

Below, we consider the class of reversal-bounded counter automata and the class of admissible counter systems. However, other types of counter systems with Presburger-definable reachability sets exist, see numerous examples in [PAR 66, ARA 77, HOP 79, ESP 97, COM 98, FIN 00, LER 03] (see also the generalizations presented in [LER 05, BOZ 10]). Besides, let us briefly recall below why reachability sets for VASS (succinct counter automata without zero-tests) may not be semilinear, witnessing the fact that semilinearity is not always guaranteed even for harmless VASS. In Figure 8.11, we present a slight variant of the VASS described in Figure 8.2 by adding two components to store counter values just before entering in the control state $q_0$ for the first time. We can show that

$$\{(\vec{x}(1), \vec{x}(2), \vec{x}(6)) : (q_0, \vec{0}) \xrightarrow{*} (q_1, \vec{x})\} = \{(n_1, n_2, n_3) \in \mathbb{N}^3 : n_3 \leq n_1 \times n_2\}$$

Now, suppose that there is a Presburger formula $\varphi(\mathsf{x}_1, \ldots, \mathsf{x}_6)$ such that $\mathrm{REL}(\varphi) = \{\vec{x} : (q_0, \vec{0}) \xrightarrow{*} (q_1, \vec{x})\}$. We can build from it a Preburger formula $\chi(\mathsf{x})$ such that $\mathrm{REL}(\chi(\mathsf{x})) = \{n^2 : n \geq 0\}(= Y)$:

$$\exists\, \mathsf{x}_1, \ldots, \mathsf{x}_5\ \varphi(\mathsf{x}_1, \ldots, \mathsf{x}_5, \mathsf{x}) \wedge \mathsf{x}_1 = \mathsf{x}_2 \wedge (\forall\, \mathsf{x}' (\mathsf{x}' > \mathsf{x}) \Rightarrow \neg\exists\, \mathsf{x}_3, \mathsf{x}_4, \mathsf{x}_5\ \varphi(\mathsf{x}_1, \ldots, \mathsf{x}_5, \mathsf{x}'))$$

Since $Y$ is infinite, there are $b \geq 0$ and $p_1, \ldots, p_m > 0$ $(m \geq 1)$ such that $(Z = )\{b + \sum_{i=1}^{m} n_i p_i : n_1, \ldots, n_m \in \mathbb{N}\} \subseteq Y$. Let $N \in \mathbb{N}$ be such that $N^2 \in Z$ and $(2N + 1) > p_1$. The value $N$ always exists since $Z$ is infinite. Since $Z$ is a linear set, we also have $(N^2 + p_1) \in Z$. However $(N + 1)^2 - N^2 = (2N + 1) > p_1$. Hence $N^2 < N^2 + p_1 < (N + 1)^2$, which leads to a contradiction.

By contrast, the reachability sets for VASS of dimension 2 can be shown to be effectively Presburger-definable [HOP 79].

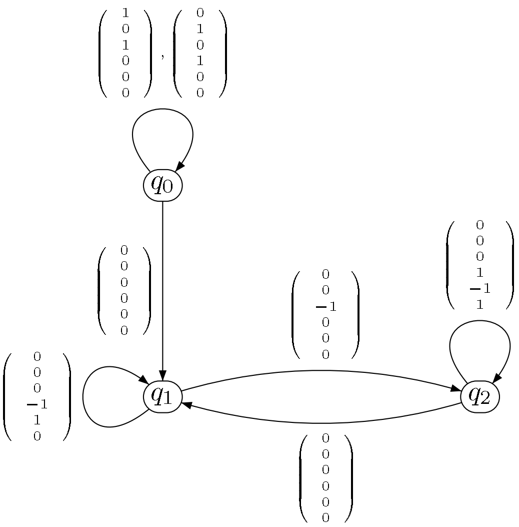

$\begin{pmatrix}1\\0\\1\\0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\\1\\0\\0\end{pmatrix}$
$q_0$
$\begin{pmatrix}0\\0\\0\\0\\0\\0\end{pmatrix}$
$\begin{pmatrix}0\\0\\-1\\0\\0\\0\end{pmatrix}$
$\begin{pmatrix}0\\0\\0\\1\\-1\\1\end{pmatrix}$
$\begin{pmatrix}0\\0\\0\\-1\\1\\0\end{pmatrix}$
$q_1$
$q_2$
$\begin{pmatrix}0\\0\\0\\0\\0\\0\end{pmatrix}$

**Figure 8.11.** *A VASS weakly computing multiplication (bis)*

### 8.4.1. *Presburger-definable reachability sets*

*Reversal-bounded counter automata*

A *reversal* for a counter occurs in a run when there is an alternation from non-increasing mode to non-decreasing mode and vice versa. Figure 8.12 presents schematically the behavior of a counter with five reversals.

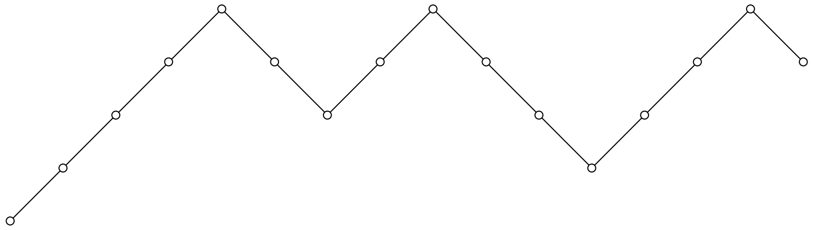


**Figure 8.12.** *Five reversals in a row*

A counter automaton is *reversal-bounded* whenever there is $r \geq 0$ such that for any run from a given initial configuration, every counter makes no more than $r$ reversals. This class of counter automata has been introduced and studied in [IBA 78]. A formal definition will follow, but before going any further, it is worth pointing out a few peculiarities of this subclass. Indeed, reversal-boundedness is defined for initialized counter automata (a counter automaton augmented with an initial configuration) and the bound $r$ depends on the initial configuration. Secondly, this class is not defined from the class of counter automata by imposing syntactic restrictions but rather semantically. Its very definition is motivated by technical and theoretical considerations rather than by constraints from case studies.

Let $\mathcal{S} = (Q, n, \delta)$ be a standard counter automaton. Let us define the auxiliary (succinct) counter automaton $\mathcal{S}_{rb} = (Q', 2n, \delta')$ such that $Q' = Q \times \{\mathrm{DEC}, \mathrm{INC}\}^n$ and $(q, \vec{mode}) \xrightarrow{\varphi'} (q', \vec{mode}') \in \delta' \overset{\text{def}}{\Leftrightarrow}$ there is $q \xrightarrow{\varphi} q' \in \delta$ such that if $\varphi$ does not deal with the $j$th component, then $\vec{mode}(j) = \vec{mode}'(j)$ and for every $i \in [1, n]$, one of the conditions below is satisfied:

- $\varphi = \mathrm{zero}(i), \vec{mode}(i) = \vec{mode}'(i), \varphi' = \varphi \wedge \bigwedge_{j\in[1,n]} \mathsf{x}'_{n+j} = \mathsf{x}_{n+j}$;
- $\varphi = \mathrm{dec}(i), \vec{mode}(i) = \vec{mode}'(i) = \mathrm{DEC}$ and $\varphi' = \varphi \wedge \bigwedge_{j\in[1,n]} \mathsf{x}'_{n+j} = \mathsf{x}_{n+j}$;
- $\varphi = \mathrm{dec}(i), \vec{mode}(i) = \mathrm{INC}, \vec{mode}'(i) = \mathrm{DEC}$ and

$$\varphi' = \varphi \wedge (\mathsf{x}'_{n+i} = \mathsf{x}_{n+i} + 1) \wedge \bigwedge_{j\in[1,n]\setminus\{i\}} \mathsf{x}'_{n+j} = \mathsf{x}_{n+j};$$

- $\varphi = \mathrm{inc}(i), \vec{mode}(i) = \vec{mode}'(i) = \mathrm{INC}$ and $\varphi' = \varphi \wedge \bigwedge_{j\in[1,n]} \mathsf{x}'_{n+j} = \mathsf{x}_{n+j}$;

- $\varphi = \mathrm{inc}(i), \vec{mode}(i) = \mathrm{DEC}, \vec{mode}'(i) = \mathrm{INC}$ and

$$\varphi' = \varphi \wedge (\mathsf{x}'_{n+i} = \mathsf{x}_{n+i} + 1) \wedge \bigwedge_{j \in [1,n] \setminus \{i\}} \mathsf{x}'_{n+j} = \mathsf{x}_{n+j}.$$

Essentially, the $n$ new components in $\mathcal{S}_{rb}$ count the number of reversals for each component from $\mathcal{S}$. Moreover, the above construction could be easily adapted so that to control $\mathcal{S}$ by imposing that each counter does not perform more than $r$ reversals, for some fixed bound $r$. Observe that $\mathcal{S}_{rb}$ is succinct because two counters may be updated in one step. Initialized counter automaton $(\mathcal{S}, (q, \vec{x}))$ is *reversal-bounded* [IBA 78] $\overset{\text{def}}{\Leftrightarrow}$ for every $i \in [n+1, 2n], \{\vec{y}(i) : \exists \, \mathrm{run} \, (q_{rb}, \vec{x}_{rb}) \xrightarrow{*} (q', \vec{y}) \text{ in } \mathcal{S}_{rb}\}$ is finite with $q_{rb} = (q, \vec{\mathrm{INC}})$, $\vec{x}_{rb}$ restricted to the $n$ first components is $\vec{x}$ and $\vec{x}_{rb}$ restricted to the $n$ last components is $\vec{0}$. When $r \geq \max(\{\vec{y}(i) : \mathrm{run} \, (q_{rb}, \vec{x}_{rb}) \xrightarrow{*} (q', \vec{y}) \text{ in } \mathcal{S}_{rb}\} : i \in [n+1, 2n])$ $\mathcal{S}$ is said to be *$r$-reversal-bounded* from $(q, \vec{x})$.

A counter automaton $\mathcal{S}$ is *uniformly reversal-bounded* iff there is $r \geq 0$ such that for every initial configuration, the initialized counter automaton is $r$-reversal-bounded. We can check that the counter automaton in Figure 8.13 is not uniformly reversal-bounded.

Figure 8.13 contains a counter automaton $\mathcal{S}$ such that any initialized counter automaton of the form $(\mathcal{S}, (q_1, \vec{x}))$ with $\vec{x} \in \mathbb{N}^2$ is reversal-bounded. Let $\vec{x} \in \mathbb{N}^2$ and $\varphi$ be the Presburger formula

$$\varphi = (\mathsf{x}_1 \geq 2 \wedge \mathsf{x}_2 \geq 1 + \vec{x}(2) \wedge (\mathsf{x}_2 - \vec{x}(2)) + 1 \geq \mathsf{x}_1) \vee$$
$$(\mathsf{x}_2 \geq 2 \wedge \mathsf{x}_1 \geq 1 + \vec{x}(1) \wedge (\mathsf{x}_1 - \vec{x}(1)) + 1 \geq \mathsf{x}_2)$$

We can show that $\mathrm{REL}(\varphi)$ is precisely equal to the reachability set $\{\vec{y} \in \mathbb{N}^2 : (q_1, \vec{x}) \xrightarrow{*} (q_9, \vec{y})\}$.

Reversal-boundedness for counter automata is very appealing because reachability sets are Presburger-definable as stated below.

**THEOREM 8.9** *[IBA 78] Let $r \geq 0$ and $(\mathcal{S}, (q, \vec{x}))$ be an initialized counter automaton that is $r$-reversal-bounded. For each control state $q'$, the set $\{\vec{y} \in \mathbb{N}^n : \exists \, \mathrm{run} \, (q, \vec{x}) \xrightarrow{*} (q', \vec{y})\}$ is effectively Presburger-definable.*

This means that we can compute a Presburger formula that characterizes the reachable configurations whose control state is $q'$. The original proof for reversal-boundedness can be found in [IBA 78].

As a consequence of theorem 8.9, we get:

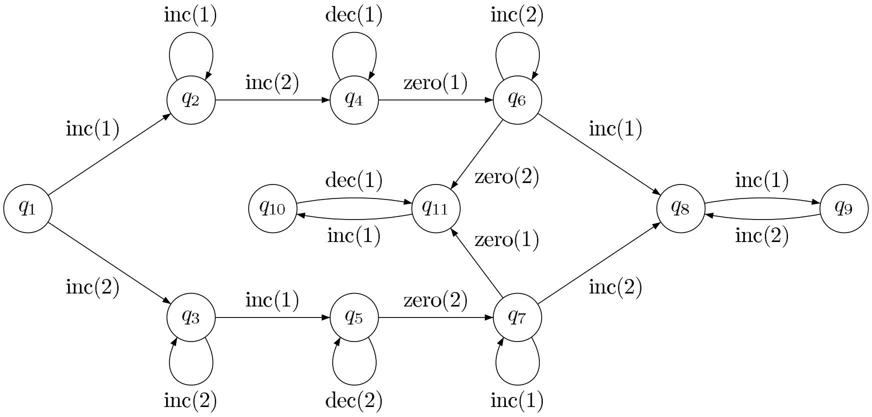


**Figure 8.13.** *A counter automaton that bounds the numbers of reversals*

COROLLARY **8.10** *The reachability problem for reversal-bounded counter automata is decidable.*

Moreover, the control state repeated reachability problem for reversal-bounded counter automata is decidable too by reduction into the reachability problem, see e.g. [DAN 01].

Let us consider another problem that can be shown decidable even though it takes as input a standard counter automaton without any further restriction.

Reachability problem with bounded number of reversals:

– *Input*: a counter automaton $\mathcal{S}$, a bound $r \in \mathbb{N}$, an initial configuration $(q_0, \vec{x_0})$ and a final configuration $(q, \vec{x})$;

– *Question*: is there a finite run of $\mathcal{S}$ with initial configuration $(q_0, \vec{x_0})$ and final configuration $(q, \vec{x})$ such that each counter has at most $r$ reversals?

Observe that when $(\mathcal{S}, (q_0, \vec{x_0}))$ is $r'$-reversal-bounded for some $r' \leq r$, we get an instance of the reachability problem with initial configuration $(q_0, \vec{x_0})$.

COROLLARY **8.11** *The reachability problem with bounded number of reversals is decidable.*

By using [GUR 81], the problem can be solved in non-deterministic exponential time.

*Affine counter systems with finite monoids*

We shall define the class of *affine counter systems* that slightly generalizes the class of succinct counter automata (roughly speaking, a counter value can be multiplied by a factor different from 1). To do so, we start by proposing a few definitions.

A *binary relation of dimension* $n$ is a relation $R \subseteq \mathbb{N}^{2n}$. $R$ is *Presburger-definable* $\overset{\text{def}}{\Leftrightarrow}$ there is a Presburger formula $\varphi(\mathsf{x}_1, \ldots, \mathsf{x}_n, \mathsf{x}'_1, \ldots, \mathsf{x}'_n)$ with $2n$ free variables such that $R = \mathrm{REL}(\varphi)$. A partial function $f$ from $\mathbb{N}^n$ to $\mathbb{N}^n$ is *affine* $\overset{\text{def}}{\Leftrightarrow}$ there exist a matrix $A \in \mathbb{Z}^{n \times n}$ and $\vec{b} \in \mathbb{Z}^n$ such that for every $\vec{a} \in \mathrm{dom}(f)$, we have $f(\vec{a}) = A\vec{a} + \vec{b}$. $f$ is *Presburger-definable* $\overset{\text{def}}{\Leftrightarrow}$ the graph of $f$ is a Presburger-definable relation.

A counter system $\mathcal{S} = (Q, n, \delta)$ is *affine* when for every transition $q \xrightarrow{\varphi} q' \in \delta$, $\mathrm{REL}(\varphi)$ is affine. In the sequel, each formula $\varphi$ labeling a transition in an affine counter system is encoded by a triple $(A, \vec{b}, \psi)$ such that

1) $A \in \mathbb{Z}^{n \times n}, \vec{b} \in \mathbb{Z}^n$;
2) $\psi$ has free variables $\mathsf{x}_1, \ldots, \mathsf{x}_n$;
3) $\mathrm{REL}(\varphi) = \{(\vec{x}, \vec{x'}) \in \mathbb{N}^{2n} : \vec{x'} = A\vec{x} + \vec{b} \text{ and } \vec{x} \in \mathrm{REL}(\psi)\}$.

The formula $\psi$ can be viewed as the guard of the transition and the pair $(A, \vec{b})$ as the (deterministic) update function. Such a triple $(A, \vec{b}, \psi)$ is called an *affine update* and we also write $\mathrm{REL}((A, \vec{b}, \psi))$ to denote $\mathrm{REL}(\varphi)$. Furthermore, succinct counter automata are affine counter systems in which the matrices are equal to the identity matrix. Moreover, in succinct counter automata the guards are reduced to the truth constant or to a zero-test. This class of counter systems has been introduced in [FIN 02].

Lemma 8.12 roughly states that the composition of affine updates is still an affine update, which shall be helpful to show that the accessibility relation for admissible counter systems is Presburger-definable.

**LEMMA 8.12** *Let* $(A_1, \vec{b_1}, \psi_1)$ *and* $(A_2, \vec{b_2}, \psi_2)$ *be two affine updates. There exists an affine update* $(A, \vec{b}, \psi)$ *such that*

$$\mathrm{REL}((A, \vec{b}, \psi)) =$$

$$\{(\vec{x}, \vec{x'}) \in \mathbb{N}^{2n} : \exists \vec{y} \in \mathbb{N}^n \; (\vec{x}, \vec{y}) \in \mathrm{REL}((A_1, \vec{b_1}, \psi_1)) \text{ and } (\vec{y}, \vec{x'}) \in \mathrm{REL}((A_2, \vec{b_2}, \psi_2))\}$$

In the forthcoming class of admissible counter systems, we shall assume that the control graph is flat. A counter system is *flat* whenever every control state belongs to at most one simple cycle, i.e. with no repeated vertex, in the control graph. Moreover,

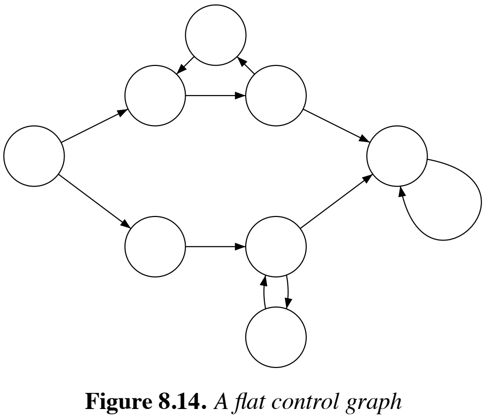


**Figure 8.14.** *A flat control graph*

we require that there is at most one transition between two control states. An example of flat control graph can be found in Figure 8.14.

Hence, it becomes essential to symbolically represent the effect of loops on counter values. This sounds a necessary condition to establish that a reachability relation is Presburger-definable. We already know by Lemma 8.12 that transitions in affine counter systems are closed under bounded compositions.

Let $R$ be a binary relation of dimension $n$. The *reflexive and transitive closure* of $R$, written $R^*$, is a subset of $\mathbb{N}^{2n}$ such that $(\vec{y}, \vec{y'}) \in R^*$ iff there are $\vec{x_1}, \ldots \vec{x_k} \in \mathbb{N}^n$ such that $\vec{x_1} = \vec{y}$, $\vec{x_k} = \vec{y'}$ and for $i \in [1, k-1]$, we have $(\vec{x_i}, \vec{x_{i+1}}) \in R$. If $R$ is Presburger-definable, then this does not imply that $R^*$ is Presburger-definable too. For instance, if $R = \{(k, 2k) \in \mathbb{N}^2 : k \in \mathbb{N}\}$ then $R^* = \{(k, 2^{k'}k) \in \mathbb{N}^2 : k, k' \in \mathbb{N}\}$ is not Presburger-definable. By contrast, if $S = \{(k, k+1) \in \mathbb{N}^2 : k \in \mathbb{N}\}$ then $S^* = \{(k, k') \in \mathbb{N}^2 : k < k',\ k, k' \in \mathbb{N}\}$ is Presburger definable. Counter systems of dimension $n$ induce naturally one-step binary relations of dimension $n$ that are Presburger-definable; the question of deciding whether their reflexive and transitive closure of is Presburger-definable would directly answer whether the reachability relations in such systems are Presburger-definable or not.

Indeed, consider the following loop with $q_1 = q_k$:

$$q_1 \xrightarrow{\varphi_1(\mathsf{x}_1,\ldots,\mathsf{x}'_n)} q_2 \xrightarrow{\varphi_2(\mathsf{x}_1,\ldots,\mathsf{x}'_n)} \cdots \xrightarrow{\varphi_{k-1}(\mathsf{x}_1,\ldots,\mathsf{x}'_n)} q_{k-1} \xrightarrow{\varphi_k(\mathsf{x}_1,\ldots,\mathsf{x}'_n)} q_k.$$

The effect of the loop can be represented by the Presburger formula below:

$$\psi(\vec{\mathsf{x}}, \vec{\mathsf{x}'}) \stackrel{\text{def}}{=} \exists\, \vec{\mathsf{y}_1}, \ldots, \vec{\mathsf{y}_k}\ \varphi_1(\vec{\mathsf{x}}, \vec{\mathsf{y}_1}) \wedge \varphi_2(\vec{\mathsf{y}_1}, \vec{\mathsf{y}_2}) \wedge \cdots \wedge \varphi_k(\vec{\mathsf{y}_k}, \vec{\mathsf{x}'})$$

The effect of visiting the loop a finite (but unbounded) number of times amounts to represent symbolically the reflexive and transitive closure of $\mathrm{REL}(\psi(\vec{\mathsf{x}}, \vec{\mathsf{x}'}))$. The best we can hope for is that $\mathrm{REL}(\psi(\vec{\mathsf{x}}, \vec{\mathsf{x}'}))^*$ is Presburger-definable.

Given $A \in \mathbb{Z}^{n \times n}$, we write $A^*$ to denote the monoid generated from $A$ with $A^* = \{A^i : i \in \mathbb{N}\}$. The identity element is naturally the identity matrix $A^0 = I$. Given a matrix $A \in \mathbb{Z}^{n \times n}$, checking whether the monoid generated by $A$ is finite, is decidable [MAN 77].

A loop in an affine counter system has the *finite monoid property* $\overset{\text{def}}{\Leftrightarrow}$ its corresponding affine update $(A, \vec{b}, \psi)$, possibly obtained by composition of several affine updates, satisfies that $A^*$ is finite. Let us introduce below the class of admissible counter systems.

**DEFINITION 8.13** *A counter system $\mathcal{S}$ is* admissible *if $\mathcal{S}$ is an affine counter system, its control graph is flat, and each loop has the finite monoid property.*

**THEOREM 8.14** *[BOI 98, FIN 02] Let $\mathcal{S}$ be an admissible counter system and $q, q' \in Q$. We can compute a Presburger formula $\varphi$ such that for every valuation $\mathbf{v}$, we have $\mathbf{v} \models \varphi$ iff $(q, (\mathbf{v}(\mathsf{x}_1), \ldots, \mathbf{v}(\mathsf{x}_n))) \xrightarrow{*} (q', (\mathbf{v}(\mathsf{x}'_1), \ldots, \mathbf{v}(\mathsf{x}'_n)))$.*

As a corollary, the reachability problem for admissible counter systems is decidable. This result can be pushed a bit further by showing that model-checking over an extension of the temporal logic CTL$^\star$ with arithmetical constraints for admissible counter systems is decidable too [DEM 06]. Indeed, theorem 8.14 states that the reachability relation is indeed Presburger-definable.

If we give up the assumption on the finite monoid property, the reachability problem is undecidable for flat affine counter systems [COR 02]. However, theorem 8.14 still holds true if we relax the notion of admissibility a bit for instance by allowing that between two control states for which no transition belongs to a cycle, more than one transitions are allowed. Giving up the flatness condition in admissible counter systems also leads to undecidable reachability problems since this new class would capture the class of counter automata.

As observed in [COM 98, FIN 02, LER 03], flatness is very often essential to get effective Presburger-definable reachability sets (but of course this is not a necessary condition, see e.g. [PAR 66, HOP 79, ESP 97]). However, flat counter systems are seldom natural in real-life applications. Therefore, a relaxed version of flatness has been considered in [LER 05, DEM 06] so that an initialized counter system $(\mathcal{S}, (q, \vec{x}))$ is *flattable* whenever there is a partial unfolding of $(\mathcal{S}, (q, \vec{x}))$ that is flat and has the same reachability set as $(\mathcal{S}, (q, \vec{x}))$. In that way, reachability questions on $(\mathcal{S}, (q, \vec{x}))$ can still be decided even in the absence of flatness but in general properties on finite traces are not preserved. For the sake of completeness, let us provide below basic definitions about flattable counter systems.

Let L be a finite union of bounded languages of the form

$$u_1(v_1)^* u_2(v_2)^* \cdots (v_k)^* u_{k+1},$$

where $u_i \in \Sigma^*$, $v_i \in \Sigma^+$, $\Sigma = \delta$ is the set of transitions from $\mathcal{S}$ such that in the expression $u_1(v_1)^* u_2(v_2)^* \cdots (v_k)^* u_{k+1}$, two consecutive transitions share an intermediate control state. So, $(\mathcal{S}, (q, \vec{x}))$ is *initially flattable* [LER 05] if there is some language L of the above form such that the configurations reachable from $(q, \vec{x})$ are those reachable by firing the sequences of transitions from L (not every such sequence leads to a run, partly because counter values should be non-negative). So, there is some language L of the above form such that

$$\{(q', \vec{x'}) : (q, \vec{x}) \xrightarrow{*} (q', \vec{x'})\} = \{(q', \vec{x'}) : (q, \vec{x}) \xrightarrow{u} (q', \vec{x'}), u \in \mathrm{L}\}$$

For instance, the initialized counter system $(\mathcal{S}, (q_1, \vec{0}))$ in Figure 8.15 is initially flattable (see e.g. further explanations about the phone controller in [COM 00]). Indeed, whenever the control state $q_1$ is visited, the counters are reset. So, by deleting the transition from $q_1$ to $q_6$, we obtain a flat counter system without modifying the reachability set from $(q_1, \vec{0})$.

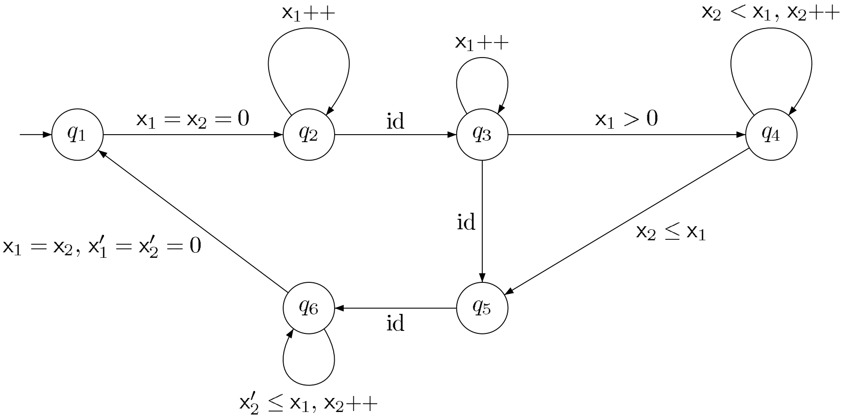


**Figure 8.15.** *Phone controller*

Similarly, $\mathcal{S}$ is *uniformly flattable* [LER 05] iff there is some language L of the above form such that the reachability relation $\xrightarrow{*}$ is equal to $\{((q, \vec{x}), (q', \vec{x'})) : (q, \vec{x}) \xrightarrow{u} (q', \vec{x'}), u \in \mathrm{L}\}$, which means that reachability can be restricted to sequences of transitions from a bounded language. Surprisingly, standard classes of counter automata contain already flattable counter systems, see many examples in [LER 05].

**THEOREM 8.15** *[LER 05] Uniformly reversal-bounded counter automata are uniformly flattable, reversal-bounded initialized counter automata are initially flattable, and the finite unions of bounded languages can be effectively computed.*

This provides an alternative proof for the effective semilinearity of the reachability relation. Indeed, an initialized counter automaton and a finite union of bounded languages can be simulated by an admissible counter system.

### 8.4.2. *Automata-based approach for Presburger arithmetic*

In the previous sections, we have seen that the satisfiability problem for Presburger arithmetic is decidable and many verification problems for subclasses of counter systems can be reduced to this problem. In this section, we shall informally describe the decidability of satisfiability problem for Presburger arithmetic by translation into the non-emptiness problem for finite-state automata. The use of automata for logical decision problems goes back to [BÜC 60a] and we shall provide below the approach by automatic structures developed in [BOU 96, BLU 00] (see also [WOL 95]). The seminal paper [BÜC 60b] describes how to use the automata-based approach to deal with Presburger arithmetic. Of course, other decision procedures exist for Presburger arithmetic: for instance, quantifier elimination method from [RED 78] improves the method developped in [COO 72].

Before presenting the principles of the automata-based approach for Presburger arithmetic, let us mention that in general, the automata-based approach consists in reducing logical problems into automata-based decision problems in order to take advantage of known results from automata theory. Alternatively, this can be viewed as a means to transform declarative statements (typically formulae) into operational devices (typically automata with sometimes rudimentary computational power). The most standard target problems in automata used in this approach is the non-emptiness problem that checks whether an automaton admits at least one accepting computation. A pioneering work by Büchi [BÜC 60a] show that Büchi automata are equivalent to formulae in monadic second-order logic (MSO) over $(\mathbb{N}, <)$; models of a formula built over the second-order variables $P_1, \ldots, P_N$ are $\omega$-sequences over the alphabet $\mathcal{P}(\{P_1, \ldots, P_N\})$. In full generality, the following are a few desirable properties of the approach:

– the reduction should be conceptually simple, apart from being semantically faithful;

– the computational complexity of the automata-based target problem should be well-characterized. In that way, a complexity upper bound is obtained to solve the source logical problem;

– last but not least, the reduction should preferably allow the optimal complexity for the source logical problem to be obtained.

We have seen that each Presburger formula $\varphi$ with $n \geq 1$ free variables defines a subset of $\mathbb{N}^n$, namely $\mathrm{REL}(\varphi) \subseteq \mathbb{N}^n$, that corresponds to the set of variable valuations that make $\varphi$ true. For instance, $\mathrm{REL}(\mathsf{x} = \mathsf{y}+\mathsf{z}) = \{(k_1, k_2, k_3) \in \mathbb{N}^3 : k_1 = k_2+k_3\}$. The automata-based approach for Presburger arithmetic consists of representing the tuples in $\mathrm{REL}(\varphi)$ by a regular language that can be effectively defined, for instance with the help of a finite-state automaton. In that way, satisfiability of $\varphi$, which is equivalent to the non-emptiness of $\mathrm{REL}(\varphi)$, becomes equivalent to the non-emptinesss of a finite-state automaton (which is an easy problem to solve once the automaton is built). In order to define regular languages, first we need to specify how natural numbers and tuples of natural numers are encoded by words over a finite alphabet. Numerous options are possible (see e.g. [LER 03, KLA 04b]) and below we adopt a simple and standard encoding in which natural numbers are viewed as finite words over the alphabet $\{0, 1\}$ by using a binary representation in which the least significant bit is first. We adopt a representation of natural numbers that is not unique, for instance the number five can be encoded by $101$ or by $101000$. Tuples of natural numbers of dimension $n$ are represented by finite words over the alphabet $\{0, 1\}^n$ by using an equal length representation for each number. Typically, the pair $\left(\begin{smallmatrix}5\\8\end{smallmatrix}\right)$ can be represented by the word $\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)$ over the alphabet $\{0, 1\}^2$. So, we introduce the map $f : \mathbb{N} \to \mathcal{P}(\{0, 1\}^*)$ such that $f(0) \stackrel{\text{def}}{=} 0^*$ and for $k > 0$, $f(k) \stackrel{\text{def}}{=} b_k \cdot 0^*$ where $b_k$ is the shortest binary representation of $k$ (least significant bit first). The map $f$ is extended to subsets of $\mathbb{N}$ in the obvious way as well as to $n$-tuples of natural numbers with alphabet $\{0, 1\}^n$ such that $f(\vec{x}) \subseteq \mathcal{P}((\{0, 1\}^n)^*)$ with $\vec{x} \in \mathbb{N}^n$ and $\vec{b} \in f(\vec{x})$ iff for $i \in [1, n]$, the projection of $\vec{b}$ on the $i$th row belongs to $f(\vec{x}(i))$. The map $f$ is typically a state-encoding schema in the sense of [BOI 98, LEG 08].

Given a Presburger formula $\varphi$ with $n \geq 1$ free variables and a finite-state automaton $\mathcal{A}$ over the alphabet $\{0, 1\}^n$, we write $\varphi \approx \mathcal{A}$ whenever $\mathrm{L}(\mathcal{A}) = f(\mathrm{REL}(\varphi))$.

**THEOREM 8.16** *(see e.g. [BOU 96]) Given a Presburger formula $\varphi$, we can build a finite-state automaton $\mathcal{A}_\varphi$ such that $\varphi \approx \mathcal{A}_\varphi$.*

We also have $\mathrm{REL}(\varphi) \subseteq \mathrm{REL}(\psi)$ iff $\mathrm{L}(\mathcal{A}_\varphi) \subseteq \mathrm{L}(\mathcal{A}_\psi)$ (see e.g., [LEG 08, theorem 3.22]).

The finite-state automaton $\mathcal{A}_\varphi$ can be built recursively over the structure of $\varphi$. For instance, conjunction is handled by the product construction, existential quantifier is handled by projection, negation is handled by the complement construction, see details below. Nevertheless, a crude complexity analysis of the construction of $\mathcal{A}_\varphi$ reveals a non-elementary worst-case complexity. Indeed, for every negation, a complementation needs to be operated. However, developments related to the optimal size of automata can be found in [KLA 04a].

The recursive definition is based on the following properties. Let $\varphi$ and $\psi$ be Presburger formulae with free variables $\mathsf{x}_1, \ldots, \mathsf{x}_n$.

*conjunction* If $\varphi \approx \mathcal{A}$ and $\psi \approx \mathcal{B}$, then $\varphi \wedge \psi \approx \mathcal{A} \cap \mathcal{B}$ where $\cap$ is the product construction computing intersection;

*negation* If $\varphi \approx \mathcal{A}$, then $\neg\varphi \approx \overline{\mathcal{A}}$ where $\bar{\cdot}$ performs complementation, which may cause an exponential blow-up;

*quantification* If $\varphi \approx \mathcal{A}$, then $\exists\ \mathsf{x}_n\ \varphi \approx \mathcal{A}'$ where $\mathcal{A}'$ is built over the alphabet $\{0,1\}^{n-1}$ by forgetting the $n$th component. Typically, $q \xrightarrow{\vec{b}} q'$ in $\mathcal{A}'$ whenever there is a transition $q \xrightarrow{\vec{b'}} q'$ in $\mathcal{A}$ such that $\vec{b}$ and $\vec{b'}$ agree on the $n-1$ first bit values.

In the above construction, we assumed that $\varphi$ and $\psi$ share the same set of free variables, which does not always hold true for arbitrary formulae. If it is not the case, $\varphi \approx \mathcal{A}$ and $\psi \approx \mathcal{B}$, then we perform an operation that consists of adding dummy bits. For instance, suppose that $\varphi$ contains the free variables $\mathsf{x}_1, \ldots, \mathsf{x}_n$. We can build the automaton $\mathcal{A}'$ over the alphabet $\{0,1\}^{n+1}$ obtained by adding the $n+1$th component. Typically, $q \xrightarrow{\vec{b}} q'$ in $\mathcal{A}'$ whenever there is a transition $q \xrightarrow{\vec{b'}} q'$ in $\mathcal{A}$ such that $\vec{b}$ and $\vec{b'}$ agree on the $n$ first bit values. It remains to deal with atomic formulae to achieve the inductive building of the automaton.

The proof of theorem 8.16 is clearly based on the above constructions but we need to complete the argument in order to deal with atomic formulae. Without any loss of generality, we can restrict ourselves to equalities of the form $\mathsf{x} = \mathsf{y} + \mathsf{z}$ (at the cost of introducing new variables in order to deal with sums made of more than two variables). Such a restriction is only helpful to simplify the presentation of the method but it makes sense to consider the full language with linear constraints in order to optimize the reduction to automata, see e.g. [BOI 02, BOU 96]. The variables in $\mathsf{x} = \mathsf{y} + \mathsf{z}$ are not necessarily distinct.

The automaton for $\mathsf{x}_1 = \mathsf{x}_2 + \mathsf{x}_3$ is described in the left part of Figure 8.16 where $q_1$ is the initial state as well as the final state. The state $q_1$ represents a carry-over of 0 whereas the state $q_2$ represents a carry-over of 1. We can check that $(\mathsf{x}_1 = \mathsf{x}_2 + \mathsf{x}_3) \approx \mathcal{A}$. The right part of Figure 8.16 describes the automaton for $\mathsf{x}_1 = \mathsf{x}_2 + \mathsf{x}_2$.

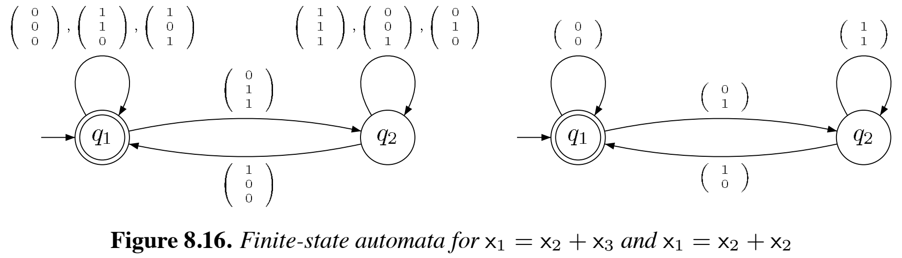

$\begin{pmatrix}0\\0\\0\end{pmatrix}, \begin{pmatrix}1\\1\\0\end{pmatrix}, \begin{pmatrix}1\\0\\1\end{pmatrix}$
$\begin{pmatrix}1\\1\\1\end{pmatrix}, \begin{pmatrix}0\\0\\1\end{pmatrix}, \begin{pmatrix}0\\1\\0\end{pmatrix}$
$\begin{pmatrix}0\\1\\1\end{pmatrix}$
$q_1$
$q_2$
$\begin{pmatrix}1\\0\\0\end{pmatrix}$
$\begin{pmatrix}0\\0\end{pmatrix}$
$\begin{pmatrix}1\\1\end{pmatrix}$
$\begin{pmatrix}0\\1\end{pmatrix}$
$q_1$
$q_2$
$\begin{pmatrix}1\\0\end{pmatrix}$

**Figure 8.16.** *Finite-state automata for* $\mathsf{x}_1 = \mathsf{x}_2 + \mathsf{x}_3$ *and* $\mathsf{x}_1 = \mathsf{x}_2 + \mathsf{x}_2$

The automata-based approach for Presburger arithmetic can be extended to richer theories such as $(\mathbb{R}, \mathbb{Z}, +, \leq)$, see e.g. [BOI 02], or can be refined by providing other reductions, see e.g. [LER 03, KLA 04a, SCH 07]. An overview of automata-based decision procedures for Presburger arithmetic and related formalisms can be found in [KLA 04a].

### 8.4.3. *A selection of tools for Presburger arithmetic*

So far, we have seen how to reduce verification problems into the satisfiability problem for Presburger arithmetic. Then, we presented the principle of an automata-based decision procedure for Presburger arithmetic by viewing sets of tuples defined in Presburger arithmetic as regular languages. Below, we provide a (non-exhaustive) list of tools that can handle first-order logics with linear arithmetic. In that way, we have provided the natural set of steps to perform formal verification of infinite-state systems dealing with counters:

– LIRA implements decision procedures based on automata-theoretic techniques for the first-order theory of $(\mathbb{Z}, +, <)$ and for other related logics with linear arithmetic [BEC 07]. It is closely related to MONA [BIE 96], LASH [BOI 01] and PRESTAF [COU 05]. Contrary to numerous SMT solvers, LIRA can handle quantifiers (this is also true for MONA and LASH very briefly described below);

– the MONA tool provides an implementation for the automata-based decision procedure for weak monadic second-order logic WS1S [BIE 96]. The logic WS1S is known to be strongly related to Presburger arithmetic, see e.g. [BÜC 60b];

– LASH is an automata library that provides the implementation of standard constructions on automata [BOI 01] as well as constructions for linear inequations. Comparisons of data structures used in MONA and LASH can be found in [KLA 04a, Chapter 5]. As an application domain, LASH has been used successfully to verify properties on counter systems, see e.g., [BOI 98];

– TAPAS is a suite of libraries [LER 09] dedicated to first-order logics of linear arithmetic. The application programming interface GENEPI for such logics encapsulate many standard solvers such as LIRA or MONA. FAST [BAR 06] is a tool over TAPAS that is designed to verify reachability properties of counter systems; this is a client application in TAPAS;

– the tool CVC3 is an automatic theorem prover for Satisfiability Modulo first-order Theories (SMT), see e.g., [BAR 08], [BAR 07]. CVC3 is the last offspring of a series of popular SMT provers, which originated at Stanford University with the SVC system. In particular, it builds on the code base of CVC Lite, its most recent predecessor. The automatic theorem prover CVC3 (and the new version CVC4) is a tool that can prove the validity of first-order formulae in a large number of built-in logical theories, including rational and integer linear arithmetic, arrays, tuples, bit vectors, etc;

– Z3 is an efficient SMT solver, see e.g., [MOU 08], that can deal with linear real and integer arithmetic. This is an SMT solver developed by Microsoft Research that is freely available for academic research. Z3 is designed to tackle problems that arise in software verification and software analysis.

## 8.5. Concluding remarks

The verification of infinite-state systems is a very tough problem for which decision procedures do not always exist. In this chapter, we have illustrated the verification methods for such systems on recursive Petri nets and on subclasses of counter systems.

As far as SRPNs are concerned, we have seen that the addition of recursion to Petri nets increases the expressive power of the computational model even though some of the verification problems remain decidable, such as the reachability problem. The decidability proof for that problem on SRPNs uses a solver for the reachability problem for standard Petri nets as a blackbox. Similarly, the computation of linear invariants for Petri nets can be adapted to SRPNs as shown in section 8.3.

As far as counter systems are concerned, we have shown how to reduce a verification problem in a subclass of counter systems (for instance for the reversal-bounded counter automata) into satisfiability in some first-order theory. In order to solve the instances of the logical problem, one option consists of eliminating quantifiers and then using dedicated SMT solvers such as Z3 or CVC3. Alternatively, a Presburger formula can be effectively transformed into a finite-state automaton such that satisfiability is equivalent to the non-emptiness of the language recognized by the automaton. This allows the use of tools dedicated to decision procedures for automata such as LIRA or LASH. Alternatively, the formula can be given to a suite of libraries such as TAPAS and then satisfiability can be checked with any standard solver that is plugged in.

At some abstract level, similar ideas can be found to verify timed systems with real-time constraints (timed automata, timed Petri nets, see e.g. Chapter 9) or pushdown systems even though the methods are undertaken differently. The wealth of infinite-state systems as well as the diversity of properties that requires verification has induced the development of numerous methods and tools even though two central problems always need to be solved in order to run verification tools:

1) how do you symbolically represent an infinite set (configurations, processes, data)?

2) which data structures allow you to represent concisely such sets (when possible) in order to effectively manipulate the symbolic representations in verification tools?

## 8.6. Bibliography

[ABD 96] ABDULLA P., JONSSON B., “Verifying programs with unreliable channels”, *Information and Computation*, vol. 127, num. 2, p. 91–101, 1996.

[ALU 94] ALUR R., DILL D., "A theory of timed automata", *Theoretical Computer Science*, vol. 126, p. 183–235, 1994.

[ARA 77] ARAKI T., KASAMI T., "Decidability problems on the strong connectivity of Petri net reachability sets", *Theoretical Computer Science*, vol. 4, p. 99–119, 1977.

[BAR 06] BARDIN S., LEROUX J., POINT G., "FAST extended release", in *CAV'06*, vol. 4144 of *Lecture Notes in Computer Science*, Springer, p. 63–66, 2006.

[BAR 07] BARRETT C., TINELLI C., "CVC3", in *CAV'07*, vol. 4590 of *Lecture Notes in Computer Science*, Springer, p. 298–302, 2007.

[BAR 08] BARRETT C., SEBASTIANI R., SESHIA S., TINELLI C., "*Satisfiability modulo theories*", vol. 185 of *Frontiers in Artificial Intelligence and Applications*, Chapter 26, p. 825–885, IOS Press, 2008.

[BEC 07] BECKER B., DAX C., EISINGER J., KLAEDTKE F., "LIRA: handling constraints of linear arithmetics over the integers and the reals", in *CAV'07*, vol. 4590 of *Lecture Notes in Computer Science*, Springer, p. 307–310, 2007.

[BER 80] BERMAN L., "The complexity of logical theories", *Theoretical Computer Science*, vol. 11, p. 71–78, 1980.

[BER 01] BERARD B., BIDOIT M., FINKEL A., LAROUSSINIE F., PETIT A., PETRUCCI L., SCHNOEBELEN P., *Systems and Software Verification, Model-checking Techniques and Tools*, Springer, 2001.

[BIE 96] BIEHL M., KLARLUND N., RAUHE T., "Mona: decidable arithmetic in practice", in *FTRTFT'96*, vol. 1135 of *Lecture Notes in Computer Science*, Springer, p. 459–462, 1996.

[BLU 00] BLUMENSATH A., GRÄDEL E., "Automatic structures", in *LICS'00*, p. 51–62, 2000.

[BOI 98] BOIGELOT B., Symbolic methods for exploring infinite state spaces, PhD thesis, University of Liège, 1998.

[BOI 01] BOIGELOT B., JODOGNE S., WOLPER P., "On the use of weak automata for deciding linear arithmetic with integer and real variables", in *IJCAR'01*, vol. 2083 of *Lecture Notes in Artificial Intelligence*, Springer, p. 611–625, 2001.

[BOI 02] BOIGELOT B., WOLPER P., "Representing arithmetic constraints with finite automata: an overview", in *ICLP'02*, vol. 2401 of *Lecture Notes in Computer Science*, Springer, p. 1–19, 2002.

[BOR 76] BOROSH I., TREYBIG L., "Bounds on positive integral solutions of linear diophantine equations", *AMS*, vol. 55, p. 299–304, 1976.

[BOU 96] BOUDET A., COMON H., "Diophantine equations, Presburger arithmetic and finite automata", in *CAAP'96*, vol. 1059 of *Lecture Notes in Computer Science*, Springer, p. 30–43, 1996.

[BOU 06a] BOUAJJANI A., BOZGA M., HABERMEHL P., IOSIF R., MORO P., VOJNAR T., "Programs with lists are counter automata", in *CAV'06*, vol. 4144 of *Lecture Notes in Computer Science*, Springer, p. 517–531, 2006.

[BOU 06b] BOUAJJANI A., HABERMEHL P., ROGALEWICZ A., VOJNAR T., "Abstract tree regular model checking of complex dynamic data structures", *SAS'06*, vol. 4134 of *Lecture Notes in Computer Science*, Springer, p. 52–70, 2006.

[BOZ 10] BOZGA M., IOSIF R., KONEČNÝ F., "Fast acceleration of ultimately periodic relations", in *CAV'10*, vol. 6174 of *Lecture Notes in Computer Science*, Springer, p. 227–242, 2010.

[BÜC 60a] BÜCHI J., "On a decision method in restricted second-order arithmetic", *in Logic, Methodology, and Philosophy of Science*, p. 1–11, 1960.

[BÜC 60b] BÜCHI R., "Weak second-order arithmetic and finite automata", *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, vol. 6, p. 66–92, 1960.

[CAS 99] CASSANDRAS C. G., LAFORTUNE S., *Introduction to Discrete Event Systems*, Kluwer Academic Publishers, 1999.

[CLA 00a] CLARKE E., GRUMBERG O., JHA S., LU Y., VEITH H., "Counter-example-guided abstraction refinement", in *CAV'00*, vol. 1855 of *Lecture Notes in Computer Science*, Springer, p. 154–169, 2000.

[CLA 00b] CLARKE E., GRUMBERG O., PELED D., *Model Checking*, The MIT Press Books, 2000.

[COL 90] COLOM J. M., SILVA M., "Convex geometry and semiflows in P/T nets. A comparative study of algorithms for computation of minimal P-semiflows", *in Advances in Petri Nets*, vol. 483 of *Lecture Notes Computer Science*, Springer-Verlag, p. 79–112, June 1990.

[COM 98] COMON H., JURSKI Y., "Multiple counters automata, safety analysis and Presburger arithmetic", in *CAV'98*, vol. 1427 of *Lecture Notes in Computer Science*, Springer, p. 268–279, 1998.

[COM 00] COMON H., CORTIER V., "Flatness is not a weakness", in *CSL'00*, vol. 1862 of *Lecture Notes in Computer Science*, Springer, p. 262–276, 2000.

[COO 72] COOPER D., "Theorem proving in arithmetic without multiplication", *Machine Learning*, vol. 7, p. 91–99, 1972.

[COR 02] CORTIER V., "About the decision of reachability for register machines", *Theoretical Informatics and Applications*, vol. 36, num. 4, p. 341–358, 2002.

[COU 05] COUVREUR J., "A BDD-like implementation of an automata package", in *CIAA'04*, vol. 3317 of *Lecture Notes in Computer Science*, Springer, p. 310–311, 2005.

[DAN 01] DANG Z., IBARRA O., SAN PIETRO P., "Liveness verification of reversal-bounded multicounter machines with a free counter", in *FST&TCS'01*, vol. 2245 of *Lecture Notes in Computer Science*, Springer, p. 132–143, 2001.

[DEM 06] DEMRI S., FINKEL A., GORANKO V., VAN DRIMMELEN G., "Towards a model-checker for counter systems", in *ATVA'06*, vol. 4218 of *Lecture Notes in Computer Science*, Springer, p. 493–507, 2006.

[El 95] EL FALLAH SEGHROUCHNI A., HADDAD S., "A Formal model for coordinating plans in multiagents systems", in *Proceedings of Intelligent Agents Workshop*, Oxford, United Kingdom, Augusta Technology Ltd, Brooks University, 1995.

[El 96] EL FALLAH SEGHROUCHNI A., HADDAD S., "A recursive model for distributed planning", *Second International Conference on Multi-Agent Systems*, Kyoto, Japan, 1996.

[ESP 94] ESPARZA J., NIELSEN M., "Decidability issues for Petri nets - a survey", *Bulletin of the European Association for Theoretical Computer Science*, vol. 52, p. 245–262, 1994.

[ESP 97] ESPARZA J., "Petri nets, commutative context-free grammars, and basic parallel processes", *Fundamenta Informaticae*, vol. 31, num. 13, p. 13–26, 1997.

[ESP 99] ESPARZA J., FINKEL A., MAYR R., "On the verification of broadcast protocols", in *LICS'99*, p. 352–359, 1999.

[FER 79] FERRANTE J., RACKOFF C., *The Computational Complexity of Logical Theories*, vol. 718 of *Lecture Notes in Mathematics*, Springer, 1979.

[FIN 97] FINKEL A., WILLEMS B., WOLPER P., "A direct symbolic approach to model checking pushdown systems", in *INFINITY'97*, vol. 9 of *Electronic Notes in Theoretical Computer Science*, Elsevier Science Publishers, 1997.

[FIN 00] FINKEL A., SUTRE G., "Decidability of reachability problems for classes of two counter automata", in *STACS'00*, vol. 2256 of *Lecture Notes in Computer Science*, Springer, p. 346–357, 2000.

[FIN 01] FINKEL A., SCHNOEBELEN P., "Well-structured transitions systems everywhere!", *Theoretical Computer Science*, vol. 256, num. 1–2, p. 63–92, 2001.

[FIN 02] FINKEL A., LEROUX J., "How to compose Presburger accelerations: applications to broadcast protocols", in *FST&TCS'02*, vol. 2256 of *Lecture Notes in Computer Science*, Springer, p. 145–156, 2002.

[FIS 74] FISCHER M., RABIN M., "Super-exponential complexity of Presburger arithmetic", in *Complexity of Computation*, vol. 7 of *SIAM-AMS proceedings*, American Mathematical Society, p. 27–42, 1974.

[GAN 09] GANTY P., MAJUMDAR R., RYBALCHENKO A., "Verifying liveness for asynchronous programs", in *POPL'09*, ACM, p. 102–113, 2009.

[GIN 66] GINSBURG S., SPANIER E., "Semigroups, Presburger formulas and languages", *Pacific Journal of Mathematics*, vol. 16, num. 2, p. 285–296, 1966.

[GRÄ 88] GRÄDEL E., "Subclasses of Presburger arithmetic and the polynomial-time hierarchy", *Theoretical Computer Science*, vol. 56, p. 289–301, 1988.

[GUR 81] GURARI E., IBARRA O., "The complexity of decision problems for finite-turn multicounter machines", in *ICALP'81*, vol. 115 of *Lecture Notes in Computer Science*, Springer, p. 495–505, 1981.

[HAD ] HADDAD S., POITRENAUD D., Decidability and undecidability results for recursive Petri nets, Report , University.

[HAD 99] HADDAD S., POITRENAUD D., "Theoretical aspects of recursive Petri nets", *in Proc. of the* $20^{th}$ *Int. Conf. on Applications and Theory of Petri nets*, vol. 1639 of *Lecture Notes in Computer Science*, Williamsburg, VA, USA, Springer-Verlag, p. 228–247, 1999.

[HAD 00] HADDAD S., POITRENAUD D., "Modelling and analyzing systems with recursive Petri nets", in *Proc. of the* $5^{th}$ *Workshop on Discrete Event Systems - Analysis and Control*, Gand, Belgique, Kluwer Academics Publishers, p. 449–458, 2000.

[HAD 01] HADDAD S., POITRENAUD D., "Checking linear temporal formulas on sequential recursive Petri nets", in *Proc of the* $8^{th}$ *International Symposium on Temporal Representation and Reasonning*, Cividale del Friuli, Italy, IEEE Computer Society Press, 2001.

[HAD 07] HADDAD S., POITRENAUD D., "Recursive Petri nets – theory and application to discrete event systems", *Acta Informatica*, vol. 44, num. 7–8, p. 463–508, Springer, December 2007.

[HEN 03] HENZINGER T., JHALA R., MAJUMDAR R., SUTRE G., "Software verification with BLAST", in *SPIN'03*, vol. 2648 of *Lecture Notes in Computer Science*, Springer, p. 235–239, 2003.

[HEN 05] HENZINGER T., MAJUMDAR R., RASKIN J., "A classification of symbolic transitions systems", *ACM Transactions on Computational Logic*, vol. 6, num. 1, p. 1–32, 2005.

[HOP 79] HOPCROFT J., PANSIOT J., "On the reachability problem for 5-dimensional vector addition systems", *Theoretical Computer Science*, vol. 8, p. 135–159, 1979.

[IBA 78] IBARRA O., "Reversal-bounded multicounter machines and their decision problems", *Journal of the Association for Computing Machinery*, vol. 25, num. 1, p. 116–133, 1978.

[JAN 79] JANTZEN M., "On the hierarchy of Petri net languages", *RAIRO*, vol. 13, num. 1, p. 19–30, 1979.

[KAI 10] KAISER A., KROENING D., WAHL T., "Dynamic cutoff detection in parameterized concurrent programs", in *CAV'10*, vol. 6174 of *Lecture Notes in Computer Science*, Springer, p. 645–659, 2010.

[KAR 69] KARP R. M., MILLER R. E., "Parallel program schemata", *Journal of Computer and System Sciences*, vol. 3, num. 2, p. 147–195, 1969.

[KIE 89] KIEHN A., "Petri nets systems and their closure properties", in *Advances in Petri Nets 1989*, vol. 424 of *Lecture Notes in Computer Science*, Springer-Verlag, p. 306-328, 1989.

[KLA 04a] KLAEDTKE F., Automata-based decision procedures for weak arithmetics, PhD thesis, Institut für Informatik, Albert-Ludwigs-University, Freiburg, February 2004.

[KLA 04b] KLAEDTKE F., "On the automata size for Presburger arithmetic", in *LICS'04*, IEEE, p. 110–119, 2004.

[KOS 82] KOSARAJU R., "Decidability of reachability in vector addition systems", in *STOC'82*, p. 267–281, 1982.

[LEG 08] LEGAY A., Generic methods for the verification of infinite-state systems, PhD thesis, University of Liège, 2008.

[LER 03] LEROUX J., Algorithmique de la vérification des systèmes à compteurs. approximation et accélération. implémentation de l'outil FAST., PhD thesis, ENS de Cachan, France, 2003.

[LER 05] Leroux J., Sutre G., “Flat counter systems are everywhere!”, in *ATVA’05*, vol. 3707 of *Lecture Notes in Computer Science*, Springer, p. 489–503, 2005.

[LER 09] Leroux J., Point G., “TaPAS: the Talence Presburger arithmetic suite”, in *TACAS’09*, vol. 5505 of *Lecture Notes in Computer Science*, Springer, p. 182–185, 2009.

[LIP 76] Lipton R. J., The reachability problem requires exponential space, Report num. 62, Department of Computer Science, Yale University, 1976.

[MAN 77] Mandel A., Simon I., “On finite semigroups of matrices”, *Theoretical Computer Science*, vol. 5, num. 2, p. 101–111, 1977.

[MAY 84] Mayr E., “An algorithm for the general Petri net reachability problem”, *SIAM Journal of Computing*, vol. 13, num. 3, p. 441–460, 1984.

[MAY 97] Mayr R., “Combining Petri nets and PA-processes”, in *Proc. of the* $3^{rd}$ *Int. Symposium on Theoretical Aspects of Computer Software*, vol. 1281 of *Lecture Notes in Computer Science*, Sendai, Japan, Springer-Verlag, p. 547–561, 1997.

[MCM 93] McMillan K., *Symbolic Model Checking*, Kluwer Academic Publishers, 1993.

[MIN 67] Minsky M., *Computation: Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, NJ, 1967.

[MOU 08] de Moura L., Björner N., “Z3: An efficient SMT solver”, in *TACAS’08*, vol. 4963 of *Lecture Notes in Computer Science*, Springer, p. 337–340, 2008.

[OPP 78] Oppen D., “A $2^{2^{2pn}}$ upper bound on the complexity of Presburger arithmetic”, *Journal of Computer and System Sciences*, vol. 16, num. 3, p. 323–332, 1978.

[PAP 81] Papadimitriou C., “On the complexity of integer programming”, *Journal of the Association for Computing Machinery*, vol. 28, num. 4, p. 765–768, 1981.

[PAR 66] Parikh R., “On context-free languages”, *Journal of the Association for Computing Machinery*, vol. 13, num. 4, p. 570–581, 1966.

[PET 81] Peterson J., *Petri Net Theory and the Modelling of Systems*, Prentice-Hall, 1981.

[PRE 29] Presburger M., “Über die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchem die addition als einzige operation hervortritt”, *Comptes Rendus du premier congrès de mathématiciens des Pays Slaves, Warsaw*, p. 92–101, 1929.

[RAC 78] Rackoff C., “The covering and boundedness problems for vector addition systems”, *Theoretical Computer Science*, vol. 6, num. 2, p. 223–231, 1978.

[RED 78] Reddy C., Loveland W., “Presburger arithmetic with bounded quantifier alternation”, in *STOC’78*, ACM press, p. 320–325, 1978.

[REI 98] Reisig W., Rozenberg G., Eds., *Lectures on Petri Nets I: Basic Models*, vol. 1491 of *Lecture Notes in Computer Science*, Springer, 1998.

[REU 90] Reutenauer C., *The Mathematics of Petri Nets*, Masson and Prentice, 1990.

[SCH 07] Schuele T., Schneider K., “Verification of data paths using unbounded integers: automata strike back”, in *HVC’06*, vol. 4383 of *Lecture Notes in Computer Science*, Springer, p. 65–80, 2007.