THE BULLETIN OF SYMBOLIC LOGIC
Volume 16, Number 4, Dec. 2010

WELL (AND BETTER) QUASI-ORDERED TRANSITION SYSTEMS

PAROSH AZIZ ABDULLA

Abstract. In this paper, we give a step by step introduction to the theory of *well quasi-ordered* transition systems. The framework combines two concepts, namely (i) transition systems which are *monotonic* wrt. a *well-quasi ordering*; and (ii) a scheme for symbolic *backward* reachability analysis. We describe several models with infinite-state spaces, which can be analyzed within the framework, e.g., Petri nets, lossy channel systems, timed automata, timed Petri nets, and multiset rewriting systems. We will also present *better quasi-ordered* transition systems which allow the design of efficient symbolic representations of infinite sets of states.

§1. Introduction.

1.1. Background. Current capabilities in computer technology allow enormously complicated implementations of such systems, making the task of producing error-free products more and more difficult. Consequently, it is of great practical and economical importance to develop methods which make the design process less error-prone. In other words, there is a real need of techniques for rigorous verification of software in order to complement testing and guarantee a higher degree of reliability. It is now widely accepted that validation methods should be *automatic*; this would allow engineers to perform verification (like they perform compilation) on programs without needing to be familiar with the complex constructions and algorithms behind the tools.

1.2. Finite-state systems. Some of the most notable advances in the area of automated (algorithmic) verification during the last 20 years have been achieved in the area of *finite-state systems*. This success has largely been due to the invention of *model checking* [19, 36]. In model checking, the system is modelled as a finite graph where the nodes represent the states (sometimes referred to as *configurations*) of the program, and the edges encode a transition relation $\longrightarrow$ between configurations. The size and complexity of applications which can be handled have increased rapidly through integration with symbolic techniques such as BDDs [16, 17, 32], and (more recently) through the use of SAT-solvers [14]. Existing tools can now routinely handle systems with millions of states. These methods are designed

Received November 18, 2009.

1079-8986/10/1604-0001/$6.90

to work on finite (but large) state spaces, and have been successfully used in industrial-sized projects, especially in the area of hardware verification.

While the finite-state framework is well suited for reasoning about hardware circuits, it fails to deal with several essential aspects of behaviours for software systems. The reason is that these behaviors involve features which give rise to infinite state spaces. Examples of such features include variables ranging over infinite domains, unbounded communication media, timing constraints, dynamic process creation, parameterization (systems with unbounded numbers of components), multi-threading, and dynamically allocated data structures. Therefore, a large amount of work has been devoted to extending the applicability of model checking to *infinite-state systems*.

1.3. Essentially finite-state systems. One of the first breakthroughs in infinite-state model checking was achieved by Alur, Courcoubetis, and Dill in their classical paper on timed automata [11]. The idea is based on *finite partitioning*. Given a system with infinitely many configurations, we define an equivalence relation $\equiv$ on the set of configurations such that the following two conditions are satisfied:

- $\equiv$ has a finite index (a finite number of equivalence classes).
- $\equiv$ forms a *congruence*, i.e., equivalent configurations make transitions to equivalent configurations. More precisely, if $c_1 \equiv c_2$ and $c_1 \longrightarrow c_3$ (i.e., c_1 can make a transition to c_3) then $c_2 \longrightarrow c_4$ for some $c_4 \equiv c_3$. This condition is equivalent to saying that $\equiv$ is a *bisimulation* wrt. the transition relation $\longrightarrow$.

This means that we can build an abstract finite-state system, where each configuration is the representative of one equivalence class, and where there is a transition from one configuration to another (in the abstract system) if there is a transition between the two corresponding equivalence classes. Models such as timed automata, which allow finite partitioning, are said to be *essentially finite-state* since they allow the extraction of an equivalent finite-state system.

1.4. Well quasi-ordered systems. In this paper, we introduce the basic ingredients of a framework which is widely adopted for infinite-state verification. Compared to finite partitioning, we consider a weaker condition namely that of having a pre-order $\preceq$ rather than an equivalence relation $\equiv$. This gives a more general framework in the following sense:

- Having an equivalence relation is a special case of having a pre-order in the sense that $\equiv$ is also assumed to be symmetric.
- We require that the transition relation is *monotonic* wrt. $\preceq$: if $c_1 \preceq c_2$ and $c_1 \longrightarrow c_3$ then $c_2 \longrightarrow c_4$ for some c_4 with $c_3 \preceq c_4$. This is equivalent to saying that $\preceq$ is a simulation wrt. the transition relation. Notice that in the special case where $\preceq$ is an equivalence relation, the requirement that $\preceq$ is a simulation amounts to the requirement that $\preceq$ is a bisimulation.

- Instead of working with equivalence classes (each represented by one of its configurations), we work with sets of configurations that are upward closed wrt. $\preceq$. Such an upward closed set is represented by one of its minimal elements. Again, we observe that, in the special case where $\preceq$ is an equivalence relation, each upward closed set is an equivalence class and each (minimal) element can be taken to be the representative of the equivalence class.
- We require that $\preceq$ is a *Well Quasi-Ordering* (*WQO* for short). This means that, for any infinite sequence $c_0, c_1, c_2, \ldots$ there are i, j with $i < j$ and $c_i \preceq c_j$. If $\preceq$ is an equivalence relation then the condition of $\preceq$ being a WQO amounts to the equivalence relation having a finite index.

Concretely, our framework is based on *combining* two concepts, namely

1. transition systems which are *monotonic* wrt. a *well-quasi ordering*; and
2. a scheme for symbolic *backward* reachability analysis.

Given a class of models, we define a preorder $\preceq$ on the set of configurations such that (1) $\preceq$ is a simulation relation on the considered models, and (2) $\preceq$ is a WQO. If such a preorder can be defined, then it can be proved that the reachability of an upward-closed set of configurations (wrt. $\preceq$) can be checked algorithmically (automatically). Indeed, (1) monotonicity implies that for any upward-closed set, the set of its predecessors is an upward-closed set, and (2) the fact that $\preceq$ is a WQO implies that every upward-closed set can be characterized by its *finite* set of minimal elements. Therefore, starting from an upward-closed set of configurations U, the iterative computation of the backward reachable configurations from U necessarily terminates since only a finite number of steps are needed to capture all minimal elements of the set of predecessors of U. Obviously, this requires that upward-closed sets can be effectively represented and manipulated (i.e., there are procedures, e.g., for computing immediate predecessors and for checking entailment). This general scheme can be applied for the verification of safety properties since the problem can be reduced to checking the reachability of a set of bad configurations which is typically an upward-closed set wrt. $\preceq$. (For instance, mutual exclusion is violated as soon as there are (at least) two processes in the critical section.)

1.5. A historical perspective. The first paper that suggests combining WQOs with symbolic backward reachability analysis appeared in 1993 [4]. The paper defines the method in the context of *lossy channel systems*. The work in [1], which was published in 1996, extended the method of [4], and presented for the first time the general framework as a tool for model checking of infinite-state systems. The paper (and its journal version [2]) also shows how to apply the algorithms for lossy channel systems, Petri nets, timed automata, and relational automata. In 1998, we applied the framework to derive one of the first positive results for systems which are infinite in two dimensions [6]. More precisely, we presented an algorithm for checking safety properties in systems consisting of arbitrary numbers of processes each with a real-valued clock. In 2000, we modified the framework by using the theory of *Better Quasi-Orderings* (*BQOs*) which is a non-trivial refinement of the theory of WQOs [9]. The BQO approach allows to work with much more efficient symbolic representations than WQOs.

Five years after the publication of [1], the papers [7] and [26] presented in 2001 tutorials and surveys of existing results together with a set of simple extensions of the framework.

In [25] Finkel presented the model of *completely specified protocols* which is very similar to lossy channel systems. However, the paper presents only algorithms for checking termination using *forward reachability analysis*. In particular, the algorithms cannot be used to check safety properties.

Despite its simplicity, the framework of well quasi-ordered transition systems has shown to be quite powerful, and has been applied to derive verification algorithms for numerous models such as broadcast protocols [23], lossy channel systems [4, 5], timed Petri nets [10], cache coherence protocols [21], timed networks [8], multiset rewriting systems [3], and data nets [30].

REMARK. The class of systems we consider in this tutorial is often referred as *well-structured systems* in the literature. However, we avoid this name here to avoid confusion. The term *well-structured systems* has been used to define different types of models. For instance, it is used in [24] to describe systems which are *strictly monotonic*. This is a much stronger condition than monotonicity. Among the models we consider in this paper, only Petri nets satisfy strict monotonicity.

1.6. Outline. In the next Section we introduce several notions which we will use throughout the paper. We will illustrate the main concepts of our methodology in Section 3 through the classical model of Petri nets. In Section 4 we give the formal definition of *well quasi-ordered transition systems*, and present the first version of the algorithm for symbolic backward reachability analysis. In Section 5 we propose a refined version of the algorithm which is more appropriate for implementation. We illustrate how to use the reachability algorithm in order to check safety properties in Section 6. We apply the framework to lossy channel systems and timed automata in Sections 7 resp. 8. In Section 9 we introduce the notion of *constraint systems* which we use to give a symbolic version of the reachability algorithm. In Section 10, we describe a methodology for building more and more complicated well quasi-ordered constraint systems based on Hangman's theorem; and then apply the methodology to build a constraint system for timed Petri nets in Section 11. In Section 12, we explain the role of better quasi-orderings in the design of efficient constraint systems, and then apply them for the verification of timed Petri nets and constraint multiset rewriting systems in Sections 13 resp. 14.

§**2. Preliminaries.** We give preliminary notions and concepts which we will use in the rest of the paper.

2.1. Multisets and words. We use $\mathbb{N}$, $\mathbb{Z}$, and $\mathbb{R}^{\geq 0}$ to represent the set of natural numbers, integers, and non-negative reals respectively. For a set A, we use $A^{\circledast}$ to denote the set of finite multisets over A. We view a multiset over A as a mapping from A to $\mathbb{N}$. Sometimes, we write multisets as lists, so if $a, b \in A$ then $[a, b, b, a, a]$ represents a multiset M over A where $M(a) = 3$, $M(b) = 2$ and $M(x) = 0$ for $x \neq a, b$. We may also write M as $\left[a^3, b^2\right]$. For multisets M_1 and M_2 over $\mathbb{N}$, we write $M_1 \leq M_2$ if $M_1(a) \leq M_2(a)$ for all $a \in A$. We define the addition $M_1 + M_2$ of multisets M_1, M_2 to be the multiset M where $M(a) = M_1(a) + M_2(a)$, and (assuming $M_1 \leq M_2$) we define the subtraction $M_2 - M_1$ to be the multiset M where $M(a) = M_2(a) - M_1(a)$, for each $a \in A$. For natural numbers n_1 and n_2, we define $n_2 \ominus n_1$ to be 0 if $n_1 \geq n_2$ and $n_2 - n_1$ otherwise. We extend the operation $\ominus$ to multisets in an analogous manner to addition and subtraction. We write $a \in M$ to denote that $M(a) > 0$. Sometimes, we interpret a set $B \subseteq A$ as a multiset where $B(a) = 1$ if $a \in B$ and $B(a) = 0$ if $a \notin B$. We use $\emptyset$ to denote the empty multiset, i.e., $\emptyset(a) = 0$ for all $a \in A$; and use $|M|$ to denote the size of M, i.e., $|M| = \sum_{a \in A} M(a)$.

We use A^* to denote the set of finite words over A, and use $w_1 \cdot w_2$ to denote the concatenation of the words w_1 and w_2. Sometimes, we omit the concatenation operator and simply write $w_1 w_2$. We use ε to denote the empty string. If $w = a_1 a_2 \dots a_n \neq \varepsilon$, we define $last(w) := a_n$.

For a natural number $n \in \mathbb{N}$, we use $n^\bullet$ to denote the set $\{1, 2, \dots, n\}$.

2.2. Well quasi-orderings. A *pre-order* $(A, \preceq)$ consists of a set A and a reflexive and transitive relation $\preceq$ on A. If A is known from the context, then we simply represent the pre-order by the relation $\preceq$. We say that $\preceq$ is an equivalence relation if it is also symmetric. We say that $\preceq$ is *decidable* if, given $a, b \in A$, we can algorithmically check whether $a \preceq b$. We write $a \prec b$ to denote that $a \preceq b$ and $b \not\preceq a$. A set $U \subseteq A$ of configurations is said to be *upward closed* (wrt. $\preceq$), if whenever $c \in U$ and $c \preceq c'$ then $c' \in U$. For $a \in A$, we define $\widehat{a} := \{b \mid a \preceq b\}$, i.e., $\widehat{a}$ is the upward closure of a wrt. $\preceq$. For a set $B \subseteq A$, we define $\widehat{B} := \bigcup_{a \in B} \widehat{a}$. For sets $B_1, B_2 \subseteq A$, we use $B_1 \preceq_{\forall\exists} B_2$ to denote that for all $b_2 \in B_2$ there is a $b_1 \in B_1$ with $b_1 \preceq b_2$. Observe that $B_1 \preceq_{\forall\exists} B_2$ iff $\widehat{B_2} \subseteq \widehat{B_1}$.

An infinite sequence $a_0, a_1, a_2, \dots$ of elements in A is said to be *good* if there are i and j such that $i < j$ and $a_i \preceq a_j$. The sequence is called *bad* otherwise. The pre-order $\preceq$ is said to be a *Well Quasi-Ordering* (*WQO* for short) if all infinite sequences over A are good.

For an upward closed set U, we define a *generator* of U to be a set B such that:

- $\widehat{B} = U$, i.e., U can be generated from B by taking the upward closure of B wrt. $\preceq$.

- $a \preceq b$ implies $a = b$ for all $a, b \in B$. In other words, the set B is canonical in the sense that all its elements are incomparable wrt. $\preceq$.

We observe that the set B contains only minimal elements (if $a \prec b$ for some a then $b \notin B$). On the other hand, if $\preceq$ is not anti-symmetric, then the set B need not be unique (given two elements $a \preceq b \preceq a$, then any one of a and b may belong to B). We use *gen* (U) to denote a function which returns a unique generator of U. In other words, if there are are several generators of U, then *gen* (U) gives an arbitrary (but fixed) such generator. If $\preceq$ is a partial order (i.e., it is also anti-symmetric), then there is indeed a unique generator of U.

Assume that $\preceq$ is a WQO. It follows by canonicity that *gen* (U) is finite; otherwise we would have an infinite set of incomparable elements from which we can build a bad sequence. This means that each upward closed set U can be characterized by a *finite* set of configurations, namely its generator *gen* (U). The set *gen* $(U) = \{a_1, \ldots, a_n\}$ is a finite characterization of U in the sense that $U = \widehat{a_1} \cup \cdots \cup \widehat{a_n}$.

§3. Petri Nets.

We illustrate the main ideas of our methodology, using the model of *Petri Nets*. After recalling the standard definitions of Petri nets, we describe the transition system induced by a Petri net. We describe how checking safety properties can be translated to the reachability of sets of configurations which are upward closed wrt. a natural ordering on the set of configurations.[1] Finally, we give a sketch of an algorithm to solve the reachability problem.

3.1. Model. A Petri net $\mathcal{N}$ is a tuple (P, T, F), where P is a finite set of *places*, T is a finite set of *transitions*, and $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation*. If $(p, t) \in F$ then p is said to be an *input* place of t; and if $(t, p) \in F$ then p is said to be an *output* place of t. We use *In* $(t) := \{p \mid (p, t) \in F\}$ and *Out* $(t) := \{p \mid (t, p) \in F\}$ to denote the sets of input places and output places of t respectively.

Figure 1 shows an example of a Petri net with three places (drawn as circles), namely `L`, `W`, and `C`; and two transitions (drawn as rectangles), namely t_1 and t_2. The flow relation is represented by edges from places to transitions, and from transitions to places. For instance, the flow relation in the example includes the pairs $(\mathtt{L}, t_1)$ and $(t_2, \mathtt{W})$, i.e., `L` is an input place of t_1, and `W` is an output place of t_2.

The transition system induced by a Petri net is defined by the set *configurations* together with the *transition relation* defined on them. A *configuration* c of a Petri net[2] is a multiset over P. The configuration c defines the number of *tokens* in each place. Figure 1 (a) shows a configuration where there is

[1] Reachability of upward closed sets of configurations is referred to as the *coverability problem* in the Petri net literature.

[2] A configuration in a Petri net is often called a *marking* in the literature.

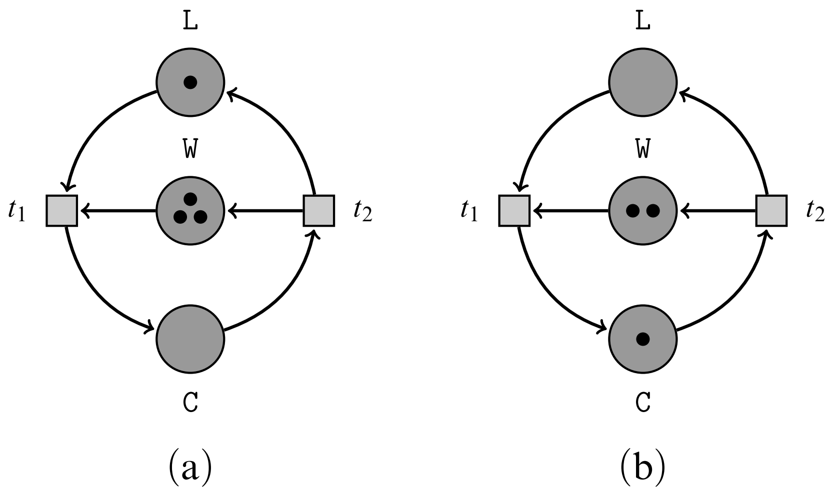

FIGURE 1. (a) A simple Petri net. (b) The result of firing t_1.

one token in place L, three tokens in place W, and no token in place C. The configuration corresponds to the multiset $[\mathtt{L}, \mathtt{W}^3]$.

The operational semantics of a Petri net is defined through the notion of *firing* transitions. This gives a transition relation on the set of configurations. More precisely, when a transition t is fired, then a token is removed from each input place, and a token is added to each output place of t. The transition is fired only if each input place has at least one token. Formally, we write $c_1 \longrightarrow c_2$ to denote that there is a transition $t \in T$ such that $c_1 \geq \mathit{In}(t)$ and $c_2 = c_1 - \mathit{In}(t) + \mathit{Out}(t)$. For sets C_1, C_2 of configurations, we write $C_1 \longrightarrow C_2$ to denote that $c_1 \longrightarrow c_2$ for some $c_1 \in C_1$ and $c_2 \in C_2$. We define $\stackrel{*}{\longrightarrow}$ to the reflexive transitive closure of $\longrightarrow$.

The Petri net of Figure 1 can be seen as a model of a simple mutual exclusion protocol, where access to the critical section is controlled by a global lock. A process is either *waiting* or is in its *critical section*. Initially, all the processes are in their waiting states. When a process wants to access the critical section, it must first acquire the lock. This can be done only if no other process has already acquired the lock. From the critical section, the process eventually releases the lock and moves back to the waiting state. The numbers of tokens in places W and C represent the number of processes in their waiting states and critical sections respectively. Absence of tokens in L means that the lock is currently taken by some process.

The set C_{init} of *initial configurations* are those of the form $[\mathtt{L}, \mathtt{W}^n]$ where $n \geq 0$. In other words, all the processes are initially in their waiting states, and the lock is free. The transition t_1 models a process moving to its critical section, while the transition t_2 models a process going back to its waiting state.

As an example, if we start from the configuration $\left[\mathtt{L}, \mathtt{W}^4\right]$, we can fire the transition t_1 to obtain the configuration $\left[\mathtt{C}, \mathtt{W}^3\right]$ from which we can fire the transition t_2 to obtain the configuration $\left[\mathtt{L}, \mathtt{W}^4\right]$, and so on.

A set C of configurations is said to be *reachable* if $C_{init} \xrightarrow{*} C$.

3.2. Safety properties. We are interested in checking a safety property for the Petri net in Figure 1. In a safety property, we want to show that "nothing bad happens" during the execution of the system. Typically, we define a set *Bad* of configurations, i.e., configurations which we do not want to occur during the execution of the system. In this particular example, we are interested in proving mutual exclusion. The set *Bad* contains those configurations that violate mutual exclusion, i.e., configurations in which at least two processes are in their critical sections. These configurations are of the form $\left[\mathtt{L}^k, \mathtt{W}^m, C^n\right]$ where $n \geq 2$. Checking the safety property can be carried out by checking whether we can fire a sequence of transitions taking us from an initial configuration to a bad configuration, i.e., we check whether the set *Bad* is reachable.

We will work with sets of configurations which are upward closed with respect to $\leq$. Such sets are interesting in our setting since all sets of bad configurations which occur in our examples are upward closed. For instance, in our example, whenever a configuration contains two processes in their critical sections then any larger configuration will also contain (at least) two processes in their critical sections, so the set *Bad* is upward closed. In this manner, checking the safety property amounts to deciding reachability of an upward closed set. Below, we give a sketch of backward reachability algorithm for checking safety properties. In fact, since the ordering $\leq$ is anti-symmetric, it follows that each upward closed set has a unique generator.

3.3. Algorithm. As mentioned above, we are interested in checking whether it is the case that *Bad* reachable. The safety property is violated iff the question has a positive answer. The algorithm, illustrated in Figure 2, starts from the set of bad configurations, and tries to find a path backwards through the transition relation to the set of initial configurations. The algorithm operates on upward closed sets of configurations. An upward closed set is symbolically represented by a finite set of configurations, namely the members of its generator. In the above example, the set *gen* (*Bad*) is the singleton $\left\{\left[\mathtt{C}^2\right]\right\}$. Therefore, the algorithm starts from the configuration $c_0 = \left[\mathtt{C}^2\right]$. From the configuration c_0, we go backwards and derive the generator of the set of configurations from which we can fire a transition and reach a configuration in $Bad = \widehat{c_0}$. Transition t_1 gives the configuration $c_1 = [\mathtt{L}, \mathtt{W}, \mathtt{C}]$, since $\widehat{c_1}$ contains exactly those configurations from which we can fire t_1 and reach a configuration in $\widehat{c_0}$. Analogously, transition t_2 gives the configuration $c_2 = \left[\mathtt{C}^3\right]$, since $\widehat{c_2}$ contains exactly those configurations from which we can fire t_2 and reach a configuration in $\widehat{c_0}$. Since $c_0 \leq c_2$,

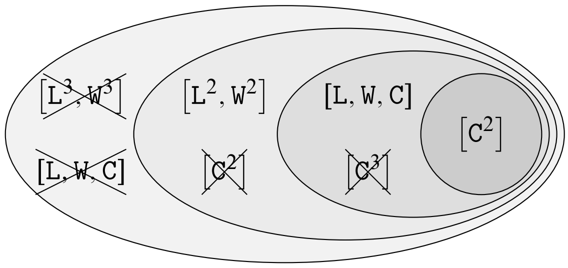

FIGURE 2. Running the backward reachability algorithm on the example Petri net. Each ellipse contains the configurations generated during one iteration. The subsumed configurations are crossed over.

it follows that $\widehat{c_2} \subseteq \widehat{c_0}$. In such a case, we say that c_2 is *subsumed* by c_0. Since $\widehat{c_2} \subseteq \widehat{c_0}$, we can discard c_2 safely from the analysis without the loss of any information. Now, we repeat the procedure on c_1, and obtain the configurations $c_3 = \left[\mathtt{L}^2, \mathtt{W}^2\right]$ (via t_1), and $c_4 = \left[\mathtt{C}^2\right]$ (via t_2), where c_4 is subsumed by c_0. Finally, from c_3 we obtain the configurations $c_5 = \left[\mathtt{L}^3, \mathtt{W}^3\right]$ (via t_1), and $c_6 = [\mathtt{L}, \mathtt{W}, \mathtt{C}]$ (via t_2). The configurations c_5 and c_6 are subsumed by c_3 and c_1 respectively. The iteration terminates at this point since all the newly generated configurations were subsumed by existing ones, and hence there are no more new configurations to consider. In fact, the set $B = \left\{\left[\mathtt{C}^2\right], [\mathtt{L}, \mathtt{W}, \mathtt{C}], \left[\mathtt{L}^2, \mathtt{W}^2\right]\right\}$ is the generator of the set of configurations from which we can reach a bad configurations. The three members in B are those configurations which are not discarded in the analysis (they were not subsumed by other configurations). To check whether *Bad* is reachable, we check the intersection $\widehat{B} \cap C_{init}$. Since the intersection is empty, we conclude that *Bad* is not reachable, and hence the safety property is satisfied by the system.

§4. Well quasi-ordered transition systems. In this section, we introduce *well quasi-ordered transition systems*. Their main characteristic is that they are monotonic wrt. a WQO on the set configurations. We present a scheme for checking reachability of sets configurations which are upward closed wrt. the ordering. From the scheme we extract sufficient conditions which will enable us to transform the scheme into an algorithm. The sufficient conditions are used to give a formal definition of the notion of a well quasi-ordered transition system.

4.1. Transition systems. A transition system $\mathcal{T}$ is a tuple $(C, \longrightarrow, \preceq, C_{init})$, where C is a set of configurations, $\longrightarrow \ \subseteq C \times C$ is a transition relation on C, $\preceq$ is a decidable pre-order on C, and $C_{init} \subseteq C$ is the set of *initial configurations*. We write $c_1 \longrightarrow c_2$ to denote that $(c_1, c_2) \in \longrightarrow$. For sets C_1 and C_2 of configurations, we use $C_1 \longrightarrow C_2$ to denote that there are

$c_1 \in C_1$ and $c_2 \in C_2$ such that $c_1 \longrightarrow c_2$. We use $\xrightarrow{*}$ to denote the reflexive transitive closure of $\longrightarrow$. A set C of configurations is said to be *reachable* if $C_{init} \xrightarrow{*} C$.

4.2. Scheme. We will check safety properties using Scheme 1 for backward reachability analysis.

SCHEME 1 Backward Reachability

Input: • $\mathcal{T} = (C, \longrightarrow, \preceq, C_{init})$: transition system.
• *Bad*: upward closed set of configurations.
Output: Is *Bad* reachable?

$i \leftarrow 0$
$U_0 := Bad$
repeat
    $U_{i+1} \leftarrow U_i \cup Pre(U_i)$
    $i \leftarrow i + 1$
until $U_i = U_{i-1}$
if $C_{init} \cap U_i \neq \emptyset$ then
    return true
else
    return false
end if

The scheme inputs a transition system $\mathcal{T} = (C, \longrightarrow, \preceq, C_{init})$, together with an upward closed set *Bad* of configurations, and checks whether *Bad* is reachable. The basic step in the scheme consists of computing *predecessors*. For a set C of configurations, we define its set of predecessors to be the set $Pre(C) := \{c \mid \exists c' \in C \cdot c \longrightarrow c'\}$. In other words, the set $Pre(C)$ contains exactly all configurations from which we can reach a configuration in C through performing one transition.

In Scheme 1, we start with the set *Bad* of configurations, and apply the function *Pre* repeatedly, generating a sequence $U_0, U_1, U_2, \ldots$ of sets of configurations, where $U_0 := Bad$, and $U_{i+1} := U_i \cup Pre(U_i)$ for $i \geq 0$. We observe that the set U_i characterizes the set of configurations from which the set *Bad* is reachable within i steps. The iteration stops if/when we reach a point $i > 0$ where $U_i = U_{i-1}$. In such a case, the set U_i contains exactly the configurations from which we can reach a bad configuration. The validity of the safety property is then equivalent to the emptiness of the intersection of the sets C_{init} and U_i.

4.3. Algorithm. We extract an algorithm (Algorithm 2) from Scheme 1 by imposing a number of conditions on the transition system $\mathcal{T}$. We collect these conditions in order to define well quasi-ordered transition systems below. First, we require that $\mathcal{T}$ is *monotonic* wrt. the ordering $\preceq$ in the following

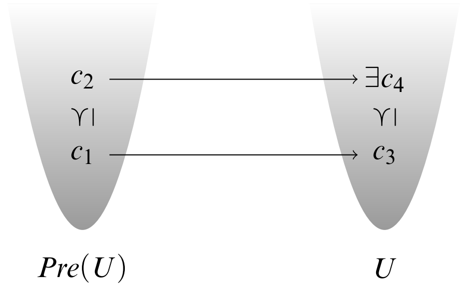

FIGURE 3. Monotonicity, *Pre*, and upward closedness.

sense: for all configurations c_1, c_2, c_3, whenever $c_1 \preceq c_2$ and $c_1 \longrightarrow c_3$ then $c_2 \longrightarrow c_4$ for some $c_4 \succeq c_3$. This is equivalent to saying that $\preceq$ is a simulation wrt. the relation $\longrightarrow$ on configurations.

There is an important relationship between upward closedness, monotonicity, and predecessor sets (illustrated in Figure 3). More precisely, monotonicity implies that upward closedness is preserved by the application of *Pre*. Consider an upward closed set U. Let c_1 be a member of $Pre(U)$ and let $c_2 \succeq c_1$. We will show that c_2 is also a member of $Pre(U)$. Since $c_1 \in Pre(U)$ (by definition), we know by definition that there is a $c_3 \in U$ such that $c_1 \longrightarrow c_3$. By monotonicity it follows that there is a c_4 such that $c_3 \preceq c_4$ and $c_2 \longrightarrow c_4$. From $c_3 \in U$ and $c_3 \preceq c_4$ it follows that $c_4 \in U$. This means that we have found a configuration $c_4 \in U$ such that $c_2 \longrightarrow c_4$, which implies that $c_2 \in Pre(U)$. Since U_0 is upward closed, and the relation $\longrightarrow$ is monotonic, it follows that all the sets U_i which arise in Scheme 1 are upward closed.

The second condition we require is that the pre-order $\preceq$ should be a WQO. From the discussion in Section 2, together with WQO of $\preceq$ and the fact that each U_i is upward closed, it follows that each U_i can be characterized by a *finite* set of configurations, namely any generator of U_i.

To take advantage of monotonicity and WQO of $\preceq$, we define a binary relation $\rightsquigarrow$ on the set of configurations. Intuitively, $c_1 \rightsquigarrow c_2$ iff $c_2 \in gen\,(Pre\,(\widehat{c_1}))$. For a configuration c, we define $(c \rightsquigarrow)$ to be the set $\{c' \mid c \rightsquigarrow c'\}$. For a (finite) set C of configurations, we define $(C \rightsquigarrow) := \bigcup_{c \in C}(c \rightsquigarrow)$. Notice that $(C \rightsquigarrow)$ is a generator of the (upward closed) set of configurations from which we can reach the upward closure of C. In particular, if $C = gen\,(U)$, for some upward closed set U, then $(C \rightsquigarrow)$ is a generator of the set of configurations from which we can reach U.

The idea of Algorithm 2 is to make use of the fact that all the sets U_i are upward closed, and employ configurations (which are members of generators) as symbolic representations of these sets. We input a finite set C_{fin} of *final* configurations that is supposed a generator of *Bad*, i.e., ALGORITHM 2 Backward Reachability

Input: • $\mathcal{T} = (C, \longrightarrow, \preceq, C_{init})$: transition system.
• C_{fin}: finite set of configurations.

Output: Is $\widehat{C_{fin}}$ reachable?

1: $i \leftarrow 0$
2: $C_0 := C_{fin}$
3: repeat
4: $\quad C_{i+1} \leftarrow gen\left(C_i \cup (C_i \rightsquigarrow)\right)$
5: $\quad i \leftarrow i + 1$
6: until $C_{i-1} \preceq_{\forall\exists} C_i$
7: if $\exists c_1 \in C_i \cdot \exists c_2 \in C_{init} \cdot c_1 \preceq c_2$ then
8: $\quad$ return true
9: else
10: $\quad$ return false
11: end if

$C_{fin} = gen\,(Bad)$. Furthermore, we replace the operation *Pre* on upward closed sets, by the operation $\rightsquigarrow$ on finite sets of configurations. Since we take a generator of the set $C_i \cup Pre(C_i)$ it follows that each C_i in Algorithm 2 is a generator of U_i in Scheme 1. Also, by definition we have that $C_i \preceq_{\forall\exists} C_{i-1}$. Therefore, the termination condition of Algorithm 2 is equivalent to $\widehat{C_i} = \widehat{C_{i-1}}$, which is identical to the termination condition of Scheme 1. This means that, upon termination, it is the case that $\widehat{C_i}$ is the set of configurations from which we can reach $Bad = \widehat{C_{fin}}$ Finally, we observe that the conditions of line 7 in both algorithms are equivalent.

Now, we show that the algorithm is guaranteed to terminate. Suppose that the algorithm does not terminate. Since the algorithm does not terminate, for each $i > 0$ there is a configuration c_i such that $c_i \in U_i$ and $c \not\preceq c_i$ for all $c \in U_{i-1}$. This means that the sequence $c_0, c_1, c_2, \ldots$ is bad, which contradicts the assumption that $\preceq$ is a WQO.

4.4. Well quasi-ordered transition systems. We collect the conditions which need to be satisfied by the transition system in order to transform Scheme 1 into Algorithm 2. A *Well Quasi-Ordered Transition Systems* (*or WTS for short*) $(C, \longrightarrow, \preceq, C_{init})$ satisfies the following five conditions:

1. $\mathcal{T}$ is monotonic. This implies that the predecessor set of an upward closed set of configurations is upward closed.
2. $\preceq$ is a WQO. We need this property for two reasons: to represent upward closed sets by a finite set of configurations (a generator of the set); and to guarantee termination of the algorithm.
3. For each c, we can compute the (finite) set $(c \rightsquigarrow)$. This is needed in line 4 of the algorithm.

4. $\preceq$ is decidable. This is needed in line 4 and line 6 of the algorithm. More precisely, we know that both C_i and $(C_i \rightsquigarrow)$ are finite. Therefore, we can compute C_{i+1} by discarding the irrelevant configurations (configurations which are subsumed by smaller ones in the set). We can also check the termination condition by making pairwise comparison of configurations in the sets C_i and C_{i-1}.
5. For each c, we can check whether there is a $c' \in C_{init}$ such that $c \preceq c'$. We need this property to be able to check the condition of line 7 in the algorithm.

This defines a methodology for verification of safety properties for a wide class of computation models. Given a model, we first define the induced transition system by specifying the (infinite) set of configurations, the transition relation, the ordering, and the set of initial configurations. Then, we show that such a transition system is a WTS. Now, we can apply Algorithm 2 to check safety properties.

We take the example of Petri nets. Consider a Petri net $\mathcal{N} = (P, T, F)$. The set of configurations and the transition relation were defined in Section 3. The ordering is the multiset ordering $\leq$ on configurations. The definition of the set of initial configurations depends on the application in question. Our methodology allows us to choose quite powerful theories for specifying sets of initial configurations. For instance, we can use Presburger formulas, where in a Petri net with places $p_1, \ldots, p_n$, the formula $\phi_{Init}(x_1, \ldots, x_n)$ characterizes the set of configurations where the numbers of tokens $x_1, \ldots, x_n$ in the places $p_1, \ldots, p_n$ satisfy the formula. For instance, in the case of mutual exclusion protocol of Section 3, this set contains all configurations of the form $[\mathtt{L}, \mathtt{W}^n]$ where $n \geq 0$. This set is characterized by the formula $(x_1 = 1) \wedge (x_2 \geq 0) \wedge (x_3 = 0)$, where x_1, x_2, x_3 represent the numbers of tokens in the places L, W, and C respectively. The transition system induced by a Petri net is a WTS as follows:

1. The transition relation is monotonic. For configurations c_1, c_2, c_3, if $c_1 \preceq c_2$ and $c_1 \longrightarrow c_3$ then $c_2 \longrightarrow c_3 + c_2 - c_1$. We observe that $c_3 \leq c_3 + c_2 - c_1$. In the example of Figure 1, we have $c_1 = \left[\mathtt{L}, \mathtt{W}^4\right] \longrightarrow \left[\mathtt{C}, \mathtt{W}^3\right] = c_2$. If we take $c_3 = \left[\mathtt{L}^2, \mathtt{W}^4, \mathtt{C}\right] \succeq c_1$ then $c_3 \longrightarrow \left[\mathtt{L}, \mathtt{W}^3, \mathtt{C}^2\right] = c_4 \succeq c_2$.
2. The pre-order $\leq$ on configurations (multisets of natural numbers) is a WQO by Dickson's lemma [22].
3. We define $(c \rightsquigarrow) := \{c' \mid \exists t \in T \cdot c' = c \ominus Out\,(t) + In\,(t)\}$. For instance, in the example of Figure 1, we have $\left[\mathtt{L}^2, \mathtt{W}^2, \mathtt{C}^2\right] \rightsquigarrow \left[\mathtt{L}^3, \mathtt{W}^3, \mathtt{C}\right]$, $\left[\mathtt{L}^2, \mathtt{W}^2, \mathtt{C}\right] \rightsquigarrow \left[\mathtt{L}^3, \mathtt{W}^3\right]$, $\left[\mathtt{L}^2, \mathtt{W}^2\right] \rightsquigarrow \left[\mathtt{L}^3, \mathtt{W}^3\right]$, etc.
4. The ordering $\leq$ on configurations is decidable: Given two configurations c_1 and c_2, we check that $c_1(p) \leq c_2(p)$ for all $p \in P$.
5. Suppose that C_{init} is characterized by a Presburger formula. For each configuration c, we can check whether there is a $c' \in C_{init}$ such that

$c \preceq c'$ as follows. Let the set P of places be $\{p_1, \ldots, p_n\}$. Suppose that C_{init} is characterized by the formula $\phi_{Init}(x_1, \ldots, x_n)$, where x_i corresponds to the number of tokens in place p_i for $i\colon 1 \leq i \leq n$. Let $c(p_i) = k_i$ for $i\colon 1 \leq i \leq n$. Then, there is a $c' \in C_{init}$ such that $c' \preceq c$ iff the formula $\phi_{Init}(x_1, \ldots, x_n) \wedge (x_1 \geq k_1) \wedge \cdots \wedge (x_n \geq k_n)$ is satisfiable. The latter is again a Presburger formula, and hence its satisfiability can be checked. In the example of Figure 1, we can use three variables x_1, x_2, x_3 to denote the number of tokens in the places $\mathtt{L}, \mathtt{W}, \mathtt{C}$ respectively. Then, checking the termination condition of the algorithm amounts to checking the satisfiability of the three formulas

$$(x_1 = 1) \wedge (x_2 \geq 0) \wedge (x_3 = 0) \wedge (x_3 \geq 2),$$
$$(x_1 = 1) \wedge (x_2 \geq 0) \wedge (x_3 = 0) \wedge (x_1 \geq 1) \wedge (x_2 \geq 1) \wedge (x_3 \geq 1),$$
$$(x_1 = 1) \wedge (x_2 \geq 0) \wedge (x_3 = 0) \wedge (x_1 \geq 2) \wedge (x_2 \geq 2).$$

None of these formulas is satisfiable, and hence the safety property is satisfied.

§**5. Refined algorithm.** We present Algorithm 3, a refined version of Algorithm 2 which is more suitable for implementation. In Algorithm 2, all the predecessors of the members of C_i are computed together during the same iteration. Algorithm 3 on the other hand stores the members of the generators in a variable `ToExplore`. The correctness of the algorithm is not dependent on the order in which the configurations are considered. The user may therefore use different strategies to implement `ToExplore`: a queue (which gives a breadth-first search), a stack (which gives a depth-first search), or the configurations may be considered according to certain measures such their sizes, forms, etc. These search strategies give different degrees of efficiency in different applications.

To understand the correctness of the refined algorithm, we refer to the following three lemmata (illustrated in Figure 4) which describe a number of properties of the relations $\longrightarrow$, $\rightsquigarrow$, and $\preceq$. In the lemmata, let c_1, c_2, c_3 be configurations.

$$\begin{tikzcd} c_1 \arrow[r, squiggly] \arrow[d, phantom, "\rotatebox{90}{$\preceq$}"] & c_2 \arrow[d, phantom, "\rotatebox{90}{$\preceq$}"] \\ c_3 \arrow[r, squiggly] & \exists c_4 \end{tikzcd} \qquad \begin{tikzcd} c_1 \arrow[r, squiggly] \arrow[d, phantom, "\rotatebox{-90}{$\preceq$}"] & c_2 \arrow[dl] \\ \exists c_3 & \end{tikzcd} \qquad \begin{tikzcd} c_1 \arrow[r] \arrow[d, phantom, "\rotatebox{90}{$\preceq$}"] & c_3 \arrow[d, phantom, "\rotatebox{90}{$\preceq$}"] \\ \exists c_4 & c_2 \arrow[l, squiggly] \end{tikzcd}$$

FIGURE 4. From left to right: the relations in Lemma 5.1, Lemma 5.2, and Lemma 5.3 respectively.

LEMMA 5.1. *If $c_1 \rightsquigarrow c_2$ and $c_3 \preceq c_1$ then there is a c_4 such that $c_3 \rightsquigarrow c_4$ and $c_4 \preceq c_2$.*

ALGORITHM 3 Refined Backward Reachability

Input: • $\mathcal{T} = (C, \longrightarrow, \preceq, C_{init})$: transition system.
• C_{fin}: finite set of configurations.

Output: Is $\widehat{C_{fin}}$ reachable?

```
ToExplore ← C_fin
Explored := ∅
while ToExplore ≠ ∅ do
   remove some c from ToExplore
   if ∃c′ ∈ C_init · c ⪯ c′ then
      return true
   else if ∃c′ ∈ Explored · c′ ⪯ c then
      discard c
   else
      ToExplore := ToExplore ⋃ {c′ | c ⇝ c′}
      Explored := {c} ⋃ {c′ | c′ ∈ Explored ∧ (c ⋠ c′)}
   end if
end while
return false
```

PROOF. Suppose that $c_1 \rightsquigarrow c_2$ and $c_3 \preceq c_1$. Since $c_1 \rightsquigarrow c_2$ it follows by definition that $c_2 \in gen\,(Pre\,(\widehat{c_1}))$ and hence $c_2 \longrightarrow c_5$ for some $c_5 \succeq c_1$. From $c_3 \preceq c_1$ and $c_1 \preceq c_5$ we know that $c_3 \preceq c_5$. From $c_2 \longrightarrow c_5$ and $c_3 \preceq c_5$ it follows that $c_2 \longrightarrow \widehat{c_3}$, i.e., $c_2 \in Pre\,(\widehat{c_3})$. By definition there is a $c_4 \in gen\,(Pre\,(\widehat{c_3}))$ with $c_4 \preceq c_2$. Since $c_4 \in gen\,(Pre\,(\widehat{c_3}))$ we know by definition that $c_3 \rightsquigarrow c_4$. ⊣

The following lemma follows immediately from the definition of $\rightsquigarrow$.

LEMMA 5.2. *If $c_1 \rightsquigarrow c_2$ then $c_2 \longrightarrow c_3$ for some $c_3 \succeq c_1$.*

LEMMA 5.3. *If $c_1 \longrightarrow c_3$ and $c_2 \preceq c_3$ then there is a c_4 such that $c_2 \rightsquigarrow c_4$ and $c_4 \preceq c_1$.*

PROOF. Suppose that $c_1 \longrightarrow c_3$ and $c_2 \preceq c_3$. This means that $c_1 \in Pre\,(\widehat{c_2})$. By definition there is a $c_4 \in gen\,(Pre\,(\widehat{c_2}))$ with $c_4 \preceq c_1$. Since $c_4 \in gen\,(Pre\,(\widehat{c_2}))$ we know by definition that $c_2 \rightsquigarrow c_4$. ⊣

For a configuration c, we define $Rank(c)$ to be the smallest n such that there is a sequence $c_0 \rightsquigarrow c_1 \rightsquigarrow \ldots \rightsquigarrow c_n$ where $c_0 = c$ and there is a $c' \in C_{init}$ such that $c_n \preceq c'$.

Now, we are ready to explain Algorithm 3. The algorithm maintains two sets of configurations: a set `ToExplore`, initialized to C_{fin}, of configurations that have not yet been analyzed; and a set `Explored`, initialized to the empty set, of configurations that contains information about the configurations that have already been analyzed. The algorithm preserves the following two invariants:

1. $C_{init} \xrightarrow{*} \widehat{(\texttt{ToExplore} \bigcup \texttt{Explored})}$ implies $C_{init} \xrightarrow{*} \widehat{C_{fin}}$; and
2. If $C_{init} \xrightarrow{*} \widehat{C_{fin}}$, then there is $c \in \texttt{ToExplore}$ such that both $Rank(c) < \infty$ and $\forall c' \in \texttt{Explored}.\ Rank(c) < Rank(c')$.

Initially, the first invariant holds since $(\texttt{ToExplore} \bigcup \texttt{Explored}) = C_{fin}$. The second invariant also holds initially as follows: Suppose that $C_{init} \xrightarrow{*} \widehat{C_{fin}}$, i.e., there is a $c \in C_{fin}$ and $c' \in C_{init}$ such that $c' \xrightarrow{*} \widehat{c}$. Then, the property $Rank(c) < \infty$ holds by Lemma 5.3 and the definition of the relation $\longrightarrow$.

Due to the invariants, the following two conditions can be checked during each step of the algorithm:

- From the second invariant, if ToExplore becomes empty then the algorithm terminates with a negative answer; and
- From the first invariant and the definition of $\longrightarrow$, if a configuration c is detected such that $c \preceq c'$, for some $c' \in C_{init}$, then the algorithm terminates with a positive answer.

If neither of the two conditions is satisfied, the algorithm proceeds by picking and removing a configuration c from ToExplore. Two possibilities arise depending on the value of c:

- If there exists a configuration $c' \in \texttt{Explored}$ with $c' \preceq c$, then we discard c. The first invariant is preserved since this operation will not change the value of $\widehat{(\texttt{ToExplore} \bigcup \texttt{Explored})}$. If $C_{init} \xrightarrow{*} C_{fin}$, then the second invariant and Lemma 5.1 imply that there is still some $c_1 \in \texttt{ToExplore}$ such that $Rank(c_1) < Rank(c') \leq Rank(c) \leq \infty$. This means that the second invariant will also be preserved by this step.
- Otherwise, we generate the successors of c with respect to $\rightsquigarrow$ put them in ToExplore, and move c to Explored. Let $\texttt{Explored}^{old}$ and $\texttt{Explored}^{new}$ be the values of the set Explored before resp. after performing the operation. Define $\texttt{ToExplore}^{old}$ and $\texttt{ToExplore}^{new}$ analogously. The operation preserves the first invariant as follows: Suppose that $C_{init} \xrightarrow{*} \widehat{(\texttt{ToExplore}^{new} \bigcup \texttt{Explored}^{new})}$, i.e., there are configurations c_1, c_2, c_3 such that such that $c_1 \in C_{init}$,

  $$c_2 \in (\texttt{ToExplore}^{new} \bigcup \texttt{Explored}^{new})$$

  $c_2 \preceq c_3$, and $c_1 \xrightarrow{*} c_3$. If

  $$c_2 \in (\texttt{ToExplore}^{old} \bigcup \texttt{Explored}^{old})$$

  then the result follows from the induction hypothesis. Otherwise, it must be the case that $c \rightsquigarrow c_2$ (since the only new members of $\texttt{ToExplore} \bigcup \texttt{Explored}$ are the $\rightsquigarrow$-successors of c). By Lemma 5.2 there is a c_4 such that $c \preceq c_4$ and $c_2 \longrightarrow c_4$. Since $c_2 \preceq c_3$ it follows by monotonicity that $c_3 \longrightarrow c_5$ for some $c_5 \succeq c_4$. From

$c \preceq c_4$ and $c_4 \preceq c_5$ we have $c \preceq c_5$. This means that $C_{init} \xrightarrow{*} \widehat{\left(\mathtt{ToExplore}^{old} \bigcup \mathtt{Explored}^{old}\right)}$, and hence by the induction hypothesis we have $C_{init} \xrightarrow{*} \widehat{C_{fin}}$. The operation also preserves the second invariant as follows: Assume that $C_{init} \xrightarrow{*} \widehat{C_{fin}}$. Since c does not satisfy the test in line 5 of the algorithm, it follows that $0 < Rank(c)$. If $0 < Rank(c) < \infty$, then there is some c_1 with $c \rightsquigarrow c_1$ and $Rank(c_1) < Rank(c)$; and the invariant will obviously be preserved. Suppose that $Rank(c) = \infty$, Since $C_{init} \xrightarrow{*} \widehat{C_{fin}}$ it follows by the induction hypothesis and the second invariant that there is a $c_1 \in \mathtt{ToExplore}^{old}$ such that $Rank(c_1) < \infty$ and $Rank(c_1) < Rank(c_2)$ for each $c_2 \in \mathtt{Explored}^{old}$. Since $c_1 \neq c$ it follows that $c_1 \in \mathtt{ToExplore}^{new}$ and hence the invariant still holds.

Furthermore, we remove all configurations in `Explored` which are larger than c with respect to $\preceq$. This operation preserves both invariants trivially.

The following theorem follows immediately from the invariants.

Theorem 5.4. *Algorithm* 3 *is partially correct.*

The reason why the algorithm always terminates is that only a finite set of configurations can be added to `Explored`. This can be explained as follows. Whenever a new element c is added to `Explored` it is ensured that $c' \not\preceq c$, for each c' already added to `Explored`. This means that the configurations added to `Explored` form a sequence $c_1, c_2, c_3, \ldots$, such that $c_i \not\preceq c_j$ for all $i < j$. By WQO of $\preceq$ it follows that this sequence is finite. This gives the following theorem.

Theorem 5.5. *Algorithm* 3 *is guaranteed to terminate.*

§6. Safety properties.

Sometimes, it is easier to describe safety properties by specifying the set of *allowed* (or *bad*) traces, rather than the set of bad configurations of the system. To formalize the idea, we first equip transition systems with *actions* (*lables*) which represent their interaction with the environment. Then, we recall the standard notion of finite automata which we use to specify sets of bad traces of the system. Checking a safety property is thus transformed to the reachability of accepting states of the finite automaton when composed with the transition system. The method will also explain why checking a safety property (almost always) translates to the reachability of an upward set of configurations.

6.1. Labeled transition systems. We fix a finite set A of *observable actions* which represent interactions between the transition system and its environment. We also assume a *silent action* ε, where $\varepsilon \notin A$, and define $A^\varepsilon := A \cup \{\varepsilon\}$. A *Labeled Transition System* (*LTS*) $\mathcal{T}$ is a tuple $(C, \longrightarrow, \preceq, C_{init})$ (i.e., of the same form as a transition system). The difference is that the relation

$\longrightarrow$ is indexed by the set of actions A^ε. Formally, $\longrightarrow\ = \left\{\xrightarrow{a} \mid a \in A^\varepsilon\right\}$, where $\xrightarrow{a}\ \subseteq C \times C$. We write $c_1 \xrightarrow{a} c_2$ to denote that $(c_1, c_2) \in \xrightarrow{a}$. A *trace* of $\mathcal{T}$ is a word $a_1 a_2 \ldots a_n \in A^*$ such that there is a sequence of transitions of the form $c_0 \xrightarrow{a_1} c_1 \xrightarrow{a_2} c_2 \cdots \xrightarrow{a_n} c_n$ where $c_0 \in C_{init}$.

The definition of WTS is extended in the obvious way from transition systems to LTS.

A class of safety properties can be described by giving regular sequences of observable actions which are allowed when the system executes. Formally, we are given an LTS $\mathcal{T}$, and a regular set Σ set over A, and want to check whether $Traces(\mathcal{T}) \subseteq \Sigma$.

In the example of Figure 1, we can take the set A to be $\{enter, exit\}$, and label each transition of the form (c_1, t_1, c_2) with *enter*, and each transition of the form (c_1, t_2, c_2) with *exit*. Intuitively, the action *enter* indicates that a process enters the critical section, while the action *exit* indicates that a process leaves the critical section. We can define the set Σ to be the regular language $enter \cdot (exit \cdot enter)^*$, i.e., it cannot happen that two processes enter their critical sections consecutively without a process leaving its critical section in between, and conversely it cannot happen that two processes leave their critical sections consecutively without a process entering its critical section in between.

6.2. Finite automata. We recall the standard definition of finite automata. A *finite automaton* $\mathcal{A}$ is a tuple $\left(Q, \delta, Q_{init}, Q_{fin}\right)$, where Q is a finite set of states, δ is the set of transitions, $Q_{init} \subseteq Q$ is the set of initial states, and $Q_{fin} \subseteq Q$ is the set if final states. Each transition is a triple of the form (s_1, a, s_2) where $s_1, s_2 \in S$ and $a \in A^\varepsilon$. The language $Lang(\mathcal{A})$ of $\mathcal{A}$ is defined as usual.

Given an LTS $\mathcal{T} = (C, \longrightarrow, \preceq, C_{init})$ and finite automaton $\mathcal{A} = \left(Q, \delta, Q_{init}, Q_{fin}\right)$, we define the composition $(\mathcal{T} \| \mathcal{A})$ to be an LTS $\mathcal{T}'$ in which $\mathcal{T}$ and $\mathcal{A}$ synchronize over transitions with actions in A. More precisely, the LTS $\mathcal{T}' := (C', \longrightarrow', \preceq', C'_{init})$, where

- $C' = \{(c, q) \mid (c \in C) \wedge (q \in Q)\}$.
- $(c_1, q_1) \xrightarrow{a}{}' (c_2, q_2)$ iff one of the following conditions is satisfied:
  - $a \neq \varepsilon$, $c_1 \xrightarrow{a} c_2$, and $(q_1, a, q_2) \in \delta$. This corresponds to transitions where $\mathcal{T}$ and $\mathcal{A}$ synchronize on actions in A.
  - $a = \varepsilon$, $c_1 \xrightarrow{\varepsilon} c_2$, and $q_1 = q_2$. This corresponds to transitions where $\mathcal{T}$ moves silently without synchronizing with $\mathcal{A}$.
  - $a = \varepsilon$, $c_1 = c_2$, and $(q_1, \varepsilon, q_2) \in \delta$. This is symmetric to the previous case.
- $(c_1, q_1) \preceq' (c_2, q_2)$ iff $c_1 \preceq c_2$ and $q_1 = q_2$.
- $(c, q) \in C'_{init}$ iff $c \in C_{init}$ and $q \in Q_{init}$.

It is straightforward to verify that if $\mathcal{T}$ is a WTS then $\mathcal{T}'$ is also a WTS.

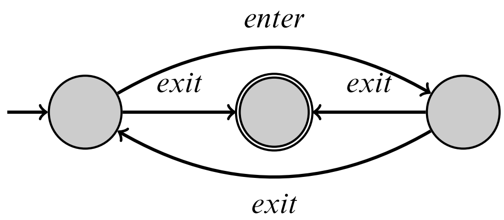

FIGURE 5. Bad traces of the example in Figure 1.

6.3. Algorithm. Algorithm 4 solves the problem when the set of allowed traces is regular (e.g., specified by a finite automaton). The algorithm needs one extra condition compared to Algorithms 2 and 3, namely that the set $gen(C)$ is given. This set is trivially known in the examples of this paper. For instance, in the case of Petri nets, the set $gen(C)$ is given by the singleton $\{c_0\}$ where $c_0(p) = 0$ for all places p.

ALGORITHM 4 Checking Safety Properties

Input: • $\mathcal{T} = (C, \longrightarrow, \preceq, C_{init})$: LTS.
• Σ: regular set of words over A.
Output: $Traces(\mathcal{T}) \subseteq \Sigma$?
1: construct $\mathcal{A}$ s.t. $Lang(\mathcal{A}) = \neg\Sigma$
2: $\mathcal{T}' \leftarrow (\mathcal{T}\|\mathcal{A}) = (C', \longrightarrow', \preceq', C'_{init})$
3: $C_{fin} \leftarrow \{(c, q) \mid c \in gen(C) \wedge q \in Q_{fin}\}$.
4: use Algorithm 3 to check whether $\widehat{C_{fin}}$ is reachable.

In Algorithm 4, we first construct a finite-state automaton automaton $\mathcal{A}$ which accepts the complement of Σ, and then form the product $(\mathcal{T}\|\mathcal{A})$. The problem of deciding whether $\mathcal{T}$ satisfies the safety property represented by Σ has now been transformed to the question whether a state of the product in which the $\mathcal{A}$-component is accepting is reachable. More precisely, violating the safety property is equivalent to the reachability of $\widehat{C_{fin}}$ where $C_{fin} = \{(c, q) \mid c \in gen(C) \wedge q \in Q_{fin}\}$. Furthermore, the set C_{fin} is finite since both $gen(C)$ and Q are finite. This explains why we can transform checking a safety property to the reachability of the upward closure of a finite set of configurations C_{fin}: we specify the bad traces by a finite automaton $\mathcal{A}$. Then, the members of C_{fin} correspond to those configuration in the composition where the $\mathcal{T}$-component is a member of the set $gen(C)$ and the $\mathcal{A}$-component is an accepting state in $\mathcal{A}$.

For the example of Figure 1, the automaton of Figure 5 describes the set of *bad* traces, i.e., the traces violating the safety property.

§7. **Lossy channel systems.** We introduce the model of lossy channel systems [4]. We give the LTS induced by a lossy channel system, and show that it is a WTS. We illustrate the model by a simple protocol.

7.1. Model. A *Lossy Channel System*, (*LCS* for short), consists of a finite-state process which operates on a finite set of channels. Each channel behaves as an unbounded FIFO queue which is unreliable in the sense it can lose messages. Typically, the control (finite-state) part models the total behavior of a number of processes which communicate over the channels. With each transition of the control part there may be associated an operation on the channels. This operation may remove a message from the head of a channel or insert a message at the end of a channel. In addition, a channel can nondeterministically lose messages at any time.

We fix a finite set $\mathtt{C}$ of channels, a finite set A of actions, and a finite set M of messages which may reside inside the channels. An LCS $\mathcal{L}$ is a tuple (S, T, s_{init}), where S is a finite set of *control states*, T is a finite set of *transitions*, and $s_{init} \in S$ is the *initial control state*. A transition t is a tuple (s_1, op, a, s_2), where $s_1, s_2 \in S$, $a \in A^\varepsilon$, and op is an operation of one of the following forms (where $\mathtt{c} \in \mathtt{C}$ and $m \in M$):

- $\mathtt{c}!m$ is a *send* operation. The operation appends m to the end of channel $\mathtt{c}$.
- $\mathtt{c}?m$ is a *receive* operation. The operation removes m from the head of channel $\mathtt{c}$ (it is enabled only if m is at the head of channel $\mathtt{c}$).
- *nop* is an empty operation which does not affect the contents of the channels.

For an action $a \in A^\varepsilon$, we define T_a to be the set of transitions of the form (s_1, op, a, s_2).

Below, we apply the methodology of Sections 4–6 to derive an algorithm which checks safety properties for LCS.

7.2. LTS. We define the LTS $\mathcal{T} = (C, \longrightarrow, \preceq, C_{init})$ induced by an LCS $\mathcal{L} = (S, T, s_{init})$. A configuration $c \in C$ is a pair (s, β) where $s \in S$ and β is a mapping from $\mathtt{C}$ to M^*. Intuitively, the state of the control part is given by s, while the *channel state* is given by β. For a channel $\mathtt{c}$, $\beta(\mathtt{c})$ gives the content of channel $\mathtt{c}$ (which is a word over M).

To define the transition relation $\longrightarrow$, we first give some definitions. For a channel state β, a channel $\mathtt{c}$, and a word w, we use $\beta[\mathtt{c} \leftarrow w]$ to be the channel state β' such that $\beta'(\mathtt{c}) = w$, and $\beta'(\mathtt{c}') = \beta(\mathtt{c}')$ if $\mathtt{c}' \neq \mathtt{c}$. For words $w_1, w_2 \in M^*$, we write $w_1 \preceq^* w_2$ to denote that w_1 is a (not unnecessarily contiguous) subword of w_2. For channel states β_1 and β_2, we write $\beta_1 \preceq^* \beta_2$ to denote that $\beta_1(\mathtt{c}) \preceq^* \beta_2(\mathtt{c})$ for all channels $\mathtt{c} \in \mathtt{C}$.

For an action $a \in A^\varepsilon$ and configurations $c_1 = (s_1, \beta_1)$, $c_2 = (s_2, \beta_2)$, we write $c_1 \xrightarrow{a} c_2$ to denote that there are channel states β_1', β_2' and a transition $(s_1, op, a, s_2) \in T$ such that the following conditions are satisfied:

1. $\beta_1' \preceq^* \beta_1$.
2. One of the following conditions is satisfied:
   - t is of the form $(s_1, \texttt{c}!m, s_2)$ and $\beta_2' = \beta_1'[\texttt{c} \leftarrow \beta_1'(\texttt{c}) \cdot m]$.
   - t is of the form $(s_1, \texttt{c}?m, s_2)$ and $\beta_1' = \beta_2'[\texttt{c} \leftarrow m \cdot \beta_2'(\texttt{c})]$.
   - t is of the form (s_1, nop, s_2) and $\beta_2' = \beta_1'$.
3. $\beta_2 \preceq^* \beta_2'$.

The system starts from control state s_1 and channel state β_1, and then performs a transition which consists of three steps. First, an arbitrary set of messages is lost obtaining a smaller channel state β_1', while preserving the control state s_1. Then, the system changes control state to s_2, and channel state to β_2' (the latter according to the operation op). Finally, a set of messages is lost again to obtain the channel state β_2. In other words, the actual transition is both preceded and followed by phases where the system may non-deterministically lose messages.

We define the ordering $\preceq$ on the set of configurations such that, for configurations $c_1 = (s_1, \beta_1)$ and $c_2 = (s_2, \beta_2)$, we have $c_1 \preceq c_2$ iff $s_1 = s_2$, and $\beta_1 \preceq^* \beta_2$.

The set C_{init} is the singleton $\{(s_{init}, \beta_{init})\}$ where $\beta_{init}(\texttt{c}) = \varepsilon$ for all channels $\texttt{c} \in \texttt{C}$. In other words, the system starts from a configuration where the control part is in its initial state, and where all the channels are empty.

7.3. WTS. First, we observe that $gen(C)$ is the (finite) set $\{(q, \beta_{init}) \mid q \in Q\}$. We show that the LTS obtained by an LCS is a WTS.

- The transition relation is monotonic since if $c_1 \preceq c_2$ then c_1 can first lose messages and transform into c_2. In this way, c_2 can perform (at least) the same transitions as c_1.
- From Higman's lemma [27] it follows that the pre-order $\preceq$ on configurations is a WQO (see more details in Section 10).
- The ordering $\preceq$ on configurations is decidable. Given two configurations $c_1 = (s_1, \beta_1)$ and $c_2 = (s_2, \beta_2)$, we can check $c_1 \preceq c_2$, by checking whether $s_1 = s_2$ and whether $\beta_1(\texttt{c})$ is a subword of $\beta_2(\texttt{c})$ for all channels $\texttt{c} \in \texttt{C}$.
- For configurations c_1 and c_2, the relation $c_1 \rightsquigarrow c_2$ holds if there is a transition $t = (s_1, op, a, s_2) \in T$ such that one of the following conditions holds:
  - $op = \texttt{c}!m$ and $\beta_1 = \beta_2[\texttt{c} \leftarrow \beta_2(\texttt{c}) \cdot m]$.
  - $op = \texttt{c}!m$, $last\left(\beta_1(\texttt{c})\right) \neq m$, and $\beta_1 = \beta_2$.
  - $op = \texttt{c}!m$, $\beta_1(\texttt{c}) = \varepsilon$, and $\beta_1 = \beta_2$.
  - $op = \texttt{c}?m$ and $\beta_2 = \beta_1[\texttt{c} \leftarrow m \cdot \beta_1(\texttt{c})]$.
  - $op = nop$ and $\beta_1 = \beta_2$.
- For a configuration $c = (s, \beta)$, $C_{init} \cap \widehat{c} = \emptyset$ amounts to $s = s_{init}$ and $\beta_{init} \preceq^* \beta$. Since $\beta_{init} \preceq^* \beta$ holds trivially, the test is equivalent to $s = s_{init}$.

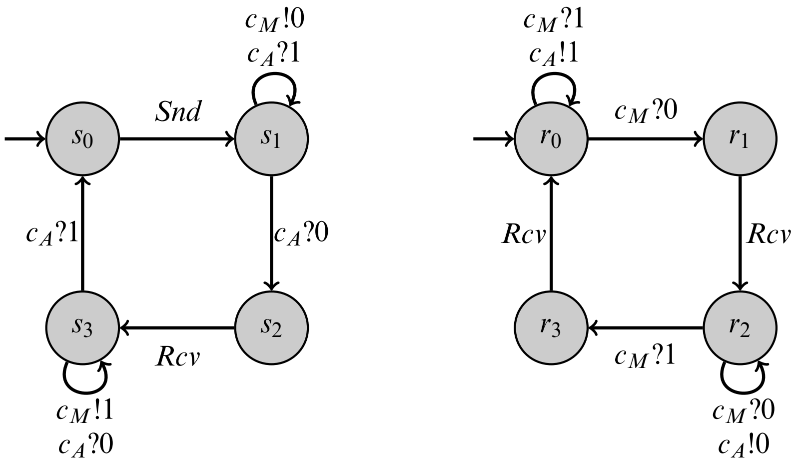

FIGURE 6. The alternating bit protocol as an LCS. The left and right parts of the figure represent the Sender and Receiver processes respectively.

7.4. Example: the Alternating Bit Protocol. In this section we model the classical Alternating Bit Protocol [13] as an LCS. The model is illustrated in Figure 6.

The alternating bit protocol contains a *Sender* and a *Receiver* that communicate over two FIFO channels c_M (used to transmit messages from the Sender to the Receiver) and c_A (used to transmit acknowledgments from the Receiver to the Sender). Both channels are faulty in the sense that they can lose (but not reorder) messages.

The purpose of the protocol is to transmit messages from the Sender to the Receiver in correct order, in spite of the fact that the channels can lose messages. Corruption of messages can also be taken into account by modeling it as loss (some mechanism will detect and discard a corrupted message). The operation of the protocol is the following:

The *Sender* reads a pending message to be sent to the Receiver (the action *Snd*). It adds a sequence number to the message, sends it over the channel c_M to the Receiver and awaits an acknowledgment from the Receiver with the same sequence number. If the message arrives, the procedure is repeated with the next pending message but with sequence numbers inverted. If no acknowledgment arrives within some time period the Sender retransmits the message. Retransmissions are repeated until a corresponding acknowledgment arrives.

The *Receiver* receives messages with accompanying sequence numbers from the channel c_M. When the message has the expected sequence number, the message is delivered (the action *Rcv*), and the Receiver looks for a message with inverted sequence number. Messages with non-expected sequence numbers are discarded. The Receiver sends acknowledgments to the Sender over the channel c_A. An acknowledgment contains the sequence number of the last received message.

In our example, we do not (need to) model the actual contents of the messages, and hence, a message is represented simply by its sequence number (which is either 0 or 1).

As mentioned earlier, the control part of an LCS may be used to represent the total behaviour of several processes. In our case, the control part (Figure 6) represents the Sender and the Receiver. To simplify the figure, we have omitted the empty channel operation *nop* and the empty action ε. For instance, the transition from s_0 to s_1 does not modify the channels, the transition from s_1 to s_2 performs the silent action ε, and so on. The protocol operates on the two channels c_M and c_A. The set A is $\{Snd, Rcv\}$, where *Snd* represents the sending of a message by the environment to the protocol, and *Rcv* represents the reception of a message by the environment from the protocol. The set M is $\{0, 1\}$. This means that the model of the Alternating Bit Protocol is the LCS $\mathcal{L} = (S, T, s_{init})$ where

- S is the set of pairs of the form (s, r), where $s \in \{s_0, s_1, s_2, s_3\}$ and $r \in \{r_0, r_1, r_2, r_3\}$.
- s_{init} is the state (s_0, r_0).
- A is the set $\{Snd, Rcv\}$.
- C is the set $\{c_M, c_A\}$.
- T consists of the tuples of the form $((s, r), op, a, (s', r'))$ where either $r = r'$ and (s, op, a, s') is a transition in the Sender component or $s = s'$ and (r, op, a, r') is a transition in the Receiver component. Examples of such transitions are $((s_0, r_0), nop, Snd, (s_1, r_0))$ and $((s_1, r_0), c_M?0, \varepsilon, (s_1, r_1))$.

We require the protocol to satisfy the following property: the environment cannot send two messages to the protocol without first receiving a message; and conversely the environment cannot receive two messages from the protocol without first sending a message. We apply the method of Sections 6 to verify this safety property. Figure 7 depicts the finite automaton $\mathcal{A}$ which specifies the set of bad traces, namely traces where two consecutive occurrences of *Snd* or *Rcv* may occur (or if *Rcv* occurs first). We apply the method of Sections 5- 6 to verify Algorithm 3 on the composition $\mathcal{T}' = (\mathcal{T} \| \mathcal{A}) = (C', \longrightarrow', \preceq', C'_{Init})$. Notice that a configuration in C' is of the form $(((s, r), \beta), q)$ where $s \in \{s_0, s_1, s_2, s_3\}$, $r \in \{r_0, r_1, r_2, r_3\}$, β is a mapping from $\{c_M, c_A\}$ to $\{0, 1\}$, and $q \in \{q_0, q_1, q_2\}$. Intuitively, the pair (s, r) is the state of the control part of $\mathcal{L}$ (given by the local states s and r of the sender and receiver respectively). The mapping β is the channel state of $\mathcal{L}$, and hence $((s, r), \beta)$ is a configuration of $\mathcal{L}$. Finally, q is the state of $\mathcal{A}$. To simplify the notation, we will write such a configuration simply as tuple (s, r, q, w_M, w_A), where $w_M = \beta(c_M)$ and $w_A = \beta(c_A)$. The set C_{fin} contains

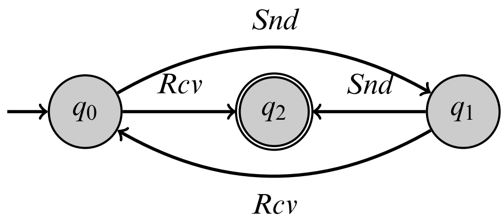

FIGURE 7. Bad traces of the alternating bit protocol.

all configurations of $\mathcal{T}'$ of the form $(s, r, q_2, \varepsilon, \varepsilon)$ where $s \in \{s_0, s_1, s_2, s_3\}$, $r \in \{r_0, r_1, r_2, r_3\}$.

Observe that $c_{init} = (s_0, r_0, q_0, \varepsilon, \varepsilon)$ is the only initial configuration in C'_{init}. When Algorithm 3 is applied to $\mathcal{T}'$ and C_{fin}, it answers that $\widehat{C_{fin}}$ is not reachable. When the algorithm terminates the set `Explored` contains the following configurations:

$$\begin{array}{lll}
(s_0, r_0, q_0, 0, \varepsilon) & (s_0, r_0, q_0, \varepsilon, 0) & (s_1, r_0, q_1, 01, \varepsilon) \\
(s_1, r_0, q_1, \varepsilon, 0) & (s_1, r_1, q_1, 1, \varepsilon) & (s_1, r_1, q_1, \varepsilon, 0) \\
(s_1, r_2, q_0, 1, \varepsilon) & (s_1, r_2, q_0, \varepsilon, 01) & (s_2, r_2, q_0, 1, \varepsilon) \\
(s_2, r_2, q_0, \varepsilon, 1) & (s_3, r_0, q_0, 0, \varepsilon) & (s_3, r_0, q_0, \varepsilon, 10) \\
(s_3, r_2, q_1, 10, \varepsilon) & (s_3, r_2, q_1, \varepsilon, 1) & (s_3, r_3, q_2, 0, \varepsilon) \\
(s_3, r_3, q_2, \varepsilon, 1) & &
\end{array}$$

The set contains also all configurations of the form $(s, r, q, \varepsilon, \varepsilon)$ where the triple (s, r, q) does not occur in the above list. Notice that there is no configuration $c \in$ `Explored` where $c \preceq c_{init}$ and hence $C_{init} \cap \widehat{\texttt{Explored}} = \emptyset$.

§8. Timed automata.

We recall the classical model of timed automata [12] and describe how it induces a WTS.

8.1. Model. A *Timed Automaton* (Figure 8) consists of a finite-state process which operates on a finite set of clocks. A clock assumes its values form the set of non-negative real numbers. Transitions of the automaton may check or reset values of the clocks.

We fix a finite set X of clocks. A timed automaton $\mathcal{T}$ is a tuple (S, T, s_{init}), where S is a finite set of *control states*, T is a finite set of *transitions*, and $s_{init} \in S$ is the *initial control state*. A transition t is a tuple (s_1, op, s_2), where $s_1, s_2 \in S$, and op is an operation of one of the following forms:

- $\mathtt{x} := 0$, where $\mathtt{x} \in \mathtt{X}$, resets the value of clock x to zero.
- $\mathtt{x} \sim k$, where $\mathtt{x} \in \mathtt{X}$, $\sim \in \{<, \leq, =, >, \geq\}$, and $k \in \mathbb{N}$. The transition tests the value of clock x, and is enabled only if the relation $\mathtt{x} \sim k$ holds.
- *nop* is the empty operation.

8.2. Transition system. We define the transition system $\mathcal{T} = (C, \longrightarrow, \preceq, C_{init})$ induced by a timed automaton $\mathcal{T} = (S, T, s_{init})$. A configuration

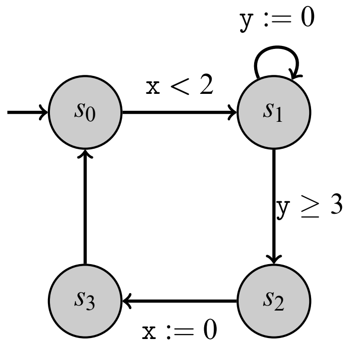

FIGURE 8. A timed automaton with two clocks x and y. The operations $\mathtt{x} := 0$ and $\mathtt{y} := 0$ reset the values of clocks x resp. y to zero. Transitions with no labels perform the empty operation.

$c \in C$ is a pair (s, β) where $s \in S$ and β is a mapping from $\mathtt{X}$ to $\mathbb{R}^{\geq 0}$. Intuitively, the state of the control part is given by s, while the clock values are given by β.

A timed automaton can perform two types of transitions, namely *timed transitions* and *discrete transitions*. First, we define the *timed transition relation* $\longrightarrow_{Timed}$. For a clock state β and a non-negative real number $\delta \in \mathbb{R}^{\geq 0}$, we define $\beta + \delta$ to be the clock state β' such that $\beta'(\mathtt{x}) = \beta(\mathtt{x}) + \delta$ for all $\mathtt{x} \in \mathtt{X}$. For a configuration $c = (s, \beta)$, we write $c + \delta$ to denote the configuration $(s, \beta + \delta)$. A timed transition is of the form $c \longrightarrow_{Timed} (c + \delta)$ where $\delta \in \mathbb{R}^{\geq 0}$. Intuitively, a timed transition corresponds to passage of time by an amount δ, and hence all clock values are increased by δ.

Next, we define the *discrete transition relation* $\longrightarrow_{Disc}$. For configurations $c_1 = (s_1, \beta_1)$ and $c_2 = (s_2, \beta_2)$, we write $c_1 \longrightarrow_{Disc} c_2$ to denote that there is a transition $t \in T$ of the form (s_1, op, s_2) such that one of the following conditions is satisfied:

- *op* is of the form $\mathtt{x} := 0$, $\beta_2(\mathtt{x}) = 0$, and $\beta_2(\mathtt{y}) = \beta_1(\mathtt{y})$ if $\mathtt{y} \neq \mathtt{x}$.
- *op* is of the form $\mathtt{x} \sim k$, $\beta_1(\mathtt{x}) \sim k$, and $\beta_1 = \beta_2$.
- *op* is of the form *nop* and $\beta_1 = \beta_2$.

We define $\longrightarrow \; := \; \longrightarrow_{Timed} \cup \longrightarrow_{Disc}$.

The ordering $\preceq$ in the case of timed automata turns out to be an equivalence relation $\equiv$, namely the classical *region equivalence* of [12]. More precisely, let *max* be the maximum integer which occurs syntactically in the definition of the timed automaton. For $x \in \mathbb{R}^{\geq 0}$, let $\mathit{fract}(x)$ and $\lfloor x \rfloor$ be the fractional and integral parts of r respectively. For configurations $c_1 = (s_1, \beta_1)$ and $c_2 = (s_2, \beta_2)$, we have $c_1 \equiv c_2$ iff the following properties hold for all clocks $\mathtt{x}, \mathtt{x}' \in \mathtt{X}$.

- $s_1 = s_2$.
- $\beta_1(\mathtt{x}) > max$ iff $\beta_2(\mathtt{x}) > max$.
- if $\beta_1(\mathtt{x}) \leq max$ then $\lfloor \beta_1(\mathtt{x}) \rfloor = \lfloor \beta_2(\mathtt{x}) \rfloor$.
- if $\beta_1(\mathtt{x}) \leq max$ and $\beta_1(\mathtt{x}') \leq max$ then the following property holds: $fract\,(\beta_1(\mathtt{x})) \leq fract\,(\beta_1(\mathtt{x}'))$ iff $fract\,(\beta_2(\mathtt{x})) \leq fract\,(\beta_2(\mathtt{x}'))$.

Each equivalence class of $\equiv$ is called a region. The set C_{init} is the singleton $\{(s_{init}, \beta^0)\}$ where $\beta^0(\mathtt{c}) = 0$ for all clocks $\mathtt{x} \in \mathtt{X}$. In other words, the system starts from a configuration where the control part is in its initial state, and where all clock values are equal to 0.

8.3. Finite partitioning. The region construction can be seen as an instance of the method of *finite partitioning* which works as follows. A *Finitely Partitioned Transition System* (*FPTS* for short) is a tuple $(C, \longrightarrow, \equiv, C_{init})$ where C is a set of configurations, $\longrightarrow\ \subseteq C \times C$ is a transition relation on C, $\equiv$ is an equivalence relation on C, and $C_{init} \subseteq C$ is the set of *initial configurations*. Furthermore, the following conditions are satisfied:

- $\equiv$ is a *congruence* wrt. $\longrightarrow$. In other words, for all configurations c_1, c_2, c_3, whenever $c_1 \equiv c_2$ and $c_1 \longrightarrow c_3$ then $c_2 \longrightarrow c_4$ for some $c_4 \equiv c_3$.
- $\equiv$ has a finite number of equivalence classes.

In [12] it is shown that $\equiv$ is indeed a congruence. Furthermore, the number of regions is finite. Hence, the transition system induced by a timed automaton is a FPTS.

Our method based on WTS is a generalization of finite partitioning. In fact, each FPTS is a WTS as follows:

- An equivalence relation is a pre-order which is symmetric.
- Each bisimulation is a simulation by definition.
- In case a pre-order $\preceq$ is equivalence relation, then each set which is upward closed wrt. $\preceq$ is an equivalence class. Conversely, each equivalence class is an upward closed set.
- In case a pre-order $\preceq$ is equivalence relation, the condition that $\preceq$ is a WQO is equivalent to the condition that the number of equivalence classes is finite.

§9. Symbolic analysis. In this section, we present a symbolic version of Algorithm 3 (Section 5). More precisely, we introduce the notion of a *constraint system* $\mathcal{C} = (C, \longrightarrow, \Psi, C_{init})$, where $C, \longrightarrow, C_{init}$ are of the same forms as in the case of transition systems (Section 4). Compared to transition systems, we replace the ordering $\preceq$ by a set of *constraints*. A constraint ϕ represents an infinite set $[\![\phi]\!]$ of configurations. The advantage of working with constraints is twofold. First, they sometimes remove unnecessary details in the definitions of configurations, which makes the design of the reachability algorithm more clear and easier to present. For instance, timed automata are usually analyzed using the classical notion of *regions* [12] as constraints. As explained in Section 8, a region is an equivalence class, and hence in our methodology a minimal element corresponds to a representative of the equivalence class to which it belongs. However, as we observed, only certain aspects of the clock values, such as the integral parts and the ordering of the fractional parts, are relevant in the analysis of timed automata (rather than the exact clock values). Therefore, all existing algorithms for analysis of timed automata use (variants of) regions as symbolic representations instead of using concrete configurations. The second (and more important) advantage offered by constraints is that each constraint may represent a (possibly large) set of minimal elements, and hence constraints may provide a more compact representation of infinite sets of configurations. Again, referring to the literature of timed automata, the constraint system of *zones* is used to represent infinite sets of configurations, since each zone may correspond to a large number of regions (minimal elements) and therefore zones provide a much more efficient representation than regions. In a similar manner to the case of transition systems (in Section 4), we will extract sufficient conditions which will enable us to present a (symbolic) algorithm operating on constraints. We will use the sufficient conditions to give a formal definition of the notion of a *well quasi-ordered constraint system*.

9.1. Symbolic algorithm. Consider a constraint system $\mathcal{C} = (C, \longrightarrow, \Psi, C_{init})$. We will work with a set of *constraints*, where each constraint ϕ denotes a set $[\![\phi]\!] \subseteq C$ of configurations. We write $c \models \phi$ to denote that $c \in [\![\phi]\!]$. For a (finite) set $\Phi \subseteq \Psi$ of constraints, we define $[\![\Phi]\!] := \bigcup_{\phi \in \Phi} [\![\phi]\!]$, i.e., Φ denotes the union of the denotations of its members. We say that Φ is *reachable* if the set $[\![\Phi]\!]$ is reachable. We define an *entailment* relation $\sqsubseteq$ on constraints such that $\phi_1 \sqsubseteq \phi_2$ if $[\![\phi_2]\!] \subseteq [\![\phi_1]\!]$, i.e., ϕ_1 is a weaker constraint than ϕ_2. For a constraint ϕ, we define $Pre(\phi)$ to be a finite set of constraints such that $[\![Pre(\phi)]\!] := \left\{ c \mid c \stackrel{*}{\longrightarrow} [\![\Phi]\!] \right\}$. Later, in the definition of a well quasi-ordered constraint system, we will assume that such a set always exists and that it is computable.

The symbolic algorithm (Algorithm 5) inputs a transition system $\mathcal{T}$, together with a finite set Φ_{fin} of constraints, and checks whether Φ_{fin} is reachable. The definition of Algorithm 5 is analogous to that of Algorithm 3; the difference being that we now use constraints rather than minimal elements as symbolic representations of sets of configurations.

9.2. Well quasi-ordered constraint systems. Considering Algorithm 5, we need $\mathcal{C}$ to satisfy the following conditions:

1. For each constraint ϕ, the set $Pre(\phi)$ is finite and computable. This is needed in line 10.
2. The relation $\sqsubseteq$ is decidable. This is needed in lines 7 and 11.
3. For each constraint ϕ, we can decide whether C_{init} intersects with $[\![\phi]\!]$. This is needed in line 5.

ALGORITHM 5 Symbolic (Constraint-Based) Backward Reachability

Input: • $\mathcal{C} = (C, \longrightarrow, \Psi, C_{init})$: transition system.
• Φ_{fin}: finite set of constraints.
Output: Is Φ_{fin} reachable?

```
1:  ToExplore ← Φ_fin
2:  Explored := ∅
3:  while ToExplore ≠ ∅ do
4:     remove some φ from ToExplore
5:     if C_init ∩ [[φ]] ≠ ∅ then
6:        return true
7:     else if ∃φ' ∈ Explored · φ' ⊑ φ then
8:        discard φ
9:     else
10:       ToExplore := ToExplore ⋃ Pre(φ)
11:       Explored := {φ} ⋃ {φ' | φ' ∈ Explored ∧ (φ ⋢ φ')}
12:    end if
13: end while
14: return false
```

4. The pre-order $\sqsubseteq$ is a WQO on the set of constraints. This is needed to guarantee termination of the algorithm.

We say that $\mathcal{C}$ is a *Well quasi-ordered Constraint System* (*WCS*) if it satisfies the above conditions.

9.3. WTS vs. WCS. Each transition system $\mathcal{T} = (C, \longrightarrow, \preceq, C_{init})$ induces a constraint system $\mathcal{C} = (C, \longrightarrow, \Psi, C_{init})$, where $\Psi = \{\phi_c \mid c \in C\}$ with $[\![\phi_c]\!] = \{c' \mid c \preceq c'\}$. In other words, a constraint ϕ_c characterizes an upward closed set, namely the upward closure $\widehat{c}$ of c. Thus the constraints play the role of minimal elements. In fact, if $\mathcal{T}$ is a WTS then $\mathcal{C}$ is a WCS as follows:

1. $Pre\,(\phi_c) = \{\phi_{c'} \mid c' \in (c \rightsquigarrow)\}$.
2. $\phi_{c_1} \sqsubseteq \phi_{c_2}$ iff $c_1 \preceq c_2$.
3. C_{init} intersects with $[\![\phi_c]\!]$ iff there is a $c' \in C_{init}$ such that $c \preceq c'$.
4. $\sqsubseteq$ is a WQO on the set of constraints since $\preceq$ is a WQO on C.

Given that $\mathcal{T}$ is a WTS, we can improve Algorithm 5 by weakening the definition of the relation $\sqsubseteq$ on constraints. We will now have $\phi_1 \sqsubseteq \phi_2$ if $[\![\phi_1]\!] \preceq_{\forall\exists} [\![\phi_2]\!]$. In other words, we do not require any longer that $[\![\phi_2]\!]$ is a subset of $[\![\phi_1]\!]$, only that for each configuration in $[\![\phi_2]\!]$ there is one smaller (wrt. $\preceq$) in $[\![\phi_1]\!]$. This means that the constraints which are generated in Algorithm 5 will not necessarily cover all the members of the set of configurations from which we can reach the bad states. However, they are guaranteed to contain at least the member of a generator of this set. To take this into consideration, we only need to change the condition of line 5 to

$\exists c \in [\![\phi]\!] \cdot \exists c' \in C_{init} \cdot c \preceq c'$. In the sequel, we refer to this version as the *improved symbolic reachability algorithm*.

§10. Building WQOs. In this section, we describe a methodology for hierarchically building more and more complicated domains which are WQOs.

A simple WQO is given by $(A, =)$, where A is a finite set and the ordering $=$ is the identity relation. Any infinite sequence $a_0, a_1, a_2, \dots \in A$ is good since, due to finiteness of A, there must exist i and j with $a_i = a_j$. Another simple example of WQOs is $(\mathbb{N}, \leq)$, i.e., the standard ordering on natural numbers.

The crucial step in building rich WQO domains is Higman's lemma [27] which can be explained as follows. Consider a WQO $(A, \preceq)$. We will extend the relation $\preceq$ to the the relation $\preceq^*$ on the set A^* of finite words over A. Consider words $w = a_0a_1 \dots a_m$ and $v = b_0b_1 \dots b_n$. We write $w \preceq^* v$ to denote that there is an injection h from $m^\bullet$ to $n^\bullet$ such that (i) h is strictly increasing, i.e., $i < j$ implies $h(i) < h(j)$; and (ii) $a_i \preceq b_{h(i)}$. In other words, w is a subword of v, albeit an element of w need not be identical to the corresponding one in v (it is sufficient that it is smaller wrt. $\preceq$). Then, Higman's lemma states that $(A^*, \preceq^*)$ is a WQO.

Below, we fix a WQO $(A, \preceq)$ which we extend in different ways. First, we extend the ordering to multisets over A. More precisely, we define $(A^\circledast, \preceq^\circledast)$ such that, for multisets $M = [a_1, \dots, a_m]$ and $N = [b_1, \dots, b_n]$ in $A^\circledast$, we have $M \preceq^\circledast N$ if there is an injection h from $m^\bullet$ to $n^\bullet$ with $a_i \preceq b_{h(i)}$. Notice that this ordering is a special case of that with words, where the relative ordering of the elements inside the multiset is not relevant (this is reflected by dropping the condition that the injection is increasing). Then, by Higman's lemma it follows that $(A^\circledast, \preceq^\circledast)$ is a WQO.

Using a similar reasoning, the ordering $\left(A^k, \preceq^k\right)$ where A^k is the set of vectors of length k over A, and where $(a_1, \dots, a_k) \preceq^k (b_1, \dots, b_k)$ iff $a_i \preceq b_i$ for all $i\colon 1 \leq i \leq k$, is a WQO. In fact, this is a special case of the word ordering, where the words are of identical length k.

We also conclude the WQO of $\left(2^A, \preceq^{\mathcal{P}}\right)$ where 2^A is the powerset of A, and where $\{a_1, \dots, a_m\} \preceq^{\mathcal{P}} \{b_1, \dots, b_n\}$ if there is an injection h from $m^\bullet$ to $n^\bullet$ such that $a_i \preceq b_{h(i)}$. This is a special case of multisets where each element occurs at most once inside the set.

Using the above methodology, we can build new more complicated WQOs. Below, we give some examples. Consider a finite set A; then the following are WQOs:

- $(A, =)$.
- $(A^\circledast, \preceq_1)$, where $\preceq_1$ is the result of extending $=$ to multisets over A as described above, i.e., $\preceq_1$ is $=^\circledast$.
- $(A^*, \preceq_2)$, where $\preceq_2$ extends $=$ to words, i.e., $\preceq_2$ is $=^*$. This means that the subword relation on words over a finite alphabet is a WQO.

- $\left((A^{\circledast})^*, \preceq_3\right)$, where $a_0a_1 \ldots a_m \preceq_3 b_0b_1 \ldots b_n$ if there is an injection h from $m^\bullet$ to $n^\bullet$ such that $a_i \preceq_2 b_{h(i)}$ for all $i\colon 1 \leq i \leq n$.
- $\left((A^*)^k, \preceq_4\right)$, where $(w_1, \ldots, w_k) \preceq_4 (v_1, \ldots, v_k)$ if $w_i \preceq_2 v_i$ for all $i\colon 1 \leq i \leq k$. In other words, vectors of finite words over a finite alphabet is WQO. Consequently, the ordering on channel contents for LCS in Section 7 is a WQO.
- $\left(A \times (A^*)^k, \preceq_5\right)$, where $(q_1, \beta_1) \preceq_5 (q_2, \beta_2)$ if $q_1 = q_2$ and $\beta_1 \preceq_4 \beta_2$. This means that the ordering on configurations of LCS in Section 7 is a WQO.
- $\left(A^{\circledast} \times (A^{\circledast})^* \times A^{\circledast}, \preceq_6\right)$, where $(M_1, w, M_2) \preceq_6 \left(M_1', w', M_2'\right)$ if $M_1 \preceq_2 M_1'$, $w \preceq_3 w'$, and $M_2 \preceq_2 M_2'$. We will use $\preceq_6$ for proving WQO of the entailment relation on constraints (regions) for Timed Petri Nets (see Section 11).

§11. Timed Petri Nets. In a Timed Petri Net, each token is equipped with a real-valued clock representing the "age" of the token. The firing conditions of a transition include the usual ones for Petri nets (Section 3). Furthermore, each arc between a place and a transition is labeled with a sub-interval of the natural numbers. When a transition is fired, the tokens removed from the input places of the transition and the tokens added to the output places should have ages lying in the intervals of the corresponding arcs.

We use a set *Intrv* of *intervals*. An open interval is written as (a, b) where $z \in \mathbb{N}$ and $b \in \mathbb{N} \cup \{\infty\}$. Intervals can also be closed in one or both directions, e.g., $[a, b)$ is closed to the left and open to the right. For $\alpha \in \mathbb{R}^{\geq 0}$, we write $\alpha \in [a, b)$ to denote that $a \leq \alpha < b$. The other relations $\alpha \in (a, b)$, $\alpha \in (a, b]$, and $[a, b]$ are defined analogously.

First, we introduce the model of Timed Petri nets, and then describe a constraint system for the model by defining the set of configurations, the transition relation, the set of constraints (regions), and the initial set of configurations. Then, we proceed to show that the constraint system is WQO by showing how to compute the *Pre* operation, how to check the entailment relation, how to check intersection with initial configurations, and finally showing that the entailment relation on regions is a WQO.

11.1. Model. A *Timed Petri Net* (TPN) is a tuple $\mathcal{N} = (P, T, F)$ where P is a finite set of *places*, T is a finite set of *transitions*, and the *flow relation* F is a partial mapping from the set $(P \times T) \cup (T \times P)$ to the set *Intrv*. We define $In(t) := \{(p, \mathcal{I}) \mid F(p, t) = \mathcal{I}\}$ and $Out(t) := \{(p, \mathcal{I}) \mid F(t, p) = \mathcal{I}\}$.

11.2. Configurations. A configuration (marking) c is a finite multiset over $P \times \mathbb{R}^{\geq 0}$. The configuration c defines the numbers and ages of the tokens in each place in the net. That is, $c(p, \alpha)$ defines the number of tokens of age α in place p. For instance, the configuration of Figure 9

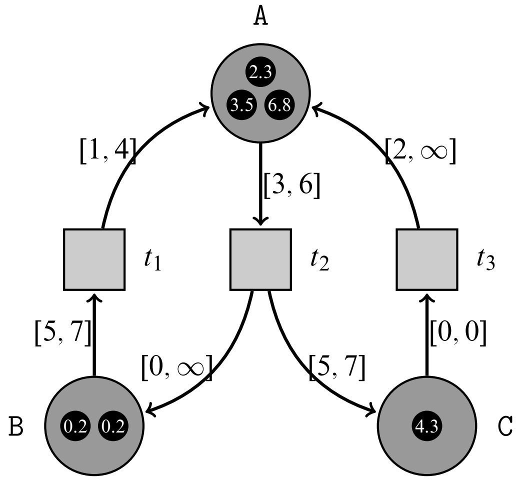

FIGURE 9. A simple TPN.

is $\left[(\mathtt{A}, 2.3), (\mathtt{A}, 3.5), (\mathtt{A}, 6.8), (\mathtt{B}, 0.2)^2, (\mathtt{C}, 4.3)\right]$. Abusing notation, we define, for each place p, a multiset $c(p)$ over $\mathbb{R}^{\geq 0}$, where $c(p)(\alpha) = c(p, \alpha)$. Notice that untimed Petri nets (Section 3) are a special case of TPNs where all intervals are of the form $[0, \infty)$. The set of initial configurations is given by identifying an initial place $p_{init} \in P$ and defining C_{init} to be the singleton $\{[(p_{init}, 0)]\}$. In other words, the initial configuration contains a single token with age zero in the initial place. This definition is not as restrictive as it might seem. Given a TPN $\mathcal{N}$ and a more general sets of initial configurations C_{init}, we can construct a new TPN $\mathcal{N}'$. The TPN $\mathcal{N}'$ first runs an initial phase where it start from $[(p_{init}, 0)]$ and then nondeterministically generates a member of C_{init}. Once this has been done, $\mathcal{N}'$ switches to the next phase, where it simulates $\mathcal{N}$.

11.3. Transition relation. In a similar manner to timed automata (Section 8), we define two types of transition relations on configurations. A *timed transition* increases the ages of all tokens by the same real number. For a configuration $c_1 = [(p_1, \alpha_1), \ldots, (p_n, \alpha_n)]$ and a number $\delta \in \mathbb{R}^{\geq 0}$, we use $c + \delta$ to denote the configuration $[(p_1, \alpha_1 + \delta), \ldots, (p_n, \alpha_n + \delta)]$. We use $c_1 \longrightarrow_{Timed} c_2$ to denote that $c_2 = c_1 + \delta$ for some $\delta \in \mathbb{R}^{\geq 0}$.

Now, we define the *discrete transition relation* $\longrightarrow_{Disc}$. For configurations c_1 and c_2, we write $c_1 \longrightarrow_{Disc} c_2$ to denote that there is a transition $t \in T$ with $In(t) = [(p_1, \mathcal{I}_1), \ldots, (p_k, \mathcal{I}_k)]$ and $Out(t) = [(q_1, \mathcal{J}_1), \ldots, (q_\ell, \mathcal{J}_\ell)]$, and multisets $M_1 = [(p_1, \alpha_1), \ldots, (p_k, \alpha_k)]$ and $M_2 = \left[(q_1, \alpha_1'), \ldots, (q_\ell, \alpha_\ell')\right]$ such that

- $\alpha_i \in \mathcal{I}_i$ for all $i\colon 1 \leq i \leq k$.
- $M_1 \leq c_1$.

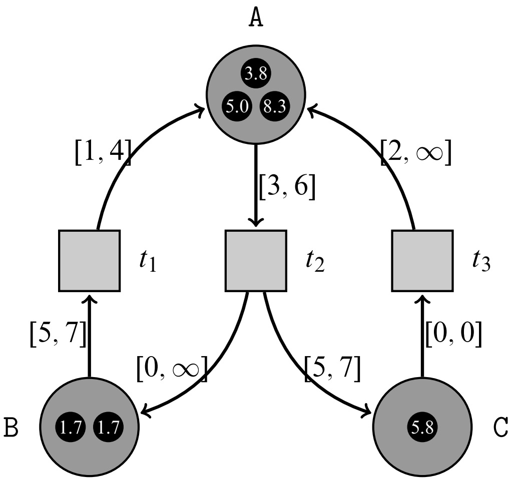

FIGURE 10. Performing a timed transition where time passes by an amount of 1.5 from the configuration of Figure 9.

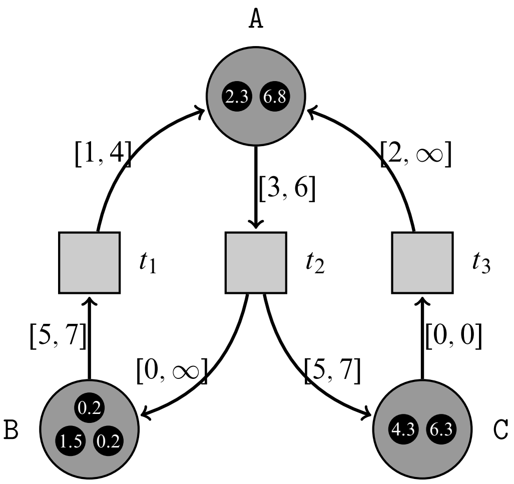

FIGURE 11. A possible result of firing transition t_2 from the configuration of Figure 9.

- $\alpha'_i \in \mathcal{J}_i$ for all $i : 1 \leq i \leq \ell$.
- $c_2 = (c_1 - M_1) + M_2$.

We say that t is *enabled* at c_1 if the first two conditions are satisfied.

11.4. Regions. We define a set of constraints called *regions*. These are extensions of the constraints with the same name we introduced for timed automata (see Section 8). The main difference is that in the case of TPNs, the number of clocks (ages of tokens) is not bounded in general (in contrast to timed automata where the set of clocks is a priori given). This implies that there is a finite number of regions for a given timed automaton, while this is not the case for a TPN.

Let *max* be the maximum integer which occurs syntactically in the definition of the TPN. We will distinguish between three types of tokens depending on their clock values (ages). In order to do this, we define three sets of symbols, namely $Sym^{max} := \{max^+\}$, $Sym^0 := \{k^0 \mid k \in \mathbb{N}, 0 \le k \le max\}$, and $Sym^+ := \{k^+ \mid k \in \mathbb{N}, 0 \le k < max\}$. We use the (singleton) set Sym^{max} to represent tokens whose ages are larger than *max*. The other two sets are used to represent tokens whose ages are smaller than (or equal to) *max*, where tokens represented by Sym^0 resp. Sym^+ have zero resp. positive fractional parts. We define $Sym := Sym^0 \cup Sym^+ \cup Sym^{max}$. We define an ordering $<$ on Sym such that $k^0 < k^+ < (k+1)^0 < max^+$ for all $k : 0 \le k < max$. We define the *signature* $sig(\alpha)$ of a real number α (where α represents the age of a token) as follows:

- If $\alpha > max$ then $sig(\alpha) := max^+$. The actual ages of these tokens are irrelevant, and hence the information about their ages is omitted in the representation. (This is because the transitions in the net cannot distinguish between different ages of tokens if these are strictly larger than *max*.)
- If $\alpha \le max$ and $fract(\alpha) = 0$ then $sig(\alpha) := \alpha^0$.
- If $\alpha \le max$ and $fract(\alpha) > 0$ then $sig(\alpha) := \lfloor \alpha \rfloor^+$. If the fractional part is positive then the age is approximated to $\lfloor \alpha \rfloor$.

For a token (p, α), we define

$$sig(p, \alpha) := (p, sig(\alpha));$$

and for a configuration $c = [(p_1, \alpha_1), \ldots, (p_n, \alpha_n)]$, we define

$$sig(c) := [sig(p_1, \alpha_1), \ldots, sig(p_n, \alpha_n)].$$

A *region* stores tokens by their signatures, and reflects the ordering of the fractional parts. For $i \in \{max, 0, +\}$, we let

$$P^i := \{(p, sym) \mid p \in P, sym \in Sym^i\}.$$

A *region* R is a triple $(Zero, Pos, Max)$ where

- $Zero \in (P^0)^{\circledast}$ is a multiset of pairs that represent tokens with zero fractional parts and values which are at most *max*. A pair of the form (p, k) represents a token of age exactly k in place p.
- $Pos \in \left((P^+)^{\circledast} - \{\emptyset\}\right)^*$. Each element in the word Pos is a non-empty multiset over P^+. The word Pos represents tokens with positive fractional parts. A pair (p, k^+) represents a token in place p with age $\alpha \le max$ such that $\lfloor \alpha \rfloor = k$. Pairs in the same multiset represent tokens whose ages have identical fractional parts. The order of the

multisets in *Pos* corresponds to the order of the fractional parts (i.e., smaller fractional parts come first in the word *Pos*).

- $Max \in (P^{max})^{\circledast}$ is a multiset over P^{max} representing tokens with ages strictly larger than *max*.

The semantics of a region $(Zero, Pos, Max)$ would not change if we allowed empty multisets to appear in *Pos*. However, we forbid this in order to obtain a unique representation. We call *Zero*, *Pos*, and *Max* the *zero*, *positive*, resp. *max* parts of R.

Consider a configuration c and a region $R = (M_0, M_1 \dots M_n, M_{n+1})$, i.e., $Zero = M_0$, $Pos = M_1 \dots M_n$, and $Max = M_{n+1}$. We use $c \models R$ to denote that there are configurations $c_0, \dots, c_{n+1}$ such that the following conditions are satisfied:

- $c \geq c_0 + c_1 + \cdots + c_n + c_{n+1}$.
- $M_i \leq sig(c_i)$ for all $i : 0 \leq i \leq n + 1$.
- If $(p_1, \alpha_1) \in M_i$ and $(p_2, \alpha_2) \in M_j$ with $1 \leq i, j \leq n$ then $fract(\alpha_1) < fract(\alpha_2)$ iff $i < j$. This condition implies $fract(\alpha_1) = fract(\alpha_2)$ iff $i = j$. Thus, tokens with identical fractional parts correspond to elements in the same multiset (unless they belong to M_{n+1}). Furthermore, the ordering among the multisets inside *Pos* reflects the ordering among the fractional parts of the clock values (increasing from left to right).

REMARKS. The region R defines a set of minimal requirements on c. More precisely, c should contain at least $\ell = \sum_{0 \leq i \leq n+1} |M_i|$ tokens. The places and ages of these tokens are constrained as described above where R specifies the integral parts of token ages and an ordering on their fractional parts (up to ages equal to *max*). A configuration c which satisfies R should have at least the ℓ tokens specified by R. In such a case, c may have any number of additional tokens (whose places and ages are then irrelevant for the satisfiability of the region by the configuration).

We notice that a configuration c defines a unique maximal (wrt. entailment on regions) region $Reg(c) = (M_0, M_1 \dots M_n, M_{n+1})$ such that $c \models R$; namely the region where there are configurations $c_0, \dots, c_{n+1}$ satisfying the following conditions:

- $c = c_0 + \cdots + c_{n+1}$.
- $M_i = sig(c_i)$ for all $i : 0 \leq i \leq n + 1$.
- If $(p_1, \alpha_1) \in M_i$ and $(p_2, \alpha_2) \in M_j$ with $1 \leq i, j \leq n$ then $fract(\alpha_1) < fract(\alpha_2)$ iff $i < j$.

The set of regions induces a natural ordering $\preceq$ on the set of configurations, where $c_1 \preceq c_2$ iff $c_1 \models R$ implies $c_2 \models R$ for all regions R. Let us define an equivalence relation $\equiv_{Reg}$ on configurations such that $c_1 \equiv_{Reg} c_2$ iff $Reg(c_1) = Reg(c_2)$. In fact, $c_1 \preceq c_2$ iff there are c_2', c_2'' such that $c_2 = c_2' + c_2''$ and $c_1 \equiv_{Reg} c_2'$. Equivalently, $c_1 \preceq c_2$ iff $Reg(c_1) \sqsubseteq Reg(c_2)$. It can be verified that $(C, \longrightarrow, \preceq, C_{init})$ where C is the set of configurations of the

TPN, C_{init} is the set of initial markings, $\longrightarrow$ is the transition relation, and $\preceq$ is the above ordering, is a WTS.

EXAMPLES. Assume that $max = 7$. Consider a configuration

$$c = \big[(p_1, 3.2), (p_1, 1.0), (p_1, 5.7), (p_1, 1.8), (p_2, 4.2), (p_2, 9.2), (p_3, 4.0), (p_3, 7.2)\big].$$

Then

$$Reg\,(c) = \left(\begin{bmatrix} (p_1, 1^0) \\ , \\ (p_3, 4^0) \end{bmatrix}, \begin{bmatrix} (p_1, 3^+) \\ , \\ (p_2, 4^+) \end{bmatrix} \big[(p_1, 5^+)\big] \big[(p_1, 1^+)\big], \begin{bmatrix} (p_2, 7^+) \\ , \\ (p_3, 7^+) \end{bmatrix}\right).$$

Consider the regions

$$R_1 = \left(\begin{bmatrix} (p_1, 1^0) \\ , \\ (p_3, 4^0) \end{bmatrix}, \big[(p_2, 4^+)\big] \big[(p_1, 5^+)\big] \big[(p_1, 1^+)\big], \begin{bmatrix} (p_2, 7^+) \\ , \\ (p_3, 7^+) \end{bmatrix}\right),$$

$$R_2 = \left(\begin{bmatrix} (p_1, 1^0) \\ , \\ (p_3, 4^0) \end{bmatrix}, \big[(p_3, 2^+)\big] \begin{bmatrix} (p_1, 3^+) \\ , \\ (p_2, 4^+) \end{bmatrix} \big[(p_1, 5^+)\big] \big[(p_1, 1^+)\big], \begin{bmatrix} (p_2, 7^+) \\ , \\ (p_3, 7^+) \end{bmatrix}\right),$$

$$R_3 = \left(\begin{bmatrix} (p_1, 1^0) \\ , \\ (p_3, 4^0) \end{bmatrix}, \begin{bmatrix} (p_1, 3^+) \\ , \\ (p_2, 4^+) \\ , \\ (p_3, 2^+) \end{bmatrix} \big[(p_1, 5^+)\big] \big[(p_1, 1^+)\big], \begin{bmatrix} (p_2, 7^+) \\ , \\ (p_3, 7^+) \end{bmatrix}\right).$$

Then $c \models R_1$, $c \not\models R_2$, and $c \not\models R_3$.

11.5. Computing predecessors. We define $Pre := Pre_{Timed} \cup Pre_{Disc}$, where Pre_{Timed} corresponds to running time backwards and Pre_{Disc} corresponds to firing transitions backwards

To define Pre_{Timed}, we introduce an operation *Rotate* on regions which simulates the effect of running time backwards from a region. For a region $R = (Zero, Pos, Max)$, we use $Rotate(R)$ to denote the set of regions of the form $(Zero', Pos', Max')$, such that one of the following conditions is satisfied:

- *Zero* is empty, and there are multisets
  - $M_1 = \big[(p_1, k_1^+), \ldots, (p_m, k_m^+)\big]$.
  - $M_1' = \big[(p_1, k_1^0), \ldots, (p_m, k_m^0)\big]$.
  - $M_2 = \big[(q_1, max^+), \ldots, (q_n, max^+)\big]$.

– $M_2' = [(q_1, max^0), \ldots, (q_n, max^0)]$.

such that the following conditions are satisfied

– $Pos = M_1 \cdot Pos'$ if $M_1 \neq \emptyset$ and $Pos = Pos'$ otherwise.

– $Max = Max' + M_2$.

– $Zero' = M_1' + M_2' \neq \emptyset$.

A configuration satisfying R does not contain any tokens whose ages have zero fractional parts ($Zero$ is empty). The first "interesting thing" to happen when running time backwards is that the fractional parts of some token ages become equal to zero. All such ages have identical fractional parts (say equal to r), i.e. their ages are of the form $k + r$, where $k \in \mathbb{N}$. Notice that all these tokens have signatures of the form (p, k^+) where $0 \leq k \leq max$. There are three types of such tokens depending on whether $k < max$, $k = max$, or $k > max$. Tokens of the first two types are represented by the multisets M_1 and M_2 respectively. These tokens will be transformed into tokens whose ages have zero fractional parts; hence they will have signatures of the form (p, k^0) (represented by the multisets M_1' and M_2'). The two multisets M_1' and M_2' represent all tokens with zero fractional parts in R' and hence $Zero' = M_1' + M_2'$. The tokens in M_1 are those with the smallest fractional parts in Pos and therefore M_1 is the first multiset in Pos (Pos is of the form $M_1 \cdot Pos'$). If M_1 is empty, then this indicates that there are not tokens of the first type (with ages of the form $k + r$, $k < max$). The third types of tokens will still have ages which are larger than max and hence their signatures will remain unchanged (these tokens will remain part of Max).

- $Zero$ is not empty, $Zero = [(p_1, k_1^0), \ldots, (p_m, k_m^0)]$, $0 < k_i \leq max$ for all $i: 1 \leq i \leq m$, $Zero'$ is empty, $Max' = Max$, and $Pos' = Pos \cdot Zero''$ where $Zero'' = \left[\left(p_1, (k_1 - 1)^+\right), \ldots, \left(p_m, (k_m - 1)^+\right)\right]$. A configuration satisfying R contains some tokens whose ages have zero fractional parts ($Zero$ is not empty). The age of such a token is some natural number $0 < k_i \leq max$. The first "interesting thing" to happen when running tie backwards is that the integral parts of ages of these tokens will be reduced by one. Also, the fractional parts become positive (and in fact larger than the fractional parts of any other tokens). Therefore, the signature of such a token will be of the form $\left(p_i, (k_i - 1)^+\right)$. Since these tokens have the highest fractional parts, the multiset representing them (i.e., $Zero''$) will be put last in Pos'. Notice that we require that no token in $Zero$ should have a zero integral part (otherwise the age of the token would be equal to zero and hence time cannot run backwards), and that no token whose age have a zero fractional part will remain in the region ($Zero'$ is empty). The tokens whose ages are larger than max (represented by Max) will not be affected since running time backwards "by a small amount" will keep their values larger than *max*.

Figure 12 shows some examples of applications of the rotation operation. We define Pre_{Timed} to be the reflexive transitive closure of *Rotate*, i.e. $R' \in Pre_{Timed}(R)$ iff there are regions $R_0, R_1, \ldots, R_n$ such that $R_0 = R$, $R_n = R'$, and $R_{i+1} \in Rotate(R_i)$ for all $i : 0 \leq i < n$.

Now, we turn our attention to computing Pre_{Disc}. We define $Pre_{Disc} := \bigcup_{t \in T} Pre_t$, where Pre_t describes the effect of running the transition t backwards. We start by describing a number of operations on regions. Consider a word $w \in \left((P^+)^{\circledast} - \{\emptyset\}\right)^*$, and a a pair $(p, k^+) \in P^+$. We define $w \ominus (p, k^+)$ to be the set of words w' satisfying the following property:

- $w = w_1 \cdot M \cdot w_2$, $w' = w_1 \cdot M' \cdot w_2$, $(p, k^+) \in M$, and $M' = M - \left[(p, k^+)\right]$.

We will use this operation to remove (the signature of) a token from the (word w corresponding to the) positive part of a region. The token may be removed from any multiset inside the word provided that the token occurs in the multiset.

We define $w \oplus (p, k^+)$ to be the set of words satisfying one of the following two properties:

- $w = w_1 \cdot M \cdot w_2$, $w' = w_1 \cdot M' \cdot w_2$, and $M' = M + \left[(p, k^+)\right]$.
- $w = w_1 \cdot w_2$, and $w' = w_1 \cdot \left[(p, k^+)\right] \cdot w_2$.

We will use this operation to add a token to the positive part of a region. The token can either be added to multiset in w (first case), or we create a new multiset containing the token (second case).

For an interval $\mathcal{I} = [a, b]$, and $\ell \in Sym$, we write $\ell \in \mathcal{I}$ to denote that $a^0 \leq \ell \leq b^0$ (recall the ordering we have defined on the members of *Sym*). The relations $\ell \in [a, b)$, $\ell \in (a, b]$, and $\ell \in (a, b)$ are defined analogously.

For a region $R = (Zero, Pos, Max)$ and a pair $(p, \ell) \in P \times Sym$ we define $R \ominus (p, \ell)$ to be the set of regions R' satisfying one the following conditions:

- $\ell \in Sym^0$, $(p, \ell) \in Zero$, $R' = (Zero', Pos, Max)$, and $Zero' = Zero - [(p, \ell)]$.
- $\ell \in Sym^+$, $R' = (Zero, Pos', Max)$, and $Pos' \in (Pos \ominus (p, \ell))$.
- $\ell \in Sym^{max}$, $(p, \ell) \in Max$, $R' = (Zero, Pos, Max')$, and $Max' = Max - [(p, \ell)]$.

The operation is used to remove a token from a region. Depending on the signature of the token, it is removed either from the zero, positive, or max part of the region.

We define $R \oplus (p, \ell)$ to be the set of regions R' satisfying one the following conditions:

- $\ell \in Sym^0$, $R' = (Zero', Pos, Max)$, and $Zero' = Zero + [(p, \ell)]$.
- $\ell \in Sym^+$, $R' = (Zero, Pos', Max)$, and $Pos' \in (Pos \oplus (p, \ell))$.
- $\ell \in Sym^{max}$, $R' = (Zero, Pos, Max')$, and $Max' = Max + [(p, \ell)]$.

R_0:	$(p,4^0)$ $(q,2^0)$	$(p,5^+)$	$(p,3^+)$ $(p,3^+)$	$(p,7^+)$ $(q,7^+)$	
R_1:		$(p,5^+)$	$(p,3^+)$ $(p,3^+)$	$(p,3^+)$ $(q,1^+)$	$(p,7^+)$ $(q,7^+)$
R_2:	$(p,7^0)$ $(p,5^0)$	$(p,3^+)$ $(p,3^+)$	$(p,3^+)$ $(q,1^+)$	$(q,7^+)$	
R_3:		$(p,3^+)$ $(p,3^+)$	$(p,3^+)$ $(q,1^+)$	$(p,6^+)$ $(p,4^+)$	$(q,7^+)$
R_4:	$(p,3^0)$ $(p,3^0)$	$(p,3^+)$ $(q,1^+)$	$(p,6^+)$ $(p,4^+)$	$(q,7^+)$	
R_5:		$(p,3^+)$ $(q,1^+)$	$(p,6^+)$ $(p,4^+)$	$(p,2^+)$ $(p,2^+)$	$(q,7^+)$
R_6:	$(p,3^0)$ $(q,1^0)$	$(p,6^+)$ $(p,4^+)$	$(p,2^+)$ $(p,2^+)$	$(q,7^+)$	
R_7:		$(p,6^+)$ $(p,4^+)$	$(p,2^+)$ $(p,2^+)$	$(p,2^+)$ $(q,0^+)$	$(q,7^+)$
R_8:	$(p,6^0)$ $(p,4^0)$	$(p,2^+)$ $(p,2^+)$	$(p,2^+)$ $(q,0^+)$	$(q,7^+)$	
R_9:		$(p,2^+)$ $(p,2^+)$	$(p,2^+)$ $(q,0^+)$	$(p,5^+)$ $(p,3^+)$	$(q,7^+)$
R_{10}:	$(p,2^0)$ $(p,2^0)$	$(p,2^+)$ $(q,0^+)$	$(p,5^+)$ $(p,3^+)$	$(q,7^+)$	
R_{11}:		$(p,2^+)$ $(q,0^+)$	$(p,5^+)$ $(p,3^+)$	$(p,1^+)$ $(p,1^+)$	$(q,7^+)$
R_{12}:	$(p,2^0)$ $(q,0^0)$	$(p,5^+)$ $(p,3^+)$	$(p,1^+)$ $(p,1^+)$	$(q,7^+)$	

FIGURE 12. A sequence of rotations: $R_{i+1} \in Rotate(R_i)$. Notice that $R_i \in Pre_{Timed}(R_0)$ for all $i : 0 \le i \le 12$.

The operation can be explained in a similar manner to the previous one.

For a region R and a multiset $M = [(p_1, k_1), \ldots, (p_m, k_m)] \in (P \times Sym)^{\circledast}$, we define $R \ominus M$ to be the set of regions R' such that there are regions $R_0, R_1, \ldots, R_{m+1}$ where $R_0 = R$, $R_{n+1} = R'$, and $R_{i+1} \in R_i \ominus (p, k_i)$ for all $i : 1 \leq i \leq m$. We define $R \oplus M$ analogously. These two operations are used to remove (add) a multiset of tokens from (to) a region.

Consider a transition $t \in T$. Let $In(t) = [(p_1, \mathcal{I}_1), \ldots, (p_k, \mathcal{I}_m)]$ and $Out(t) = [(q_1, \mathcal{J}_1), \ldots, (q_\ell, \mathcal{J}_n)]$. For a region R, we define $Pre_t(R)$ to be the set of regions R' satisfying the following condition: there are multisets $M_1 = [(p_1, k_1), \ldots, (p_m, k_m)] \in (P \times Sym)^{\circledast}$ and $M_2 = \left[(q_{i_1}, \ell_{i_1}), \ldots, (q_{i_r}, \ell_{i_r})\right] \in (P \times Sym)^{\circledast}$ such that

- $1 \leq i_1 < i_2 < \cdots < i_r \leq n$.
- $k_j \in \mathcal{I}_j$ for all $i : 1 \leq j \leq m$.
- $\ell_{i_j} \in \mathcal{J}_{i_j}$ for all $j : 1 \leq j \leq r$.
- $R' \in (R'' \oplus M_1)$ for some $R'' \in (R \ominus M_2)$.

We choose a subset of the output places of t (described by the sequence $i_1, i_2, \ldots, i_r$). Since we are running t backwards, we remove tokens corresponding to all these output places (these tokens were generated through the firing of t). For each input place we add a token (these tokens were removed through the firing of t). The tokens added and removed should have the correct ages (signatures), i.e., they should belong to the relevant intervals.

EXAMPLE. Assume that $max = 7$. Consider a region

$$R = \left(\begin{bmatrix} (p_1, 1^0) \\ , \\ (p_3, 4^0) \end{bmatrix}, \begin{bmatrix} (p_1, 3^+) \\ , \\ (p_2, 4^+) \end{bmatrix} \left[(p_1, 5^+)\right] \left[(p_1, 1^+)\right], \begin{bmatrix} (p_2, 7^+) \\ , \\ (p_3, 7^+) \end{bmatrix} \right).$$

The set $R \ominus (p_1, 4^+)$ is the singleton $\{R_1\}$ where

$$R_1 = \left(\begin{bmatrix} (p_1, 1^0) \\ , \\ (p_3, 4^0) \end{bmatrix}, \left[(p_1, 3^+)\right] \left[(p_1, 5^+)\right] \left[(p_1, 1^+)\right], \begin{bmatrix} (p_2, 7^+) \\ , \\ (p_3, 7^+) \end{bmatrix} \right).$$

Examples of regions in $R \oplus (p_1, 4^+)$ are the regions R_2 and R_3 defined as:

$$R_2 = \left(\begin{bmatrix} (p_1, 1^0) \\ , \\ (p_3, 4^0) \end{bmatrix}, \begin{bmatrix} (p_1, 3^+) \\ , \\ (p_2, 4^+) \end{bmatrix} \left[(p_1, 5^+)\right] \left[(p_1, 4^+)\right] \left[(p_1, 1^+)\right], \begin{bmatrix} (p_2, 7^+) \\ , \\ (p_3, 7^+) \end{bmatrix} \right),$$

$$R_3 = \left(\begin{bmatrix} (p_1, 1^0) \\ , \\ (p_3, 4^0) \end{bmatrix}, \begin{bmatrix} (p_1, 3^+) \\ , \\ (p_2, 4^+) \end{bmatrix} \begin{bmatrix} (p_2, 4^+) \\ , \\ (p_1, 5^+) \end{bmatrix} \left[(p_1, 1^+)\right], \begin{bmatrix} (p_2, 7^+) \\ , \\ (p_3, 7^+) \end{bmatrix} \right).$$

Let t be a transition with Let $In(t) = \left[(p_1, [3,5)), (p_2, [2,2])\right]$ and $Out(t) = \left[(p_2, (0,2)), (p_3, [2,5))\right]$. Examples of regions in $Pre_t(R)$ are the regions R_4 and R_5 defined as:

$$R_4 = \left(\begin{bmatrix} (p_1, 1^0) \\ , \\ (p_2, 2^0) \end{bmatrix}, \begin{bmatrix} (p_1, 3^+) \\ , \\ (p_2, 4^+) \\ , \\ (p_1, 4^+) \end{bmatrix} \left[(p_1, 5^+)\right] \left[(p_1, 1^+)\right], \begin{bmatrix} (p_2, 7^+) \\ , \\ (p_3, 7^+) \end{bmatrix} \right),$$

$$R_5 = \left(\begin{bmatrix} (p_1, 1^0) \\ , \\ (p_1, 2^0) \end{bmatrix}, \begin{bmatrix} (p_1, 3^+) \\ , \\ (p_2, 4^+) \end{bmatrix} \left[(p_1, 5^+)\right] \begin{bmatrix} (p_1, 1^+) \\ , \\ (p_1, 4^+) \end{bmatrix}, \begin{bmatrix} (p_2, 7^+) \\ , \\ (p_3, 7^+) \end{bmatrix} \right).$$

11.6. Checking entailment and proving WQO. Given regions $R = (Zero, Pos, Max)$ and $R' = (Zero', Pos', Max')$, we have $R \sqsubseteq R'$ iff $R \preceq_6 R'$ (see Section 10). It follows that $\sqsubseteq$ is computable and that is a WQO.

11.7. Intersection with initial configurations. We recall that C_{init} is the singleton $\{[(p_{init}, 0)]\}$. It follows that, for a region $R = (Zero, Pos, Max)$, we have $C_{init} \cap [\![R]\!] = \emptyset$ iff $Pos = \varepsilon$, $Max = \emptyset$, and either $Zero = \emptyset$ or $Zero = [(p_{init}, 0)]$.

§12. BQOs. We recall that in the previous sections we invented new constraint systems based on the fact that finite domains are WQOs under equality, and that WQOs are also closed under a basic set of operations such as building finite words, vectors, multisets, sets, etc. This means that we can start from a set of constraints over finite domains, and then repeatedly generate new constraints by building more compound data structures. A typical application of this approach was the constraint system of *regions* which we used for verification of TPNs in Section 11. Many of the constraint systems developed according to this methodology suffer from a "constraint explosion" problem, as a large number of constraints is generated during the reachability analysis algorithm. For instance, using regions, the set of generated constraints explodes even for very small TPNs. The constraint explosion can often be much reduced, either by employing the improved symbolic reachability algorithm using the relation $\preceq_{\forall\exists}$ (as described in Section 9); or by considering new constraint systems where each constraint is a set (disjunction) of the ones derived using the above mentioned set of operations. For instance, in Section 13 we will present *zones* which offer a much more compact representation of infinite sets of configurations than regions (in the same way that *zones* are more efficient than *regions* in verification tools for timed automata [29, 38]). We consider the region-induced ordering $\preceq$ on the configurations of a TPN, and apply the improved symbolic reachability algorithm on zones taking the entailment relation to be $\preceq_{\forall\exists}$.

Also, in Section 14 we will present a constraint system, where each constraint corresponds to the disjunction of a (usually large) number of simpler constraints (called *flat constraints*). Unfortunately, as we will describe in this section, well quasi-ordered constraint systems are in general not closed under disjunction or under applying the $\preceq_{\forall\exists}$ relation; and hence we cannot prove WQO of zones or of the constraints of Section 14 within the framework of Section 10. Therefore, instead of WQOs, we propose here to use an alternative approach based on a refinement of the theory of WQOs, called the theory of *Better Quasi-Qrderings* (*BQOs*) [33, 35]. We motivate why this theory allows for constraint systems which are more compact and hence less prone to constraint explosion. More precisely, BQOs offer two advantages: (i) BQO implies WQO; hence all the verification algorithms we have designed for well quasi-ordered constraint systems are also applicable to better quasi-ordered ones; and (ii) BQOs are more "robust" than WQOs. For instance, in addition to the operations of building sets, multisets, words, etc, better quasi-ordered constraint systems (in contrast to well quasi-ordered ones) are also (i) closed under disjunction: if a set of constraints is better quasi-ordered under entailment, then the set of finite disjunctions (sets) of these constraints is also better quasi-ordered under entailment; (ii) closed under $\preceq_{\forall\exists}$: if $\preceq$ is a better quasi-ordering on a set of configurations then $\preceq_{\forall\exists}$ is a better quasi-ordering on the denotations of any set of constraints.

12.1. Rado's example. We describe an example that illustrates why WQOs are in general not closed under disjunction or $\preceq_{\forall\exists}$. Consider the set $X \subseteq \mathbb{N}^2$ where $X = \{(a,b) \mid a < b\}$. Define a set $\Phi_1 = \{\phi_{a,b} \mid (a,b) \in X\}$ of constraints, such that the denotation $[\![\phi_{a,b}]\!] \subseteq X$ of $\phi_{a,b}$ is the set $\{(c,d) \mid (c > b) \vee ((c = a) \wedge (d \geq b))\}$. We show that Φ_1 is WQO under entailment. Suppose that we have a sequence $\phi_{a_1,b_1}, \phi_{a_2,b_2}, \ldots$. We show that the sequence is good. We consider two cases.

- If $b_1 < a_j$ for some $j \geq 1$. We show that this implies $\phi_{a_1,b_1} \sqsubseteq \phi_{a_j,b_j}$. Take any $(c,d) \in [\![\phi_{a_j,b_j}]\!]$. Then, either $c > b_j$ or $(c = a_j) \wedge (d \geq b_j)$. In both cases, we show that $c > a_1$ and hence $(c,d) \in [\![\phi_{a_1,b_1}]\!]$.
  - $c > b_j$. We have $a_j > b_1$ by assumption, and $b_1 > a_1$ and $b_j > a_j$ by definition of X. It follows that $c > a_1$.
  - $(c = a_j) \wedge (d \geq b_j)$. We know that $b_1 > a_1$ by definition of X, and $a_j > b_1$ by assumption. It follows that $c > a_1$.
- If $a_j \leq b_1$ for all $j \geq 1$. Then, we have a subsequence of the form $(a, b_{i_1}), (a, b_{i_2}), \ldots$ for some a, and hence there are k and ℓ such that $b_{i_k} \leq b_{i_\ell}$. We show that that this implies $\phi_{a,b_{i_k}} \sqsubseteq \phi_{a,b_{i_\ell}}$. Take any $(c,d) \in [\![\phi_{a,b_{i_\ell}}]\!]$. Then, either $c > b_{i_\ell}$ or $(c = a) \wedge (d \geq b_{i_\ell})$. In both cases, we show that $(c,d) \in [\![\phi_{a,b_{i_k}}]\!]$.
  - If $c > b_{i_\ell}$, then $c > b_{i_k}$ and hence $(c,d) \in [\![\phi_{a,b_{i_k}}]\!]$.
  - If $(c = a) \wedge (d \geq b_{i_\ell})$, then $d \geq b_{i_k}$ and hence $(c,d) \in [\![\phi_{a,b_{i_k}}]\!]$.

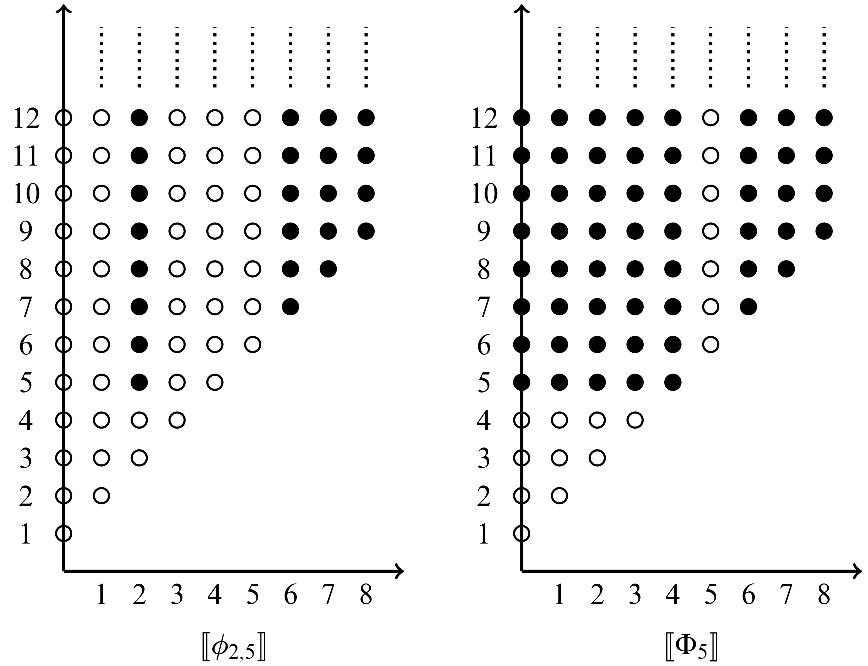

FIGURE 13. A graphic illustration of $[\![\phi_{2,5}]\!]$ and $[\![\Phi_5]\!]$. Filled circles represent points satisfying the constraint.

Now, we consider the set Φ_2 of constraints of the form ψ_j, where $\psi_j \equiv \phi_{0,j} \vee \cdots \vee \phi_{j-1,j}$. We show that $\psi_0, \psi_1, \ldots$ is bad. Consider $k < \ell$. We show that $(k,\ell) \in [\![\psi_\ell]\!]$, but $(k,\ell) \notin [\![\psi_k]\!]$. By definition of $\phi_{k,\ell}$, we know that $(k,\ell) \in [\![\phi_{k,\ell}]\!]$. Since $k < \ell$ we have $\phi_{k,\ell} \in \psi_\ell$, and hence $(k,\ell) \in [\![\psi_\ell]\!]$. Consider $\phi_{i,k}$ where $i < k$. Since $k \not> k$ and $k \neq i$ we know that that $(k,\ell) \notin [\![\phi_{i,k}]\!]$. By definition $\psi_k = \{\phi_{i,k} \mid 0 \leq i < k\}$, so $(k,\ell) \notin [\![\psi_k]\!]$. It follows that $k < \ell$ implies $[\![\psi_\ell]\!] \not\subseteq [\![\psi_k]\!]$, i.e., $\psi_k \not\sqsubseteq \psi_\ell$. This means that the sequence $\psi_1, \psi_2, \ldots$ is bad. In Figure 13, we give graphic illustrations of $[\![\phi_{2,5}]\!]$ and $[\![\psi_5]\!]$.

In fact, the above example also shows that WQOs are not closed under $\preceq_{\forall\exists}$. More precisely, suppose that we define the ordering $\preceq$ on X such that $(i,j) \preceq (k,\ell)$ iff either $i < \ell$ or both $i = k$ and $j \leq \ell$. Define the constraints $\psi_0, \psi_1, \ldots$ such that $[\![\psi_j]\!] = \cup_{0 \leq i < j} \{(k,\ell) \mid (i,j) \preceq (k,\ell)\}$. Notice that each ψ_j is upward closed wrt. $\preceq$. We can use the same reasoning as above to show that if $\ell > k$ then $(k,\ell) \in [\![\psi_\ell]\!]$ and $(k,\ell) \notin [\![\psi_k]\!]$. Since $[\![\psi_k]\!]$ is upward closed it follows that $(i,j) \notin [\![\psi_k]\!]$ for all $(i,j) \preceq (k,\ell)$. Hence, $\psi_k \not\preceq_{\forall\exists} \psi_\ell$. This means that the sequence $\psi_0, \psi_1, \ldots$ is bad wrt. $\preceq_{\forall\exists}$.

12.2. Basics of BQOs. We will introduce the basic definitions and properties of BQOs. Let $\mathbb{N}^{<*}$ ($\mathbb{N}^{<\omega}$) denote the set of finite (infinite) strictly increasing sequences over $\mathbb{N}$. For $s \in \mathbb{N}^{<*}$, we let $\lambda(s)$ be the set of natural numbers occurring in s, and if s is not empty then we let $tail(s)$ be the result of deleting the first element of s. For $s_1 \in \mathbb{N}^{<*}$ and $s_2 \in \mathbb{N}^{<*} \cup \mathbb{N}^{<\omega}$, we write $s_1 \ll s_2$ to denote that s_1 is a proper prefix of s_2. If s_1 is not empty then we write $s_1 \ll_* s_2$ to denote that $tail(s_1) \ll s_2$. An infinite set $\beta \subseteq \mathbb{N}^{<*}$ is said to be a barrier if the following two conditions are satisfied:

- there are no $s_1, s_2 \in \beta$ such that $\lambda(s_1) \subsetneq \lambda(s_2)$.
- for each $s_2 \in \mathbb{N}^{<\omega}$ there is $s_1 \in \beta$ with $s_1 \ll s_2$.

Let $(A, \preceq)$ be a quasi-ordering. An *A-pattern* is a mapping $f : \beta \to A$, where β is a barrier. We say that f is *good* if there are $s_1, s_2 \in \beta$ such that $s_1 \ll_* s_2$ and $f(s_1) \preceq f(s_2)$. We say that $(A, \preceq)$ is a *better quasi-ordering* if each A-pattern is good. We use A^ω to denote the set of infinite sequences over A. For $w \in A^\omega$, we let $w(i)$ be the the i^{th} element of w. For a quasi-ordering $(A, \preceq)$, we define the quasi-ordering $(A^\omega, \preceq^\omega)$ where $w_1 \preceq^\omega w_2$ if and only if there is a strictly increasing injection $h : \mathbb{N} \to \mathbb{N}$ such that $w_1(i) \preceq w_2(h(i))$, for each $i \in \mathbb{N}$. We shall use the following two properties (from [33]):

LEMMA 12.1. • *If β is a barrier and $\beta = \beta_1 \cup \beta_2$, then there is a barrier α such that $\alpha \subseteq \beta_1$ or $\alpha \subseteq \beta_2$.*
• *If $(A, \preceq)$ is BQO then $(A^\omega, \preceq^\omega)$ is BQO.*

Using induction on n we can generalize the first property to $\beta = \beta_1 \cup \cdots \cup \beta_n$. The base case $n = 2$ follows from the lemma. Consider $\beta = \beta_1 \cup \cdots \cup \beta_n$ where $n > 2$. Define $\beta' := \beta_2 \cup \cdots \cup \beta_n$. By the lemma there is a barrier α such that $\alpha \subseteq \beta_1$ or $\alpha \subseteq \beta'$. In the first case we are done. Otherwise, define $\alpha_i = \alpha \cap \beta_i$ for $i : 2 \leq i \leq n$. Notice that $\alpha = \alpha_2 \cup \cdots \cup \alpha_n$. By the induction hypothesis there is a barrier α' such that $\alpha' \subseteq \alpha_i$ for some $i : 2 \leq i \leq n$. The result follows from the fact that $\alpha_i \subseteq \beta_i$.

The following lemma gives the desired properties for BQOs.

THEOREM 12.2. 1. *Each BQO is a WQO.*
2. *If A is finite, then $(A, =)$ is a BQO.*
3. *If $(A, \preceq)$ is a BQO, then $(A^*, \preceq^*)$ is a BQO.*
4. *If $(A, \preceq)$ is a BQO, then $(A^\circledast, \preceq^\circledast)$ is a BQO.*
5. *If $(A, \preceq_1)$ is a BQO and $\preceq_1 \subseteq \preceq_2$ then $(A, \preceq_2)$ is a BQO.*
6. *If $(A, \preceq)$ is a BQO, then $\left(2^A, \preceq_{\forall\exists}\right)$ is a BQO.*

PROOF. The proof of property 6 can be found[3] in [31]. Below, we show properties 1–5.

1. Let $(A, \preceq)$ be a BQO. We show that $(A, \preceq)$ is a WQO. Consider a sequence $s = a_0, a_1, \ldots$ of elements in A. We show that s is good in the WQO-sense (i.e., there are i, j such that $i < j$ and $a_i \preceq a_j$). First, we show that the set $\mathbb{N}$ of natural numbers is a barrier (notice that each sequence in $\mathbb{N}$ is of length one):

[3][28] provides a proof for a weaker version of the theorem, namely that BQO of $(A, \preceq)$ is sufficient for WQO of $\left(2^A, \sqsubseteq\right)$.

- Take any $i, j \in \mathbb{N}$. If $i \neq j$ then the sets $\lambda(i) = \{i\}$ and $\lambda(j) = \{j\}$ do not intersect. Otherwise, they are identical. In both cases, $\lambda(i) \subsetneq \lambda(j)$.
- Consider any sequence $s \in \mathbb{N}^{<\omega}$. Let i be the first element in s. Then $i \in \mathbb{N}$ and $i \ll s$.

Define the function $f\colon \mathbb{N} \to A$ such that $f(i) = a_i$. Since $\mathbb{N}$ is a barrier it follows that f is an A-pattern. From the assumption that $(A, \preceq)$ is a BQO it follows that f is good in the BQO-sense, i.e., there are $i, j \in \mathbb{N}$ with $f(i) \preceq f(j)$. This means that $a_i \preceq a_j$ and hence s is good in the WQO-sense.

2. Consider $(A, =)$ where $A = \{a_1, \ldots, a_n\}$ is finite. Let $f\colon \beta \to A$ be an A-pattern. Define $\beta_i = f^{-1}(a_i)$, for $i\colon 1 \leq i \leq n$. By Lemma 12.1, there is a barrier $\alpha \subseteq \beta_i$, for some $i\colon 1 \leq i \leq n$. Take a shortest $s_1 \in \alpha$, i.e., there is no s_1' whose length is strictly smaller than the length of s_1. Consider any $s_2 \in \mathbb{N}^{<\omega}$, where $s_1 \ll_* s_2$. Since α is a barrier, we know that there is $s_3 \in \alpha$ such that $s_3 \ll s_2$, and that $\lambda(s_3) \subsetneq \lambda(s_1)$. From the fact that s_1 is of minimal length in β it follows that $s_1 \ll_* s_3$ and hence f is good.

3. Suppose that $(A, \preceq)$ is BQO. We show that $(A^*, \preceq^*)$ is BQO. Take any $b \notin A$. For $w \in A^*$, we let w' denote wb^ω (i.e., we add infinitely many b's to the end of w). It is clear that $w_1 \preceq^* w_2$ if and only if $w_1' \preceq^\omega w_2'$. Let $f\colon \beta \to A^*$ be an A^*-pattern. We know that $f'\colon \beta \to A^\omega$, where $f'(s) = w'$ iff $f(s) = w$, is an A^ω-pattern. By Lemma 12.1 it follows that there are $s_1, s_2 \in \beta$ such that $s_1 \ll_* s_2$ and $f'(s_1) \preceq^\omega f'(s_2)$, and hence $f(s_1) \preceq^* f(s_2)$. This means that f is good.

4. Follows from 3.

5. Suppose that $(A, \preceq_1)$ is a BQO. Consider an A-pattern $f\colon \beta \to A$. Since $\preceq_1$ is a BQO, it follows that f is good wrt. $\preceq_1$, i.e., there are $s_1, s_2 \in \beta$ such that $s_1 \ll_* s_2$ and $f(s_1) \preceq_1 f(s_2)$. Since $\preceq_1 \subseteq \preceq_2$ we know that $f(s_1) \preceq_2 f(s_2)$. This implies that f is good wrt. $\preceq_2$, and hence $\preceq_2$ is a BQO. ⊣

12.3. Applications of BQOs. Theorem 12.2 gives the properties we need to develop our methodology based on BQOs.

Since BQO is a stronger relation than WQO (property 1), it follows that, to prove termination of the reachability algorithms of the previous sections, it is sufficient to prove that the constraints are BQO under entailment.

Properties 2–3 mean that all the constraint systems we have built in the previous sections are BQO under entailment. In particular, the pre-orders $\preceq_1$, $\preceq_2$, $\preceq_3$, $\preceq_4$, $\preceq_5$, $\preceq_6$ in Section 10 are all BQOs.

Notice that 6 mentions explicitly that BQOs are stable under $\preceq_{\forall\exists}$. Also, by property 6, it follows that better quasi-ordered sets of constraints are closed under entailment: Consider a better quasi-ordered set of constraints $(\Phi, \sqsubseteq)$. By property 6 it follows that $(\Phi, \sqsubseteq_{\forall\exists})$ is a BQO. Notice that $\Phi_1 \sqsubseteq_{\forall\exists} \Phi_2$ implies that $\Phi_1 \sqsubseteq \Phi_2$. By property 5 it follows that $(\Phi, \sqsubseteq)$ is a BQO.

§13. Zones. In this section we introduce a constraint system called *zones* for representing infinite sets of configurations in Timed Petri nets. In a similar manner to regions, a zone Z represents minimal conditions on configurations, and thus characterizes an upward closed set of configurations. Compared with regions, zones provide a much more compact representation, and are therefore more suitable for reachability analysis. A zone Z specifies a minimum number of tokens which should be in the configuration, and then imposes certain conditions on these tokens. The conditions are formulated as specifications of the places in which the tokens should reside and restrictions on their ages. The age restrictions are stated as bounds on clocks values, and bounds on differences between values of pairs of clocks. A configuration c which satisfies Z should have at least the number of tokens specified by Z. Furthermore, the places and ages of these tokens should satisfy the conditions imposed by Z. In such a case, c may have any number of additional tokens (whose places and ages are then irrelevant for the satisfiability of the zone by the configuration).

13.1. Zones. Assume a TPN $\mathcal{N} = (P, T, F)$, and assume that *max* is the maximum integer which occurs syntactically in the definition of $\mathcal{N}$. For each $p \in P$, we will use a set X^p of variables ranging over $\mathbb{R}^{\geq 0}$. For $x \in X^p$, we use $type\,(x)$ to denote the place p. We use X to denote the set $\bigcup_{p \in P} X^p$. We will also assume a special variable $x^0 \notin X$. We use x^0 in our zone definitions to model a reference clock whose value is equal to zero. The type of x^0 is of no relevance. A *zone condition* φ is of the form $y - x \sim k$, where $\sim \in \{\leq, <\}$, $x, y \in X \cup \{x^0\}$, and $k \in \mathbb{Z}$. A *zone* Z is a finite conjunction of zone conditions. We use $Var\,(Z)$ to denote the set of variables in X which occur in Z. We occasionally consider a zone Z to be a set of zone conditions and write, for instance, $(y - x \sim k) \in Z$ to indicate that $y - x \sim k$ is one of the conjuncts in Z.

We define a total ordering $\lhd$ on elements in the set $\{<, \leq\} \times \mathbb{Z}$ such that $(\sim_1, k_1) \lhd (\sim_2, k_2)$ iff either

- $k_1 < k_2$; or
- $k_1 = k_2$ and either $\sim_1 = <$ or $\sim_2 = \leq$.

We define $(\sim_1, k_1) + (\sim_2, k_2)$ to be $(\sim_3, k_3)$ where $k_3 = k_1 + k_2$ and $\sim_3 = <$ iff either $\sim_1 = <$ or $\sim_2 = <$.

Consider a zone Z, a configuration $c = [(p_1, \alpha_1), \dots, (p_n, \alpha_n)]$, and an injection $h\colon Var\,(Z) \mapsto n^\bullet$ such that $type\,(x) = p_{h(x)}$ for each $x \in Var\,(Z)$. We extend h such that $h(x^0) = 0$. We write $c \models_h Z$ to denote that $\alpha_{h(y)} - \alpha_{h(x)} \sim k$ holds for all zone conditions $(y - x \sim k) \in Z$. When there is no risk of confusion, we simplify the notation and write $h(x)$ instead of $\alpha_{h(x)}$. For instance, we write $h(y) - h(x) \sim k$ instead of $\alpha_{h(y)} - \alpha_{h(x)} \sim k$. We write $c \models Z$ to denote that $c \models_h Z$ for some h. We define $[\![Z]\!] := \{c \mid c \models Z\}$. Intuitively, each variable in $Var\,(Z)$ represents one token. The configurations in $[\![Z]\!]$ contain tokens whose places are defined by the types of the corresponding variables, and whose clock values are related according to the zone conditions. To simplify the notation, we sometimes write $k < y - x$, $x > k$, and $x < k$ instead of $x - y < -k$, $x - x^0 < -k$, resp. $x - x^0 < k$ (we use a similar notation for $\leq$).

In a similar manner to regions (Section 11), our interpretation of zones is different from the standard one (described, e.g., in [15]). In the standard interpretation, zones characterize sets of (multi-clock) configurations, while in our interpretation a zone characterizes sets of tokens each with a single clock.

A zone Z is said to be *consistent* if $[\![Z]\!] \neq \emptyset$. We say that Z is *stable* if it satisfies the following condition:

- If $(y - x \sim_1 k_1) \in Z$ and $(z - y \sim_2 k_2) \in Z$ then $(z - x \sim_3 k_3) \in Z$ for some $(\sim_3, k_3) \lhd (\sim_1, k_1) + (\sim_2, k_2)$.

If the zone Z is stable we use the Floyd-Warshall procedure [20] to compute a zone Z' such that Z' is stable and $[\![Z']\!] = [\![Z]\!]$. We use *Stabilize*(Z) to denote Z'.

13.2. Computing predecessors. As in the case of regions (Section 11), we define $Pre := Pre_{Timed} \cup Pre_{Disc}$, where Pre_{Timed} corresponds to running time backwards and Pre_{Disc} corresponds to firing transitions backwards. Below, we assume a consistent zone Z.

First we define Pre_{Timed}. We define *Relax*(Z) to be the zone Z' we get from Z by replacing each zone condition of the form $x^0 - x \sim k$ (where $k \leq 0$ since Z is consistent) by a zone condition of the form $x^0 - x \sim 0$.

LEMMA 13.1. $Pre_{Timed}(Z) := Relax(Stabilize(Z))$.

Now, we show how to compute Pre_{Disc}. To do that, we define a number of operations. For an interval $\mathcal{I} = [a, b]$, and a variable $x \in Var(Z)$, we write $Z \otimes (\mathcal{I}, x)$ to be the zone $Z \cup \{a \leq x, x \leq b\}$. Intuitively, we add to Z the condition that the token corresponding to the variable x should have an age which belongs to the interval $[a, b]$. We define the operator for open intervals in a similar manner. We use $Z \ominus x$ to denote the zone Z' we get from Z by removing all conditions in which x occurs. For a place p and an interval $\mathcal{I} = [a, b]$, we define the *addition* $Z \oplus (p, \mathcal{I})$ of $(p, \mathcal{I})$ to Z to be the zone $Z' = Z \cup \{a \leq x, x \leq b\}$ where x is an arbitrary variable not in $Var(Z)$. Intuitively, the new zone Z' requires one additional token to be present in place p such that the age of the token is in the interval $\mathcal{I}$.

We define $Pre_{Disc} := \bigcup_{t \in T} Pre_t$, where Pre_t describes the effect of running the transition t backwards. Consider a transition $t \in T$. Let $In(t) = [(p_1, \mathcal{I}_1), \ldots, (p_k, \mathcal{I}_m)]$ and $Out(t) = [(q_1, \mathcal{J}_1), \ldots, (q_\ell, \mathcal{J}_n)]$. We define $Pre_t(Z)$ be the set of zones Z' such that there are

- $1 \leq i_1 < i_2 < \cdots < i_k \leq m$,
- variables $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$,
- zones Z_1, Z_2, Z_3

such that the following properties are satisfied:

- $\left[(q_{i_1}, \mathcal{J}_{i_1}), \ldots, (q_{i_k}, \mathcal{J}_{i_k})\right] \leq Out\,(t)$.
- $type\,(x_{i_j}) = q_{i_j}$.
- $Z_1 = Z \otimes (x_{i_1}, \mathcal{J}_{i_1}) \otimes \cdots \otimes (x_{i_k}, \mathcal{J}_{i_k})$.
- $Z_2 = Stabilize(Z_1)$.
- $Z_3 = Z_2 \ominus x_{i_1} \ominus \cdots \ominus x_{i_k}$.
- $Z' = Z_3 \oplus (p_1, \mathcal{I}_1) \oplus \cdots \oplus (p_m, \mathcal{I}_m)$.

13.3. Entailment. We recall from Section 11 that the $\sqsubseteq$ relation on regions coincides with the preorder $\preceq_6$ defined in Section 10. We also recall that ordering $\preceq$ defined on the configurations of a TPN concides with $\sqsubseteq$ in the sense that $c_1 \preceq c_2$ iff $Reg\,(c_1) \sqsubseteq Reg\,(c_2)$. We follow the methodology of Section 9 and implement the improved symbolic reachability algorithm, by defining the relation $\sqsubseteq$ on zones such that $Z_1 \sqsubseteq Z_2$ iff $[\![Z_1]\!] \preceq_{\forall\exists} [\![Z_2]\!]$. Below, we describe how to check the relation $\sqsubseteq$ on zones. To do that, we use formulas in a decidable logic, called *Difference Bound Logic* (DBL). The atomic formulas are of the form $y - x \sim k$, where x and y are variables interpreted over $\mathbb{R}^{\geq 0}$ and $k \in \mathbb{N}$. Furthermore the set of formulas is closed under the propositional connectives. Satisfiability of DBL-formulas is NP-complete [34]. NP-hardness follows by reducing the satisfiability problem for Boolean formulas. We represent each atomic proposition p by two variables x_p and y_p in the DBL formula. We replace each occurrence of p in the Boolean formula by the atomic formula $x_p - y_p \leq 0$. For NP-easiness, a non-deterministic algorithm works by guessing which zone conditions are true and which are false. A linear time test can check that the guess makes the entire formula true. A polynomial time test can check that the corresponding set of constraints on the reals is in fact satisfiable. The satisfiability of a conjunction of atomic formulas (a special case of linear programming) can be solved in cubic time using the Floyd-Warshall algorithm [20].

To give the characterization of entailment, we define the notion of areas. An *area condition* ψ is either of the form $(y - x \sim k) \vee (x > max - k)$ where $k \geq 0$; or of the form $(y - x \sim k) \vee (x > max)$ where $k < 0$. Given a zone condition $\varphi = (y - x \sim k)$, we use $\varphi^{\odot}$ to denote the area condition $(y - x \sim k) \vee (x > max - k)$ if $k \geq 0$, and the area condition $(y - x \sim k) \vee (x > max)$ if $k < 0$. An *area* A is a conjunction of area conditions. For a zone Z, we use $Z^{\odot}$ to be the area $\bigwedge_{\varphi \in Z} \varphi^{\odot}$. Given a zone Z with $Var\,(Z) = \{x_1, \ldots, x_m\}$, it is sometimes convenient to view Z as a predicate $Z(x_1, \ldots, x_m)$ on the set $\left(\mathbb{R}^{\geq 0}\right)^m$ (replacing any occurrence of x^0 by 0). Observe that $c \models_h Z$ iff $Z(h(x_1), \ldots, h(x_n))$ holds. This representation can be extended in the obvious manner to areas. Relations such as $c \models_h A$, $c \models A$, $[\![A_1]\!] \preceq_{\forall\exists} [\![A_2]\!]$, etc, are defined for areas in a similar manner to zones.

For zones Z_1 and Z_2, a renaming from Z_1 to Z_2 is an injection

$$\mathcal{R}\colon\ \mathit{Var}\,(Z_1) \mapsto \mathit{Var}\,(Z_2)$$

such that $\mathit{type}\,(x) = \mathit{type}\,\big(\mathcal{R}(x)\big)$. We use $\mathit{Ren}(Z_1)(Z_2)$ to denote the set of renamings from Z_1 to Z_2. The following lemma describes how to check the entailment relation.

LEMMA 13.2. *For zones Z_1 and Z_2 with $\mathit{Var}\,(Z_1) = \{x_1, \ldots, x_m\}$ and $\mathit{Var}\,(Z_2) = \{y_1, \ldots, y_n\}$, it is the case that $Z_1 \preceq_{\forall\exists} Z_2$ iff*

$$\forall y_1, \ldots, y_n.\ \left(\begin{array}{l} Z_2(y_1, \ldots, y_n) \Longrightarrow \\ \displaystyle\bigvee_{\mathcal{R}\in \mathit{Ren}(Z_1)(Z_2)} Z_1^{\odot}(\mathcal{R}(x_1), \ldots, \mathcal{R}(x_m)) \end{array}\right).$$

Notice that the above is a DBL-formula. We devote the rest of this subsection to the proof of Lemma 13.2. To do that, we introduce Lemma 13.3, Lemma 13.6, Lemma 13.7, and Lemma 13.8 (Lemma 13.2 follows directly from Lemma 13.7, and Lemma 13.8). The following lemma shows that, when we expand a zone Z to the area $Z^{\odot}$, then we only add configurations which are equivalent to ones which are already in in the Z.

LEMMA 13.3. *For a consistent and stable zone Z and a configuration $c \in [\![Z^{\odot}]\!]$, there is a configuration $c' \in [\![Z]\!]$ with $c \equiv_{\mathit{Reg}} c'$.*

PROOF. Let $Z = \varphi_1 \wedge \cdots \wedge \varphi_n$, i.e., $Z^{\odot} = {\varphi_1}^{\odot} \wedge \cdots \wedge {\varphi_n}^{\odot}$. Suppose that $c \models_h Z^{\odot}$. We show that $c' \models Z$ for some $c' \equiv_{\mathit{Reg}} c$. Without loss of generality, we assume that $\mathit{Var}\,(Z)$ is of the form $\{x_1, \ldots, x_\ell, x_{\ell+1}, \ldots, x_m\}$, where the following two conditions are satisfied:

1. $h(x_i) \leq \mathit{max}$ for each $i\colon 1 \leq i \leq \ell$; and
2. If $(x_j - x_i \sim k) \in Z$ for some $\ell < i, j \leq m$, and $(\sim, k) \lhd (\leq, -1)$, then $j < i$.

The second condition can be satisfied since Z is consistent.

We derive a sequence $c_\ell, c_{\ell+1}, \ldots, c_m$ of configurations and a corresponding sequence $h_\ell, h_{\ell+1}, \ldots, h_m$ such that $c_i \models_{h_i} Z \ominus x_{i+1} \ominus \cdots \ominus x_m$ for all $i\colon \ell \leq i \leq m$. For each case we first define h_i and then define $c_i = [(\mathit{type}(x_1), h(x_1)), \ldots, (\mathit{type}(x_i), h(x_i))]$.

We define $h_\ell(x_i) = h(x_i)$ for each $i\colon 1 \leq i \leq \ell$. Suppose that $c_\ell \not\models_{h_\ell} Z \ominus x_{\ell+1} \ominus \cdots \ominus x_m$. It follows that $h_\ell(x_j) - h_\ell(x_i) \nsim k$ for some $(x_j - x_i \sim k) \in Z$ where $1 \leq i, j \leq \ell$. There are two possible cases depending on whether k is negative. We show that we get a contradiction in each case:

- $k \geq 0$ and $(x_j - x_i \sim k) \vee (x_i > \mathit{max} - k) \in Z^{\odot}$. Since $c \models_h Z^{\odot}$, we have that $(h(x_j) - h(x_i) \sim k) \vee (h(x_i) > \mathit{max} - k)$. Since $h_\ell(x_j) = h(x_j)$ and $h_\ell(x_i) = h(x_i)$ it follows that $h(x_j) - h(x_i) \nsim k$. This means that $h(x_i) + k \leq h(x_j)$ and $h(x_i) > \mathit{max} - k$, and hence $h(x_j) > \mathit{max}$ which contradicts condition 1 above.

- $k < 0$ and $(x_j - x_i \sim k) \vee (x_i > max) \in Z^{\odot}$. Since $c \models_h Z^{\odot}$, we have that $(h(x_j) - h(x_i) \sim k) \vee (h(x_i) > max)$. Since $h_\ell(x_j) = h(x_j)$ and $h_\ell(x_i) = h(x_i)$ it follows that $h(x_j) - h(x_i) \not\sim k$. This means that $h(x_i) > max$ which contradicts condition 1 above.

Now, we consider $i : \ell < i \leq m$. We define $h_i(x_j) = h_{i-1}(x_j)$ if $j < i$, and define $h_i(x_i) = \rho$, where ρ is any number in $\mathbb{R}^{\geq 0}$ satisfying the following properties:

(a) $max < \rho$.
(b) If $(x_j - x_i \sim k) \in Z$ for some $j < i$ then $h_i(x_j) - k \sim \rho$.
(c) If $(x_i - x_j \sim k) \in Z$ for some $j < i$ then $\rho \sim h_i(x_j) + k$.

We show that such a ρ exists. Suppose that ρ does not exist. There are two possible cases each leading to a contradiction as follows.

- Conditions (b) and (c) cannot be satisfied. This means that there are j_1, j_2: $1 \leq j_1, j_2 < i$ such that $(x_{j_1} - x_i \sim_1 k_1) \in Z$, $(x_i - x_{j_2} \sim_2 k_2) \in Z$, and $h_i(x_{j_1}) - h_i(x_{j_2}) \not\sim_3 k_3$ where $(\sim_3, k_3) = (\sim_1, k_1) + (\sim_2, k_2)$. Since Z is consistent and stable, we know that $(x_{j_1} - x_{j_2} \sim_4 k_4) \in Z$ for some $(\sim_4, k_4) \lhd (\sim_3, k_3)$. Notice that $(x_{j_1} - x_{j_2} \sim_4 k_4) \in (Z \ominus x_i \ominus \cdots \ominus x_m)$. Since $c_{i-1} \models_{h_i} Z \ominus x_i \ominus \cdots \ominus x_m$ it follows that $h_{i-1}(x_{j_1}) - h_{i-1}(x_{j_2}) \sim_4 k_4$. From $h_i(x_{j_1}) = h_{i-1}(x_{j_1})$, $h_i(x_{j_2}) = h_{i-1}(x_{j_2})$, it follows that $h_i(x_{j_1}) - h_i(x_{j_2}) \sim_4 k_4$ which is a contradiction.
- Conditions (a) and (c) cannot be satisfied. This means that $(x_i - x_j \sim k) \in Z$, for some $1 \leq j < i$ and $h_i(x_j) \leq max - k$. We distinguish two subcases
  - If $1 \leq j \leq \ell$. Again, we distinguish two subcases:
    - If $k < 0$ then $((x_i - x_j \sim k) \vee (x_j > max)) \in Z^{\odot}$. Since $c \models_h Z^{\odot}$ it follows that $(h(x_i) - h(x_j) \sim k) \vee (h(x_j) > max)$. By condition 1, we know that $h(x_j) \leq max$. This means that $h(x_i) - h(x_j) \sim k$. By condition 1, we know that $h(x_i) > max$ and hence $max < h(x_j) + k$. Since $h_i(x_j) = h(x_j)$ it follows that $max < h_i(x_j) + k$ which contradicts $h_i(x_j) \leq max - k$.
    - If $k \geq 0$ then $((x_i - x_j \sim k) \vee (x_j > max - k)) \in Z^{\odot}$. Since $c \models_h Z^{\odot}$ it follows that $(h(x_i) - h(x_j) \sim k) \vee (h(x_j) > max - k)$. If $h(x_i) - h(x_j) \sim k$ then we get a contradiction in the same manner as above. Otherwise, $h(x_j) > max - k$. Since $h_i(x_j) = h(x_j)$ it follows that $h_i(x_j) > max - k$ which contradicts $h_i(x_j) \leq max - k$.
  - If $\ell < j \leq i - 1$. If $k < 0$, then condition 2 implies that $i < j$ which contradicts $j < i$. This means that $0 \leq k$. Since $h_i(x_j) > max$ it follows that $h_i(x_j) + k > max$ which contradicts $h_i(x_j) \leq max - k$.

It remains to show that $c_m \equiv_{Reg} c$. This follows from the fact that $h_m(x_i) = h(x_i)$ for all $i\colon 1 \leq i \leq \ell$ and both $h_m(x_i) > max$ and $h(x_i) > max$ for all $i\colon \ell < i \leq m$. ⊣

Next, we show (Lemma 13.6) that configurations which are region-equivalent satisfy the same areas. To do that, we define an equivalence relation $\approx$ on the set $\mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0}$, such that $(\alpha_1, \alpha_2) \approx \left(\alpha_1', \alpha_2'\right)$ iff the following conditions are satisfied:

- $sig(\alpha_i) = sig(\alpha_i')$ for $i \in \{1, 2\}$.
- if $\alpha_2 \leq max$ then $fract(\alpha_i) < fract(\alpha_j)$ iff $fract(\alpha_i') < fract\left(\alpha_j'\right)$, for $i, j \in \{1, 2\}$.

LEMMA 13.4. *If* $(\alpha_1, \alpha_2) \approx \left(\alpha_1', \alpha_2'\right)$ *and* $k \geq 0$ *then*

$$\begin{pmatrix} \alpha_2 - \alpha_1 \sim k \\ \vee \\ \alpha_1 > max - k \end{pmatrix} \textit{ iff } \begin{pmatrix} \alpha_2' - \alpha_1' \sim k \\ \vee \\ \alpha_1' > max - k \end{pmatrix}.$$

PROOF. We show the *only-if* direction. The *if*-direction is symmetric. We consider two cases

- $\alpha_1 \leq max - k$. It follows that $\alpha_2 - \alpha_1 \sim k$ holds and that $\alpha_2 \sim max$ (and hence $\alpha_2 \leq max$). Since $k \geq 0$ it follows that $\alpha_1 \leq max$. We notice that $\alpha_1 = k_1 + r_1$ and $\alpha_2 = k_2 + r_2$ where $k_i = \lfloor \alpha_i \rfloor$ and $r_i = fract(\alpha_i)$. Since $(\alpha_1, \alpha_2) \approx \left(\alpha_1', \alpha_2'\right)$ it follows that $\alpha_1' = k_1 + r_1'$ and $\alpha_2' = k_2 + r_2'$ where $r_i' = fract(\alpha_i')$ (the integral parts of α_i and α_i' are identical). We consider three cases:
  - $r_1 < r_2$. Since $\alpha_2 - \alpha_1 \sim k$ and $r_1 < r_2$ it follows that $k_2 - k_1 \leq k - 1$, and hence $\alpha_2' - \alpha_1' = (k_2 - k_1) + (r_2' - r_1') < k$.
  - $r_1 = r_2$. Since $(\alpha_1, \alpha_2) \approx \left(\alpha_1', \alpha_2'\right)$ and $\alpha_2 \leq max$ we know that $r_1' = r_2'$. It follows that $\alpha_2' - \alpha_1' = \alpha_2 - \alpha_1$ and therefore $\alpha_2' - \alpha_1' \sim k$.
  - $r_1 > r_2$. Since $\alpha_2 - \alpha_1 \sim k$ and $r_1 > r_2$ it follows that $k_2 - k_1 \leq k$. Since $(\alpha_1, \alpha_2) \approx \left(\alpha_1', \alpha_2'\right)$ it follows that $r_1' > r_2'$ and hence $\alpha_2' - \alpha_1' = (k_2 - k_1) + (r_2' - r_1') < k$.
- $\alpha_1 > max - k$. Since $(\alpha_1, \alpha_2) \approx \left(\alpha_1', \alpha_2'\right)$ it follows that $\alpha_1' > max - k$. ⊣

LEMMA 13.5. *If* $(\alpha_1, \alpha_2) \approx \left(\alpha_1', \alpha_2'\right)$ *and* $k < 0$ *then*

$$\begin{pmatrix} \alpha_2 - \alpha_1 \sim k \\ \vee \\ \alpha_1 > max \end{pmatrix} \textit{ iff } \begin{pmatrix} \alpha_2' - \alpha_1' \sim k \\ \vee \\ \alpha_1' > max \end{pmatrix}.$$

PROOF. We consider two cases

- $\alpha_1 \leq max$. It follows that $\alpha_2 - \alpha_1 \sim k$ and hence $\alpha_2 \sim \alpha_1 + k \leq max + k$. Since $k < 0$ it follows that $\alpha_2 < max$. The rest of the proof can be carried out in a similar manner to Lemma 13.4.
- $\alpha_1 > max$. Since $(\alpha_1, \alpha_2) \approx \left(\alpha_1', \alpha_2'\right)$ it follows that $\alpha_1' > max$. ⊣

LEMMA 13.6. *Consider an area A and configurations c_1 and c_2. If $c_1 \models A$ and $c_1 \equiv_{Reg} c_2$ then $c_2 \models A$.*

PROOF. Let $Reg\,(c)$ be of the form $(M_0, M_1 \dots M_n, M_{n+1})$, where M_i is of the form $\left[(p_{i1}, \ell_{i1}), \dots, (p_{in_i}, \ell_{in_i})\right]$. By definition we know that there are configurations $c_0, \dots, c_{n+1}$ such that

- $c = c_0 + \dots + c_{n+1}$.
- c_i is of the form $\left[(p_{i1}, \alpha_{i1}), \dots, (p_{in_i}, \alpha_{in_i})\right]$, where $\ell_{ij} = sig\left(\alpha_{ij}\right)$ for all $i : 0 \leq i \leq n+1$ and $j : 1 \leq j \leq n_i$.
- $fract\left(\alpha_{i_1 j_1}\right) < fract\left(\alpha_{i_2 j_2}\right)$ iff $i_1 < i_2$ for all $i_1, i_2 \colon 0 \leq i_1, i_2 \leq n+1$, $j_1 \colon 1 \leq j_1 \leq n_{i_1}$, and $j_2 \colon 1 \leq j_2 \leq n_{i_2}$.

Since $Reg\,(c') = Reg\,(c)$, there are also configurations $c'_0, \dots, c'_{n+1}$ such that

- $c' = c'_0 + \dots + c'_{n+1}$.
- c'_i is of the form $\left[\left(p_{i1}, \alpha'_{i1}\right), \dots, \left(p_{in_i}, \alpha'_{in_i}\right)\right]$, where $\ell_{ij} = sig\left(\alpha'_{ij}\right)$ for all $i : 0 \leq i \leq n+1$ and $j : 1 \leq j \leq n_i$.
- $fract\left(\alpha'_{i_1 j_1}\right) < fract\left(\alpha'_{i_2 j_2}\right)$ iff $i_1 < i_2$ for all $i_1, i_2 \colon 0 \leq i_1, i_2 \leq n+1$, $j_1 \colon 1 \leq j_1 \leq n_{i_1}$, and $j_2 \colon 1 \leq j_2 \leq n_{i_2}$.

This implies that $\left(\alpha_{i_1 j_1}, \alpha_{i_2 j_2}\right) \approx \left(\alpha'_{i_1 j_1}, \alpha_{i_2 j_2}\right)$ for all $i_1, i_2 \colon 0 \leq i_1, i_2 \leq n+1$, $j_1 \colon 1 \leq j_1 \leq n_{i_1}$, and $j_2 \colon 1 \leq j_2 \leq n_{i_2}$.

Since $c \models A$ we know that $c \models_h A$ for some injection h. We show that $c' \models_h A$ which implies the result.

- if $\left(\alpha_{h(y)} - \alpha_{h(x)} \sim k\right) \vee \left(\alpha_{h(x)} > max - k\right) \in A$ and $k \geq 0$. Since $c \models A$ it follows that $\left(\alpha_{h(y)} - \alpha_{h(x)} \sim k\right) \vee \left(\alpha_{h(x)} > max - k\right)$ holds. By Lemma 13.4 and the fact that $\left(\alpha_{h(x)}, \alpha_{h(y)}\right) \approx \left(\alpha'_{h(x)}, \alpha'_{h(y)}\right)$ it follows that $\left(\alpha'_{h(y)} - \alpha'_{h(x)} \sim k\right) \vee \left(\alpha'_{h(x)} > max - k\right)$ holds.
- if $\left(\alpha_{h(y)} - \alpha_{h(x)} \sim k\right) \vee \left(\alpha_{h(x)} > max\right) \in A$ and $k < 0$. Since $c \models A$ it follows that $\left(\alpha_{h(y)} - \alpha_{h(x)} \sim k\right) \vee \left(\alpha_{h(x)} > max\right)$ holds. By Lemma 13.5 and the fact that $\left(\alpha_{h(x)}, \alpha_{h(y)}\right) \approx \left(\alpha'_{h(x)}, \alpha'_{h(y)}\right)$ it follows that $\left(\alpha'_{h(y)} - \alpha'_{h(x)} \sim k\right) \vee \left(\alpha'_{h(x)} > max\right)$ holds. ⊣

LEMMA 13.7. *For zones Z_1 and Z_2 with $Var\,(Z_1) = \{x_1, \dots, x_m\}$ and $Var\,(Z_2) = \{y_1, \dots, y_n\}$, if*

$$\forall y_1, \dots, y_n. \left(\begin{array}{l} Z_2(y_1, \dots, y_n) \Longrightarrow \\ \displaystyle \bigvee_{\mathcal{R} \in Ren(Z_1)(Z_2)} Z_1{}^{\odot}(\mathcal{R}(x_1), \dots, \mathcal{R}(x_m)) \end{array} \right)$$

then $Z_1 \preceq_{\forall\exists} Z_2$.

PROOF. Suppose that $c_2 = [(p_1, \alpha_1), \ldots, (p_k, \alpha_k)] \models Z_2$. We show that there is a c_3 such that $c_3 \models Z_1$ and $c_3 \preceq c_2$. Since $c_2 \models Z_2$, we know that there is a mapping $h_2\colon \{y_1, \ldots, y_n\} \mapsto k^\bullet$ such that $c_2 \models_{h_2} Z_2$. This means that $type(y_i) = p_{h_2(y_i)}$ for all $i\colon 1 \le i \le n$ and that $Z_2(\alpha_{h_2(y_1)}, \ldots, \alpha_{h_2(y_n)})$ holds. By the premise of the lemma it follows that there is a renaming $\mathcal{R} \in Ren(Z_1)(Z_2)$ such that ${Z_1}^{\odot}(\alpha_{h_2(\mathcal{R}(x_1))}, \ldots, \alpha_{h_2(\mathcal{R}(x_m))})$ holds. Define $c_1 := \left[\left(p_{h_2(\mathcal{R}(x_1))}, \alpha_{h_2(\mathcal{R}(x_1))}\right), \ldots, \left(p_{h_2(\mathcal{R}(x_m))}, \alpha_{h_2(\mathcal{R}(x_m))}\right)\right]$. Clearly $c_1 \le c_2$. Define the mapping $h_1\colon \{x_1, \ldots, x_m\} \mapsto m^\bullet$ such that $h_1(x_i) = i$ for all $i\colon 1 \le i \le m$. By definition of a renaming, we have, for all $i\colon 1 \le i \le m$, that $type(x_i) = type\left(\mathcal{R}(x_i)\right)$, and hence $type(x_i) = p_{h_2(\mathcal{R}(x_i))}$. From this and the fact that ${Z_1}^{\odot}(\alpha_{h_2(\mathcal{R}(x_1))}, \ldots, \alpha_{h_2(\mathcal{R}(x_m))})$ holds it follows that $c_1 \models_{h_2} {Z_1}^{\odot}$. By Lemma 13.3, there is a c_3 such that $c_3 \equiv_{Reg} c_1$ and $c_3 \models Z_1$. Since $c_1 \le c_2$, we have that $c_3 \preceq c_2$. $\dashv$

LEMMA 13.8. *For zones Z_1 and Z_2 with $Var(Z_1) = \{x_1, \ldots, x_m\}$ and $Var(Z_2) = \{y_1, \ldots, y_n\}$, if $Z_1 \preceq_{\forall\exists} Z_2$ then*

$$\forall y_1, \ldots, y_n. \left(\begin{array}{l} Z_2(y_1, \ldots, y_n) \Longrightarrow \\ \displaystyle\bigvee_{\mathcal{R} \in Ren(Z_1)(Z_2)} {Z_1}^{\odot}(\mathcal{R}(x_1), \ldots, \mathcal{R}(x_m)) \end{array} \right).$$

PROOF. Suppose that there is a mapping $g\colon \{y_1, \ldots, y_n\} \mapsto \mathbb{R}^{\ge 0}$ such that $Z_2(g(y_1), \ldots, g(y_n))$ holds. We show that there is a renaming $\mathcal{R}$ from Z_1 to Z_2 such that ${Z_1}^{\odot}(g(\mathcal{R}(x_1)), \ldots, g(\mathcal{R}(x_m)))$ holds. Define $c_2 := [(p_1, \alpha_1), \ldots, (p_n, \alpha_n)]$ where $p_i = type(y_i)$ and $\alpha_i = g(y_i)$ for all $i\colon 1 \le i \le n$. Define $h_2\colon \{y_1, \ldots, y_n\} \mapsto n^\bullet$ such that $h_2(y_i) = i$. Obviously, $c_2 \models_{h_2} Z_2$. Since $Z_1 \preceq_{\forall\exists} Z_2$, there is a c_1 such that $c_1 \preceq c_2$ and $c_1 \models Z_1$. It follows by definition that there is a c_3 such that $c_3 \le c_2$ and $c_1 \equiv_{Reg} c_3$. Let c_3 be of the form $\left[\left(p_{j_1}, \alpha_{j_1}\right), \ldots, \left(p_{j_k}, \alpha_{j_k}\right)\right]$. Since $c_1 \models Z_1$ it follows that $c_1 \models {Z_1}^{\odot}$. From Lemma 13.6 we get $c_3 \models {Z_1}^{\odot}$, i.e., there is a mapping $h_1\colon \{x_1, \ldots, x_m\} \mapsto k^\bullet$ such that $type(x_i) = p_{j_{h_1(x_i)}}$ for all $i\colon 1 \le i \le m$, and ${Z_1}^{\odot}(g(y_{j_{h_1(x_1)}}), \ldots, g(y_{j_{h_1(x_m)}}))$ holds. Define the mapping $\mathcal{R} \in Ren(Z_1)(Z_2)$ such that $\mathcal{R}(x_i) = y_{j_{h_1(x_i)}}$ for all $i\colon 1 \le i \le m$. We notice that $type(x_i) = p_{j_{h_1(x_i)}} = type\left(y_{j_{h_1(x_i)}}\right)$, and hence $\mathcal{R}$ is a renaming. Since ${Z_1}^{\odot}(g(y_{j_{h_1(x_1)}}), \ldots, g(y_{j_{h_1(x_m)}}))$ holds and $\mathcal{R}(x_i) = y_{j_{h(x_i)}}$ it follows that ${Z_1}^{\odot}(g(\mathcal{R}(x_1)), \ldots, g(\mathcal{R}(x_m)))$ holds. $\dashv$

§14. Constrained multiset rewriting systems. In this Section, we consider *Constrained Multiset Rewriting Systems* (*CMRS*) [3]. A CMRS operates on configurations which are multisets of monadic predicate symbols, each with an argument ranging over the natural numbers. Transitions between configurations are defined by a finite set of rewriting rules. Each rule is conditioned by *gap-order* formulas [37] of the form $x + c < y$, $x = y$, $x < c$, $x > c$, or $x = c$, where x and y are variable ranging over the natural numbers and c is a natural number. This model can capture the behaviour of parameterized systems (systems with arbirary numbers of componenets) in which the internal states of individual components may contain values ranging over the natural numbers. There are several examples of classes of protocols which can be modelled in this manner, e.g., mutual exclusion protocols where the natural number inside each process is used to define the identity of the process, and authentication protocols where the number is used to define the key assigned to the process. In addition to the relevance of CMRS as a formalism for parameterized systems, they are also interesting as a computation model in their own right. For instance, CMRS subsume several existing models for infinite-state systems such as Petri nets and relational automata [18]. In fact, it can be shown [3] that CMRS are strictly more powerful than both these models.

14.1. Model. We assume a set $\mathbb{V}$ of variables which range over the integers, and a set $\mathbb{P}$ of unary predicate symbols. For a set $V \subseteq \mathbb{V}$, a *valuation Val* of V is a mapping from V to $\mathbb{N}$, and a renaming $\mathcal{R}$ of V is a mapping from V to $\mathbb{V}$. A renaming $\mathcal{R}$ need not be injective, i.e., several variables may be renamed to the same variable by $\mathcal{R}$. We say that $\mathcal{R}$ is a renaming to W if $\mathcal{R}(x) \in W$ for each $x \in V$. When the set V is clear from the context, we do not mention it; simply saying valuation (rather than valuation of V) and renaming (rather than renaming of V). Sometimes, we write the explicit definition of a renaming. For instance $\mathcal{R} = (x_1 \mapsto w_1, x_2 \mapsto w_2, x_3 \mapsto w_3)$ stands for $\mathcal{R}(x_1) = w_1$, $\mathcal{R}(x_2) = w_2$, and $\mathcal{R}(x_3) = w_3$. We use a similar notation for valuations.

A *condition* is a finite conjunction of formulas of the forms: $x <_c y$ or $x = y$, where $x, y \in \mathbb{V}$ and $c \in \mathbb{N}$. Here $x <_c y$ stands for $x + c < y$. Sometimes, we treat a condition ψ as a set, and write, e.g., $(x <_c y) \in \psi$ to indicate that $x <_c y$ is one of the conjuncts in ψ. A *term* is of the form $p(x)$ where $p \in \mathbb{P}$ and $x \in \mathbb{V}$. A *ground term* is of the form $p(c)$ where $p \in \mathbb{P}$ and $c \in \mathbb{N}$. A *Constrained Multiset Rewriting System (CMRS)* $\mathcal{S}$ consists of a finite set of *rules* each of the form:

$$L \hookrightarrow R \colon \psi$$

where L and R are multisets of terms, and ψ is a condition. We assume that ψ is consistent (otherwise, the rule is never enabled). For a condition ψ, we use *Var* (ψ) to denote the set of variables which occur in ψ. For a valuation *Val*, we use $Val(\psi)$ to denote the result of substituting each variable x in ψ by $Val(x)$. We use $Val \models \psi$ to denote that $Val(\psi)$ evaluates to *true*. Also, for a renaming $\mathcal{R}$, we define $\mathcal{R}(\psi)$ to be the condition we get by replacing each x in ψ by $\mathcal{R}(x)$. For a multiset T of terms we define *Var* (T), $Val(T)$, and $\mathcal{R}(T)$ in a similar manner. In particular, $\mathcal{R}(T)$ and $Val(T)$ are multisets of terms and ground terms respectively. For a rule ρ of the above form, we define $\mathit{Var}(\rho) = \mathit{Var}(L) \cup \mathit{Var}(R) \cup \mathit{Var}(\psi)$.

14.2. LTS. We describe how a CMRS induces an LTS $\mathcal{T} = (C, \longrightarrow, \preceq, C_{init})$. A *configuration* is a multiset of ground terms. The transition relation $\longrightarrow$ is induced by a set of rules. Abusing notation, we use $\xrightarrow{\rho}$ to represent the effect of applying the rule ρ, and define $\longrightarrow := \bigcup_{\rho \in \mathcal{S}} \xrightarrow{\rho}$.

More precisely, for a rule ρ of the form $L \hookrightarrow R : \psi$, we have $c_1 \xrightarrow{\rho} c_2$ if there is a valuation *Val* such that the following three conditions are satisfied:

- $\mathit{Val} \models \psi$, • $c_1 \geq \mathit{Val}(L)$, • $c_2 = c_1 - \mathit{Val}(L) + \mathit{Val}(R)$.

For a configuration c and a predicate symbol p, we use $c \xrightarrow{*} p$ to denote that p occurs in some c' with $c \xrightarrow{*} c'$.

We define the ordering $\preceq$ on the set of configurations to be the relation $\leq$ on multisets. We assume an *initial* configuration c_{init}, and a *final* predicate symbol p_{fin}. We will check whether $c_{init} \xrightarrow{*} p_{fin}$. Notice that p_{fin} characterizes an upward closed set of configurations.

EXAMPLE. The definition of the transition relation $\longrightarrow$ interprets a rule of the form given above as a collection of rewriting rules on ground terms. An instance is obtained by taking a valuation which satisfies ψ. Consider the rule:

$$[p(x), q(y)] \hookrightarrow [q(z), r(x), r(w)] : \{x <_2 y, x <_4 z, z < w\}.$$

A valuation which satisfies the condition is $\mathit{Val}(x) = 1$, $\mathit{Val}(y) = 4$, $\mathit{Val}(z) = 8$, $\mathit{Val}(w) = 10$, Therefore, we have a transition

$$[p(1), p(3), q(4)] \longrightarrow [p(3), q(8), r(1), r(10)].$$

14.3. Constraint system. We define a constraint system for CMRS as follows. A *constraint* ϕ is of the form $T : \psi$ where T is a multiset of terms and ψ is a condition. The constraint characterizes the (upward closed) set $[\![\phi]\!] = \{c \mid \exists \mathit{Val}.\ (\mathit{Val} \models \psi) \wedge (\mathit{Val}(T) \leq c)\}$ of configurations. Notice that if ψ is inconsistent, then $[\![\phi]\!]$ is empty. Such a constraint can be safely discarded in the reachability algorithm presented below. Therefore, we assume in the sequel that all conditions in constraints are consistent. We define $\mathit{Var}(\phi) := \mathit{Var}(T) \cup \mathit{Var}(\psi)$.

EXAMPLE. Consider the constraint $\phi_1 = [p(x_1), q(x_2), q(x_3)] : \{x_1 <_2 x_2, x_2 <_1 x_3\}$, and the configurations $c_1 = [p(2), q(8), q(5), p(1)]$ and $c_2 = [p(2), q(2), q(5), p(1)]$. Then $c_1 \in [\![\phi_1]\!]$ and $c_2 \notin [\![\phi_1]\!]$. Consider the constraints $\phi_2 = [p(y_1),\ q(y_2)] : \{y_1 < y_2\}$ and $\phi_3 = [p(y_1),\ q(y_2)] : \{y_1 <_4 y_2\}$. Then $\phi_2 \sqsubseteq \phi_1$ and $\phi_3 \not\sqsubseteq \phi_1$.

Below, we show computability of membership, entailment, and the predecessor function for constraints. First, we define a normal form for constraints. A constraint $T : \psi$ is said to be in *normal form* whenever the condition ψ satisfies the following requirements:

1. if $(x <_{c_1} y) \in \psi$ and $(y <_{c_2} z) \in \psi$ then $(x <_{c_3} z) \in \psi$ for some c_3 with $c_1 + c_2 + 1 < c_3$.
2. if $(x <_c y) \in \psi$ and $(y = z) \in \psi$ then $(x <_c z) \in \psi$.
3. if $(x <_c y) \in \psi$ and $(x = z) \in \psi$ then $(z <_c y) \in \psi$.
4. if $(x = y) \in \psi$ and $(y = z) \in \psi$ then $(x = z) \in \psi$.
5. For each $x, y \in \mathbb{V}$, at most one conjunct in ψ contains both x and y.
6. $Var(\psi) \subseteq Var(T)$.

LEMMA 14.1. *For each constraint ϕ we can effectively compute a constraint ϕ_{norm} such that ϕ_{norm} is in normal form and such that $[\![\phi]\!] = [\![\phi_{norm}]\!]$.*

The normalization procedure consists of repeatedly adding conjuncts to ψ which maintain properties 1-4 and removing conjuncts which violate property 5. When the above procedure stabilizes, we remove all conjuncts in ψ containing variables not in $Var(T)$. Normalization can also be used to check consistency: the constraint is consistent if and only if no inequalities of the form $x <_c x$ are generated.

LEMMA 14.2. *Membership, entailment, and Pre are computable for constraints.*

The full proof of the lemma is given in [3]. The main concepts are the following. For a constraint ϕ and a configuration c, it follows by definition that $c \in [\![\phi]\!]$ iff there is a valuation *Val* of $Var(\phi)$ such that $Val(\psi) \wedge \big(c \geq Val(T)\big)$. Computability follows since there are only finitely many valuations *Val* with $c \geq Val(T)$.

Consider constraints $\phi_1 = (T_1 : \psi_1)$ and $\phi_2 = (T_2 : \psi_2)$ which are in normal form (by Lemma 14.1 this is not a restriction). Let $Ren(\phi_1)(\phi_2)$ be the set of renamings $\mathcal{R}$ of $Var(T_1)$ such that $\mathcal{R}(T_1) \leq T_2$. Then $\phi_1 \sqsubseteq \phi_2$ is characterized by the formula by

$$\forall x_1 \ldots x_2. \Big(\psi_2 \Longrightarrow \bigvee_{\mathcal{R} \in Ren(\phi_1)(\phi_2)} \mathcal{R}(\psi_1)\Big).$$

Since the set $Ren(\phi_1)(\phi_2)$ is finite, checking the formula amounts to checking the satisfiability of a Boolean combination of formulas of the forms $x = y$ or $x <_c y$.

Let $\mathcal{S}$ be a CMRS and ϕ_2 be a constraint. We define

$$Pre(\phi_2) = \bigcup_{\rho \in \mathcal{S}} Pre_\rho(\phi_2),$$

where $Pre_\rho(\phi_2)$ describes the effect of running the rule ρ backwards from the configurations in ϕ_2. Let $\rho = (L \hookrightarrow R \colon \psi)$ and $\phi_2 = (T_2 : \psi_2)$. Let W be any set of variables such that $|W| = |Var(\phi_2) \cup Var(\rho)|$. We define $Pre_\rho(\phi_2)$ to be the set of constraints of the form $T_1 : \psi_1$, such that there are renamings $\mathcal{R}, \mathcal{R}_2$ of $Var(\rho)$ and $Var(\phi_2)$ respectively to W, and

- $T_1 = \mathcal{R}_2(T_2) \ominus \mathcal{R}(R) + \mathcal{R}(L)$,
- $\psi_1 = \mathcal{R}(\psi) \wedge \mathcal{R}_2(\psi_2)$.

EXAMPLE. Consider the constraints

$\phi_1 = [p(x_1), q(x_2), q(x_3), r(x_4)] \colon \{x_1 <_1 x_2,\ x_2 <_3 x_4,\ x_1 < x_3,\ x_1 <_8 x_4\}$,
$\phi_2 = [p(y_1), q(y_2), q(y_3), r(y_4), s(y_5)] \colon$
$\{y_1 <_1 y_3,\ y_2 <_3 y_3,\ y_1 < y_4,\ y_2 <_2 y_4\}$.

Then the set $Ren(\phi_1)(\phi_2) = \{\mathcal{R}_1, \mathcal{R}_2\}$ where

$$\mathcal{R}_1 = (x_1 \mapsto y_1,\ x_2 \mapsto y_2,\ x_3 \mapsto y_3,\ x_4 \mapsto y_4),$$
$$\mathcal{R}_2 = (x_1 \mapsto y_1,\ x_2 \mapsto y_3,\ x_3 \mapsto y_2,\ x_4 \mapsto y_4).$$

Therefore, $\phi_1 \sqsubseteq \phi_2$ is characterized by validity of the formula

$$\begin{pmatrix} y_1 <_1 y_3 \\ y_2 <_3 y_3 \\ y_1 < y_4 \\ y_2 <_2 y_4 \end{pmatrix} \Longrightarrow \begin{pmatrix} y_1 <_1 y_2 \\ y_2 <_3 y_4 \\ y_1 < y_3 \\ y_1 <_8 y_4 \end{pmatrix} \vee \begin{pmatrix} y_1 <_1 y_3 \\ y_3 <_3 y_4 \\ y_1 < y_2 \\ y_1 <_8 y_4 \end{pmatrix}.$$

Consider the constraint $\phi = [q(x_1), s(x_2), r(x_2)] \colon \{x_1 < x_2\}$ and the rule $\rho = \left[p(y_1), p(y_3)\right] \hookrightarrow \left[q(y_2), r(y_3)\right] \colon \{y_3 < y_2\}$. Fix $W = \{w_1, w_2, w_3, w_4, w_5\}$, and define

$$\mathcal{R}_2 = (x_1 \mapsto w_1,\ x_2 \mapsto w_2),$$
$$\mathcal{R} = (y_1 \mapsto w_3,\ y_2 \mapsto w_1,\ y_3 \mapsto w_4).$$

Then one member of Pre_ρ is given by $[s(w_2), r(w_2), p(w_3), p(w_4)] \colon \{w_1 < w_2, w_4 < w_1\}$, which after normalization becomes $[s(w_2), r(w_2), p(w_3), p(w_4)]$: $\{w_4 <_1 w_2\}$. Observe that (i) the normalization procedure may introduce new constants (1 in this case) which are not part of the original constraint; (ii) if we choose $\mathcal{R} = (y_1 \mapsto w_3,\ y_2 \mapsto w_1,\ y_3 \mapsto w_2)$ then the resulting constraint will denote an empty set (its conditions will be inconsistent); (iii) the size of constraints may increase when computing *Pre*.

14.4. BQO. We follow the methodology of Section 12 to show that $\sqsubseteq$ is a BQO. The challenging task in applying the method is to find an "intermediate" class of constraints, here called *flat constraints*, and then showing that (i) flat constraints are BQO; and (ii) each constraint is the union of a finite set of flat constraints. A *flat constraint* ϕ_{fl} is of the form $M_0 d_1 M_1 d_2 \ldots d_n M_n$ where $M_0, M_1, \ldots, M_n \in \mathbb{P}^{\circledast}$ and $d_1, d_2, \ldots, d_n \in \mathbb{N}$. In other words, a flat constraint is a word which alternatively contains multisets of predicate symbols and natural numbers, starting and ending with multisets of predicate symbols. Furthermore, we require that the multisets $M_0, M_1, \ldots, M_n$ are all non-empty. For a configuration c, we have $c \in [\![\phi_{fl}]\!]$ if there are natural numbers $c_0, \ldots, c_n \in \mathbb{N}$ such that (i) $c_i - c_{i-1} > d_i$ for each $i \colon 1 \leq i \leq n$; and (ii) $c(p(c_i)) \geq M_i(p)$ for each predicate symbol p and $i \colon 0 \leq i \leq n$.

We observe that Each M_i is a multiset over the finite set $\mathbb{P}$. Therefore the M_i's are BQO under the multiset ordering $\leq$ by Properties 2 and 4 in

Theorem 12.2. By a similar reasoning, the d_i's are BQO under the standard ordering $\leq$ on natural numbers. Since each flat constraint is a finite word of M_i's and d_i's, it follows by Property 3 in Theorem 12.2 that flat constraints are BQO under $\sqsubseteq$. Finally $\sqsubseteq$ is WQO by Property 1 of Theorem 12.2. This gives the following.

LEMMA 14.3. *The entailment relation* $\sqsubseteq$ *is a BQO* (*and therefore also a WQO*) *on flat constraints.*

EXAMPLE. Consider the flat constraint $\phi_1 = [p]\,4\,[q,r]\,2\,[r]$, and the configurations $c_1 = [p(1), q(7), r(7), r(11), q(3)]$, $c_2 = [p(1), q(7), r(7), r(8), q(3)]$, and $c_3 = [p(1), q(7), r(8), r(11), q(3)]$. Then $c_1 \in [\![\phi_1]\!]$ and $c_2, c_3 \notin [\![\phi_1]\!]$.

Consider the flat constraints $\phi_2 = [p]\,2\,[r]\,1\,[r]$, $\phi_3 = [p]\,5\,[r]$, and $\phi_4 = [p]\,10\,[r]$. Then $\phi_2 \sqsubseteq \phi_1$, $\phi_3 \sqsubseteq \phi_1$, and $\phi_4 \not\sqsubseteq \phi_1$.

Now, we show how to translate constraints into flat constraints.

Consider a constraint $\phi = T : \psi$ in normal form. A *flattening* F of $Var(\phi)$ is a word of the form $X_0 d_1 X_1 d_2 \ldots d_n X_n$ where $d_1, d_2, \ldots, d_n \in \mathbb{N}$ and the following three conditions are satisfied:

- $X_0, X_1, \ldots, X_n$ is a partitioning of $Var(\phi)$.
- If $(x = y) \in \psi$ then $x, y \in X_i$ for some $i : 1 \leq i \leq n$.
- If $(x <_c y) \in \psi$, $x \in X_i$, and $y \in X_j$ then $c \leq \left(\sum_{k=i+1}^{j} d_k + 1\right)$.

Intuitively, variables which are required to be equal by ϕ, are put in the same X_i. Also, variables which are ordered according to ϕ, are placed sufficiently far apart to cover the corresponding gap. The *flattening* ϕ_{fl} of ϕ *induced* by F is a flat constraint $M_0 d_1 M_1 d_2 \ldots d_n M_n$ such that $M_i(p) = \sum_{x \in X_i} T(p(x))$ for each $p \in \mathbb{P}$ and $i : 1 \leq i \leq n$. Since ϕ is in normal form, it follows that $Var(T) = Var(\phi)$ and hence M_i is not empty for each $i : 1 \leq i \leq n$.

EXAMPLE. Consider the constraint

$$\phi = [p(x_1), q(x_2), r(x_3), r(x_4)] : \{x_1 <_2 x_2, x_1 <_1 x_3, x_2 <_3 x_4, x_1 <_8 x_4\}.$$

A flattening of the condition and the induced flat constraint is given by $\{x_1\}\ 3\ \{x_2, x_3\}\ 5\ \{x_4\}$ resp. $[p]\,3\,[q,r]\,5\,[r]$. Another one is given by $\{x_1\}\ 3\ \{x_2\}\ 5\ \{x_3, x_4\}$ resp. $[p]\,3\,[q]\,5\,[r,r]$.
On the other hand, $\{x_1\}\ 3\ \{x_2\}\ 3\ \{x_3, x_4\}$ is not a flattening of the condition of ϕ.

LEMMA 14.4. *For a constraint* ϕ *and a configuration* c, *we have* $c \models \phi$ *iff* $c \models \phi_{fl}$ *for some flattening* ϕ_{fl} *of* ϕ.

From Theorem 12.2, Lemma 14.3, and Lemma 14.4, we get the following.

LEMMA 14.5. *The set of constraints is a BQO* (*and therefore also a WQO*) *under entailment.*

Acknowledgements. I would like to thank Giorgio Delzanno, Othmane Rezine and Yunyun Zhu for many helpful comments.

REFERENCES

[1] P. A. ABDULLA, K. ČERĀNS, B. JONSSON, and Y.-K. TSAY, *General decidability theorems for infinite-state systems*, ***Proceedings of the LICS '96, 11th IEEE International Symposium on Logic in Computer Science***, 1996, pp. 313–321.

[2] ———, *Algorithmic analysis of programs with well quasi-ordered domains*, ***Information and Computation***, vol. 160 (2000), pp. 109–127.

[3] P. A. ABDULLA and G. DELZANNO, *On the coverability problem for constrained multiset rewriting*, ***Proceedings of the AVIS'06, 5th International Workshop on on Automated Verification of Infinite-State Systems***, 2006.

[4] P. A. ABDULLA and B. JONSSON, *Verifying programs with unreliable channels*, ***Proceedings of the LICS '93, 8th IEEE International Symposium on Logic in Computer Science***, 1993, pp. 160–170.

[5] ———, *Verifying programs with unreliable channels*, ***Information and Computation***, vol. 127 (1996), no. 2, pp. 91–101.

[6] ———, *Verifying networks of timed processes*, ***Proceedings of the TACAS '98, 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*** (Bernhard Steffen, editor), Lecture Notes in Computer Science, vol. 1384, 1998, pp. 298–312.

[7] ———, *Ensuring completeness of symbolic verification methods for infinite-state systems*, ***Theoretical Computer Science***, vol. 256 (2001), pp. 145–167.

[8] ———, *Model checking of systems with many identical timed processes*, ***Theoretical Computer Science***, vol. 290 (2003), no. 1, pp. 241–264.

[9] P. A. ABDULLA and A. NYLÉN, *Better is better than well: On efficient verification of infinite-state systems*, ***Proceedings of the LICS '00, 16th IEEE International Symposium on Logic in Computer Science***, 2000, pp. 132–140.

[10] ———, *Timed Petri nets and BQOs*, ***Proceedings on the ICATPN'2001: 22nd International Conference on Application and Theory of Petri Nets***, Lecture Notes in Computer Science, vol. 2075, 2001, pp. 53–70.

[11] R. ALUR, C. COURCOUBETIS, and D. DILL, *Model-checking for real-time systems*, ***Proceedings of the LICS '90, 5th IEEE International Symposium on Logic in Computer Science***, 1990, pp. 414–425.

[12] R. ALUR and D. DILL, *A theory of timed automata*, ***Theoretical Computer Science***, vol. 126 (1994), pp. 183–235.

[13] K. BARTLETT, R. SCANTLEBURY, and P. WILKINSON, *A note on reliable full-duplex transmissions over half duplex lines*, ***Communications of the ACM***, vol. 12 (1969), no. 5, pp. 260–261.

[14] A. BIERE, A. CIMATTI, E. M. CLARKE, and Y. ZHU, *Symbolic model checking without BDDs*, ***Proceedings of the TACAS '99, 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems***, 1999.

[15] P. BOUYER, *Forward analysis of updatable timed automata*, ***Formal Methods in System Design***, vol. 24 (2004), no. 3, pp. 281–320.

[16] R. E. BRYANT, *Graph-based algorithms for boolean function manipulation*, ***IEEE Transactions on Computers***, vol. C-35 (1986), no. 8, pp. 677–691.

[17] J. R. BURCH, E. M. CLARKE, K. L. MCMILLAN, and D. L. DILL, *Symbolic model checking: 10^{20} states and beyond*, ***Information and Computation***, vol. 98 (1992), pp. 142–170.

[18] K. ČERĀNS, *Deciding properties of integral relational automata*, ***Proceedings of the ICALP '94, 21st International Colloquium on Automata, Languages, and Programming*** (Abiteboul and Shamir, editors), Lecture Notes in Computer Science, vol. 820, Springer Verlag, 1994, pp. 35–46.

[19] E. M. CLARKE, E. A. EMERSON, and A. P. SISTLA, *Automatic verification of finite-state concurrent systems using temporal logic specification*, ***ACM Transactions on Programming***

Languages and Systems, vol. 8 (1986), no. 2, pp. 244–263.

[20] T. CORMEN, C. LEISERSON, R. RIVEST, and C. STEIN, ***Introduction to algorithms***, MIT Press, McGraw-Hill, 2001.

[21] G. DELZANNO, *Automatic verification of cache coherence protocols*, ***Proceedings of the 12th International Conference on Computer Aided Verification*** (Emerson and Sistla, editors), Lecture Notes in Computer Science, vol. 1855, Springer Verlag, 2000, pp. 53–68.

[22] L. E. DICKSON, *Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors*, ***American Journal of Mathematics***, vol. 35 (1913), pp. 413–422.

[23] J. ESPARZA, A. FINKEL, and R. MAYR, *On the verification of broadcast protocols*, ***Proceedings of the LICS '99, 14th IEEE International Symposium on Logic in Computer Science***, 1999.

[24] A. FINKEL, *A generalization of the procedure of Karp and Miller to well structured transition systems*, ***Proceedings of the ICALP '87, 14th International Colloquium on Automata, Languages, and Programming***, 1987, pp. 499–508.

[25] ———, *Decidability of the termination problem for completely specified protocols*, ***Distributed Computing***, vol. 7 (1994), no. 3, pp. 129–135.

[26] A. FINKEL and PH. SCHNOEBELEN, *Well-structured transition systems everywhere!*, ***Theoretical Computer Science***, vol. 256 (2001), no. 1–2, pp. 63–92.

[27] G. HIGMAN, *Ordering by divisibility in abstract algebras*, ***Proceedings of the London Mathematical Society (3)***, vol. 2 (1952), no. 7, pp. 326–336.

[28] P. JANČAR, *ω^2-well quasi-orderings and reachability analysis*, Technical Report 158, Department of Computing Systems, Uppsala University, 1999.

[29] K. G. LARSEN, P. PETTERSSON, and W. YI, *UPPAAL in a nutshell*, ***International Journal on Software Tools for Technology Transfer***, vol. 1 (1997), no. 1–2.

[30] R. LAZIC, T. NEWCOMB, J. OUAKNINE, A. W. ROSCOE, and J. WORRELL, *Nets with tokens which carry data*, ***Fundamentals of Information***, (2008), no. 3, pp. 251–274.

[31] A. MARCONE, *Fine and axiomatic analysis of the quasi-orderings on $\mathcal{P}(q)$*, Technical Report 17/99/RR, Rapporto di Ricerca del Dipartimento di Matematica e Informatica dell'Università di Udine, 1999.

[32] K. L. MCMILLAN, ***Symbolic model checking***, Kluwer Academic Publishers, 1993.

[33] E. C. MILNER, *Basic wqo- and bqo-theory*, ***Graphs and orders*** (I. Rival, editor), D. Reidel Publishing Company, 1985, pp. 487–502.

[34] P. NIEBERT, M. MAHFOUDH, E. ASARIN, M. BOZGA, N. JAIN, and O. MALER, *Verification of timed automata via satisfiability checking*, ***FTRTFT'02, 7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems***, vol. 2469, 2002, pp. 225–243.

[35] M. POUZET, *Applications of well quasi-orderings and better quasi-orderings*, ***Graphs and orders*** (I. Rival, editor), D. Reidel Publishing Company, 1985, pp. 503–519.

[36] J. P. QUEILLE and J. SIFAKIS, *Specification and verification of concurrent systems in Cesar*, ***5th International Symposium on Programming, Turin***, Lecture Notes in Computer Science, vol. 137, Springer Verlag, 1982, pp. 337–352.

[37] P. REVESZ, ***Introduction to constraint databases***, Springer, 2002.

[38] S. YOVINE, *Kronos: A verification tool for real-time systems*, ***International Journal on Software Tools for Technology Transfer***, vol. 1 (1997), no. 1–2.

DEPARTMENT OF INFORMATION TECHNOLOGY
UPPSALA UNIVERSITY
P.O. BOX 337
751 05 UPPSALA, SWEDEN
E-mail: parosh@it.uu.se