Model Checking with Abstraction Refinement for Well-Structured Systems

Master Thesis

Rayna Dimitrova

May 2006

supervised by:
Prof. Dr. Andreas Podelski

second referee:
Prof. Bernd Finkbeiner

Universität des Saarlandes
Fachbereich 6.2 - Informatik

I hereby declare that this thesis is entirely my own work except where otherwise indicated. I have used only the resources given in the list of references.

Saarbrücken, 09 May 2006

Rayna Dimitrova

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Andreas Podelski for the interesting thesis subject, for his guidance and support. I would also like to thank everyone at MPII for providing an excellent environment conductive to research. Very special thanks to my parents for their constant support and encouragement.

Abstract

Abstraction plays an important role in the verification of infinite-state systems. One of the most promising and popular abstraction techniques is *predicate abstraction*. The right abstraction, i.e. the one that is sufficiently precise to prove or disprove the property under consideration, is automatically constructed by iterative *abstraction refinement*. The abstract-check-refine loop is not guaranteed to terminate in general. This results in the construction of semi-algorithms that may not terminate on some inputs.

For the class of *well-structured transition systems*, a large class of infinite-state systems, general decidability results hold. These are transition systems equipped with a well-quasi ordering on the set of states which is compatible with the transition relation. In particular *coverability*, i.e. reachability of an upward-closed set, is known to be decidable for this class of systems.

In this work we study the verification of well-structured systems w.r.t. the coverability property by means of predicate abstraction and refinement. We investigate the conditions under which the abstract-check-refine loop is guaranteed to terminate on instances of this class, provide a model checking method based on predicate abstraction and abstraction refinement and prove its completeness for this class of systems.

Contents

Chapter 1

Introduction

1.1 Software Model Checking, Predicate Abstraction and Refinement

Over the last few decades there has been an increasing research effort directed towards the automatic verification of programs. *Model checking* has emerged as a promising and powerful approach to automatic verification of finite-state systems. It is essentially exhaustive exploration of the system's state space and hence its application is limited to finite-state systems. However, many of the computational models used in practice (like channel systems) are infinite-state, i.e. variables range over unbounded domains. There are two important approaches to applying this technique to infinite-state systems. The first one is basically reduction of the system to an "equivalent" finite-state system and then exploring the resulting finite quotient space. The second approach gives rise to the so called symbolic methods where the infinite state space is explored directly. This is accomplished by manipulation of a data structure the members of which represent possibly infinite sets of states. Most often these are formulas of a fixed formalism (usually some restricted class of first order formulas over the algebraic structure over which the program computes (e.g. integers or reals)). The formulas are boolean combinations of atoms from an infinite set of atomic predicates $\mathcal{AP}$ and hence the resulting formula space is infinite.

For model checking we are interested in operations that for a given formula φ compute the formula denoting the set of predecessors or the set of successors of the states in the denotation of φ. Since the formula space is

infinite, a fixpoint iteration of such operator is in general not guaranteed to converge. *Abstraction* is used to ameliorate this problem by mapping the infinite domain into a finite one. Abstraction is used also in the case of finite-state systems for overcoming the state explosion problem. The abstraction of a program amounts to constructing a smaller abstract program in a manner which ensures that the property holds for the original system if it holds for the abstract one. A promising technique for automatic abstraction is *predicate abstraction*. It is used to prove correctness or incorrectness of a program with respect to a given specification on the basis of partial information about the set of reachable states of the system. Given a system $\mathcal{S}$ and a finite set of atomic predicates P, abstract interpretation consists in analyzing an abstraction of $\mathcal{S}$ relative to P. Automatic methods for generating predicates are usually based on the analysis of false positives. These are error reports for the abstract system, that do not have counterparts in the concrete system. The set of atoms is constructed iteratively starting with the set of atoms which appear in the specification of the property and *refining* it to eliminate false positives.

1.2 Motivation

The verification of a program w.r.t. a given property is in general an undecidable problem. However, undecidability is worked around by considering special classes of systems and restricting the type of properties to be checked. Here we are interested in *reachability*, that is whether a certain state or set of states is reachable (not reachable). Verification of other safety properties can be reduced to the reachability problem. Usually the goal is to check that a given set of error states is not reachable in any possible computation of the system. This can be done by either computing (an overapproximation of) the set of all states that are reachable from an initial state and proving that its intersection with the set of error states is empty (this is *forward* reachability analysis) or computing (an overapproximation of) the set of all states that can reach an error state and proving the emptiness of its intersection with the set of initial states (this is *backward* reachability analysis).

For some classes of infinite-state systems, such as systems with finite bisimulation/simulation quotient and systems with finite trace-equivalence it is known that reachability is decidable. Moreover, there exist forward procedures based on predicate abstraction for deciding reachability. A more

general class of systems, which includes many models used in practice, is the class of *well-structured transition systems* (*WSTS*). These are infinite-state systems equipped with a *well-quasi ordering* between states that satisfies certain compatibility property w.r.t. the transition relation. It is known that reachability is decidable for WSTS, provided that the set of error states is *upward-closed*. A symbolic backward algorithm for this class was proposed in [3]. The problem is that backward reachability analysis is often inefficient in practice; forward exploration of the state space is much more efficient. This is because forward state traversal can ensure that only parts of the state space that are reachable from an initial state and relevant for the satisfaction or violation of the specification are explored. For this reason, most of the existing tools construct an overapproximation of the set of reachable states in a forward manner. Hence, we are interested in obtaining a forward procedure employing predicate abstraction and refinement and proving its termination on WSTS and upward-closed sets of error states.

The termination of an abstraction refinement procedure depends on the heuristic used for selecting new predicates. If the set of atomic predicates is enumerable, then each time we can add a new predicate to the set of predicates and try predicate abstraction. In this case the procedure terminates if and only if there exists a safe invariant for the program which is expressible over $\mathcal{AP}$. However, this approach is not practical. Refinement based on analysis of abstract counterexamples is more efficient, but it often suffers from divergence. There are various completeness results for different predicate selection heuristics. In [4] abstract iteration with backward abstraction refinement was proven to be complete relative to a method for backward reachability analysis based on widening that is guided by an unrealistic oracle. A procedure for forward abstract iteration with backward counterexample-guided predicate refinement that is guaranteed to terminate for systems with finite trace-equivalence was presented in [13]. More recently, a practical and complete heuristic for predicate refinement based on interpolation was introduced in [16].

1.3 Contribution

We study the problem of applying predicate abstraction to WSTS. It is clear that refinement of the abstraction is needed in order to obtain a complete forward algorithm. We investigate the typical heuristic for predicate selec-

tion based on counterexample analysis and the conditions under which the refinement loop is guaranteed to converge. We present two instantiations of a standard schema for forward abstract iteration with backward predicate refinement. These instantiations result into two procedures for checking reachability. We prove that the first one is guaranteed to terminate on every member of the class of WSTS provided that the set of error states is upward-closed. For the second procedure we show its completeness for a less general class of systems - the class of WSLTS again under the assumption that the set of error states is upward-closed.

1.4 Outline

This thesis is organized as follows.

In **Chapter 2** we give the definitions of basic notions such as labeled transition systems and well-structured transition systems. We state some properties that are used in the subsequent chapters. We fix the framework of the symbolic representation and relate programs with their semantics given as transition systems.

In **Chapter 3** we give a schema of an algorithm for checking reachability by forward abstract interpretation and abstraction refinement and prove its soundness.

In **Chapter 4** we give an instantiation of the schema from the previous chapter and prove its termination for WSTS.

In **Chapter 5** we discuss the class of WSLTS. We look at a second instantiation of the schema that results in a more efficient algorithm, the termination of which we prove for instances of this class.

In **Chapter 6** we discuss possible optimizations of the proposed algorithm which might lead to losing the termination guarantee.

In **Chapter 7** we look at two families of WSTS and study the conditions under which the proposed method is applicable for them.

In **Chapter 8** we give a short overview of the classification of transition systems and the completeness results for each class. We discuss the differences between our approach and the other algorithms for WSTS.

In **Chapter 9** we give a concluding overview.

Chapter 2

Well-Structured Transition Systems

In this chapter we introduce the notions of (labeled) transition system, well-quasi ordering, and well-structured transition system. Then we look at symbolic representation of transition systems as programs and finite representation of possibly infinite sets of states as formulas over a set of atomic predicates.

2.1 Labeled Transition Systems

A system is represented by its set of states, a set of initial states, a set of labels and a transition relation.

Definition 2.1. A *(labeled) transition system* $\mathcal{S}$ is a tuple $\langle S, I, \mathcal{C}, \delta \rangle$, where

- S is a (possibly) infinite set of states,
- $I \subseteq S$ is a set of initial states,
- $\mathcal{C}$ is a finite set of labels,
- $\delta \subseteq S \times \mathcal{C} \times S$ is a transition relation.

Let $\langle S, I, \mathcal{C}, \delta \rangle$ be a labeled transition system.

Notation
Let $s_1, s_2 \in S$ be states, $c \in \mathcal{C}$ be a label and $\sigma = c^1 \dots c^m \in \mathcal{C}^*$ be a sequence of labels.

1. We write
   - $s_1 \xrightarrow{c} s_2$ if $(s_1, c, s_2) \in \delta$,
   - $s_1 \xrightarrow{\sigma} s_2$ if there exist states $t_0, t_1, \dots, t_m$, such that $t_0 = s_1$, $t_m = s_2$ and $t_0 \xrightarrow{c^1} t_1$, $t_1 \xrightarrow{c^2} t_2, \dots, t_{m-1} \xrightarrow{c^m} t_m$,
   - $s_1 \rightarrow s_2$ if there exists $c \in \mathcal{C}$, such that $s_1 \xrightarrow{c} s_2$,
   - $\rightarrow^*$ for the reflexive and transitive closure of $\rightarrow$.
2. With $\sigma[i, j)$, where $1 \leq i \leq m+1$, $1 \leq j \leq m+1$ and $i \leq j$, we denote the subsequence $c^i \dots c^{j-1}$ of σ . If $i = j$ then $\sigma[i, j)$ is the empty sequence.
3. The length of σ is $|\sigma|$.

With the given transition system $\mathcal{S}$ we associate transformer functions. The operator post_c maps a set of states to the set of all successor states under the transition $\xrightarrow{c}$ and the operator pre_c maps a set of states to the set of all predecessors under the transition $\xrightarrow{c}$.

Definition 2.2 (**Transformer functions**)**.** For $A \subseteq S$, $c \in \mathcal{C}$ and $\sigma \in \mathcal{C}^*$ we define

$$\mathsf{post}_c(A) = \{s' | \exists s \in A : \ s \xrightarrow{c} s'\}$$

$$\mathsf{pre}_c(A) = \{s | \exists s' \in A : \ s \xrightarrow{c} s'\}$$

$$\mathsf{post}(A) = \bigcup_{c \in \mathcal{C}} \mathsf{post}_c(A) = \{s' | \exists s \in A : \ s \rightarrow s'\}$$

$$\mathsf{pre}(A) = \bigcup_{c \in \mathcal{C}} \mathsf{pre}_c(A) = \{s | \exists s' \in A : \ s \rightarrow s'\}$$

$$\mathsf{post}_\sigma(A) = \{s' | \exists s \in A : \ s \xrightarrow{\sigma} s'\}$$

$$\mathsf{pre}_\sigma(A) = \{s | \exists s' \in A : \ s \xrightarrow{\sigma} s'\}$$

All of the operators defined above are monotone w.r.t. the subset relation.

2.2 Well-quasi Ordered Sets

We start with the definition of well-quasi ordering. For a set S equipped with such an ordering we give the definitions of upward- and downward-closed sets in S. Then we provide some results from the theory of well-quasi orderings.

Definition 2.3 (**Preorder**)**.** A *preorder* $\preceq$ is a reflexive and transitive binary relation on a set S.

Definition 2.4 (**Well-quasi ordering**)**.** We say that the preorder $\preceq$ on a set S is a *well-quasi ordering* if for every infinite sequence $s_0, s_1, s_2, \ldots$ of elements of S, there exist indices $0 \leq i < j$ such that $s_i \preceq s_j$.

Example 2.1 (**Well-quasi orderings**)**.**

- $(\mathbb{N}, \leq)$ - the set of natural numbers $\mathbb{N}$ with the standard ordering $\leq$ is well-quasi ordered.
- $(\mathbb{N}^k, \leq)$ - the set of vectors of k natural numbers with component-wise ordering is well-quasi ordered according to Dickson's lemma [7].
- $(\Sigma^*, \preceq)$ - the set of finite words over some alphabet Σ together with the subword relation $\preceq$ is well-quasi ordered in the case when Σ is finite according to Higman's lemma [15].

Let $(S, \preceq)$ be a well-quasi ordered set.

Definition 2.5 (**Upward-closed set**)**.** A set $A \subseteq S$ is said to be *upward-closed* if $s \in A$, $t \in S$ and $s \preceq t$ imply $t \in A$.

Upward-closed sets have the useful property that they can be represented by a finite set of minimal elements. Since in our setting we assume a given symbolic representation of sets of states, this fact is not of much interest to us.

Example 2.2 (**Upward-closed set**)**.**
Consider $(\mathbb{N}^2, \leq)$. The set $A = \{(x, y) \in \mathbb{N}^2 | x \geq 0 \wedge y \geq 6\}$ is upward-closed.

Definition 2.6 (**Minor set**)**.** Let $A \subseteq S$. A set $M \subseteq A$ is a *minor set* of A if

- for every $s \in A$ there exists $s' \in M$, such that $s' \preceq s$ and

- for any distinct $s, t \in M$ we have $s \not\preceq t$.

Lemma 2.7 (From [1]). *Every $A \subseteq S$ has at least one finite minor set.*

Thus, each upward-closed set can be represented by a finite minor set.

Definition 2.8 (**Downward-closed set**). A set $A \subseteq S$ is said to be *downward-closed* if $s \in A$, $t \in S$ and $t \preceq s$ imply $t \in A$.

Definition 2.9. Let $A \subseteq S$. Then

- the set $A{\uparrow} = \{s \in S | \exists s' \in A : s' \preceq s\}$ is the *upward-closure* of A,
- the set $A{\downarrow} = \{s \in S | \exists s' \in A : s \preceq s'\}$ is the *downward-closure* of A.

Example 2.3 (**Upward-closure**).
Consider $(\mathbb{N}^2, \leq)$. Let $A = \{(0,5), (3,2), (1,5)\}$.
Then $A{\uparrow} = \{(x,y) \in \mathbb{N}^2 | (x \geq 0 \wedge y \geq 5) \vee (x \geq 3 \wedge y \geq 2)\}$ and $\{(0,5), (3,2)\}$ is a minor set of $A{\uparrow}$.

Proposition 2.10. *For $A \subseteq S$ it holds that $A \subseteq A{\uparrow}$ and $A \subseteq A{\downarrow}$.*

Proposition 2.11. *The operator $\uparrow$ distributes over union.*

The following two lemmas formalize the facts that every infinite increasing sequence (according to $\subseteq$) of upward-closed sets and every infinite decreasing sequence of downward-closed sets eventually stabilize. In the proof of the completeness of the algorithm we present in Chapter 4 we make use of the fact that there is no strictly increasing sequence of upward-closed sets of states in a WSTS.

Lemma 2.12 (From [1]). *Let $(S, \preceq)$ be a well-quasi ordered set and $U_0, U_1, \ldots$ be an infinite sequence of upward-closed sets, such that $\forall i \geq 0 : U_i \subseteq U_{i+1}$. Then $\exists j \geq 0\, \forall k \geq j : U_k = U_j$.*

Lemma 2.13 (From [1]). *Let $(S, \preceq)$ be a well-quasi ordered set and $D_0, D_1, \ldots$ be an infinite sequence of downward-closed sets, such that $\forall i \geq 0 : D_i \supseteq D_{i+1}$. Then $\exists j \geq 0\, \forall k \geq j : D_k = D_j$.*

2.3 Well-Structured Transition Systems

For WSTS the existence of a well-quasi ordering over the infinite set of states ensures the termination of several algorithmic methods. WSTS are an abstract generalization of several families of transition systems and allow for general decidability results that can be applied to the specific classes of systems. The definition of WSTS was proposed by Finkel [8]. Independently, another definition was proposed by Abdulla *et al.* [1]. Here we follow the conceptual framework from [9]. The concept of WSTS defined there is a generalization of the earlier definitions. It defines a more general class of systems than the class from [1], to which we refer as *well-structured labeled transition systems*.

We start with the definitions of three notions of *compatibility* of a preorder on the set of states of the system with the transition relation. Let $\mathcal{S} = \langle S, I, \mathcal{C}, \delta \rangle$ be a labeled transition system and $\preceq \subseteq S \times S$ be a preorder between states. The first two conditions: *compatibility* and *strong compatibility* do not take into account the labels of the transitions, i.e. this is as if we consider transition systems without labels.

Definition 2.14 (**Compatibility**)**.** We say that $(\mathcal{S}, \preceq)$ satisfies *(upward) compatibility* if for all s_1, s_2, t_1 such that $s_1 \preceq t_1$ and $s_1 \rightarrow s_2$, there exists a state t_2 such that $s_2 \preceq t_2$ and $t_1 \rightarrow^* t_2$.

Definition 2.15 (**Strong compatibility**)**.** We say that $(\mathcal{S}, \preceq)$ satisfies *strong compatibility* if for all s_1, s_2, t_1 such that $s_1 \preceq t_1$ and $s_1 \rightarrow s_2$, there exists a state t_2 such that $s_2 \preceq t_2$ and $t_1 \rightarrow t_2$.

Since Abdulla was interested in the problem of simulation of infinite-state systems with finite-state systems, his definition included also labels on the transitions. For labeled transition systems the notion of compatibility is restricted so that it preserves the labels.

Definition 2.16 (**Strong compatibility w.r.t. labels of transitions**)**.** We say that $(\mathcal{S}, \preceq)$ satisfies *strong compatibility w.r.t. the labels of the transitions* if for all s_1, s_2, t_1 such that $s_1 \preceq t_1$ and $s_1 \xrightarrow{c} s_2$ for some $c \in \mathcal{C}$, there exists a state t_2, such that $s_2 \preceq t_2$ and $t_1 \xrightarrow{c} t_2$.

It is clear from the definitions that strong compatibility w.r.t. labels of transitions implies strong compatibility and strong compatibility implies compatibility. According to these three notions of compatibility we obtain the following three definitions.

Definition 2.17 (**WSTS**)**.** A transition system $\mathcal{S} = \langle S, I, \mathcal{C}, \delta \rangle$ together with a preorder $\preceq \subseteq S \times S$ between states is called *well-structured transition system* (*WSTS*) if the following conditions hold:

- $\preceq$ is a well-quasi ordering,
- ($\mathcal{S}$,$\preceq$) satisfies upward compatibility.

Definition 2.18 (**WSTS with strong compatibility**)**.** A transition system $\mathcal{S} = \langle S, I, \mathcal{C}, \delta \rangle$ together with a preorder $\preceq \subseteq S \times S$ between states is called a *well-structured transition system with strong compatibility* if the following conditions hold:

- $\preceq$ is a well-quasi ordering,
- ($\mathcal{S}$,$\preceq$) satisfies strong compatibility.

Definition 2.19 (**WSLTS**)**.** A labeled transition system $\mathcal{S} = \langle S, I, \mathcal{C}, \delta \rangle$ together with a preorder $\preceq \subseteq S \times S$ between states is called a *well-structured labeled transition system with strong compatibility* (*WSLTS*) if the following conditions hold:

- $\preceq$ is a well-quasi order,
- ($\mathcal{S}$,$\preceq$) satisfies strong compatibility w.r.t the labels of the transitions.

It is clear that if $\mathcal{S}$ is a WSTS with strong compatibility, it is also a WSTS. The converse is not true - the class of WSTS is strictly more general than the class of WSTS with strong compatibility.

For a transition system $\mathcal{S}$ and a well-quasi ordering $\preceq$ on the set of states, $\uparrow$ is the operator that maps each set of states to its upward-closure. The operator ucpre maps a set of states to the upward-closure of the set of predecessors, it is defined as the composition of the operators pre and $\uparrow$. We will use this operator in the analysis of the abstract counterexamples and for determining the set of refinement predicates.

Definition 2.20 (**ucpre**)**.** Let $A \subseteq S$. Let $c \in \mathcal{C}$ and $\sigma = c^1 \dots c^m \in \mathcal{C}^*$. We define:

$$\mathsf{ucpre}_c(A) = (\mathsf{pre}_c(A))\uparrow.$$

$$\mathsf{ucpre}(A) = \mathsf{pre}(A)\uparrow$$

$$\mathsf{ucpre}_\sigma(A) = \mathsf{ucpre}_{c^1}(\dots \mathsf{ucpre}_{c^m}(A) \dots)$$

Since $\uparrow$ distributes over union we have that $\mathsf{ucpre}(A) = \mathsf{pre}(A)\uparrow = \bigcup_{c \in \mathcal{C}} \mathsf{ucpre}_c(A)$.

2.4 Symbolic Representation of Labeled Transition Systems

In this section we describe the symbolic representation of transition systems and sets of states.

Let $\mathcal{V} = \mathcal{X} \cup \mathcal{X}'$ be a countable infinite set of variables where each variable in $\mathcal{X}$ occurs together with its primed version which is in $\mathcal{X}'$. Let $\mathcal{AP}$ be a fixed infinite set of *atomic formulas* (we call them *atoms*, *atomic predicates* or just *predicates*) over the variables in $\mathcal{V}$ where each atom appears with an atom representing its negated version. Formulas are constructed from atoms using the boolean connectives conjunction and disjunction. Let $\mathcal{B}(\mathcal{AP})$ be the closure of $\mathcal{AP}$ under $\wedge$ and $\vee$. The set of atoms that appear in a formula φ we denote with $\mathsf{atoms}(\varphi)$ and the set of the negations of these atoms with $\mathsf{atoms}(\neg\varphi)$. We define a preorder $\leq$ between the elements of $\mathcal{B}(\mathcal{AP})$ in the following way.

Definition 2.21 (**Order relation** $\leq$)**.**
For two formulas φ_1 and φ_2, $\varphi_1 \leq \varphi_2$ iff the implication $\varphi_1 \Rightarrow \varphi_2$ is valid. If $\varphi_1 \leq \varphi_2$ and $\varphi_2 \leq \varphi_1$ we write $\varphi_1 \equiv \varphi_2$. $\varphi_1 < \varphi_2$ is a shortcut for $\varphi_1 \leq \varphi_2$ and $\varphi_2 \not\leq \varphi_1$.

Note that in many cases validity of implication, and hence the ordering $\leq$ is not decidable.

2.4.1 Programs

A transition system actually describes the semantics of a program. Hence, transition systems are represented symbolically as programs. Here we give a formal definition of a program. We consider programs as sets of guarded commands. Actually, a program in a standard programming language can be easily translated into this form.

Definition 2.22 (**Program**)**.** A *program* is specified by a tuple $\langle X, \mathsf{init}, \delta \rangle$, where

- $X = \{x_1, \ldots x_n\} \subseteq \mathcal{X}$ is a finite set of program variables, including one or more program counters, each of which is associated with a domain,
- $\mathsf{init}(X)$ is a formula called *initial condition* that denotes the set of initial states,

- $\delta(X, X')$ is a formula describing the transition relation, where X' is the set of next-state(primed) variables.

Definition 2.23 (**Guarded command**)**.** We require that the formula δ representing the transition relation is a finite disjunction of *guarded commands*

$$\delta = c_1 \vee c_2 \vee \ldots \vee c_r$$

where each command c_i is of the form:

$$c_i : g_i(X) \wedge x_1' = e_1^i(X) \wedge \ldots \wedge x_n' = e_n^i(X).$$

g_i is the guard of the command; the transition is enabled if the guard is satisfied. The other conjuncts are the updates of the variables.

With the program $\mathcal{S} = \langle X, \mathsf{init}, \delta \rangle$ we associate a transition system $\mathcal{S} = \langle S, I, \mathcal{C}, \delta \rangle$ that describes its semantics, defined as follows:

- S consists of all possible valuations of the program variables X; it is the cartesian product of the domains of the variables in X,
- $I = \{s \in S | s \models \mathsf{init}\}$,
- $\mathcal{C} = \{c_1, c_2, \ldots, c_r\}$,
- $(s_1, c, s_2) \in \delta$ iff $c[s_1/X, s_2/X'] \equiv \mathsf{true}$.

In the above definition we used the following notation.

- With $\varphi[s/X]$ we denote the value of φ where each variable in X is evaluated with the corresponding value from s.
- If a state s satisfies a formula φ, i.e. if $\varphi[s/X] \equiv \mathsf{true}$, we write $s \models \varphi$.
- For a formula φ, $[\![\varphi]\!]$ is the set of states denoted by this formula, i.e. the states that satisfy φ. Then, provided that entailment is decidable, $\varphi_1 \leq \varphi_2$ iff $[\![\varphi_1]\!] \subseteq [\![\varphi_2]\!]$, and $\varphi_1 \equiv \varphi_2$ iff $[\![\varphi_1]\!] = [\![\varphi_2]\!]$.

Example 2.4 (**Simple transition system**)**.** The set of guarded commands of a simple program is given in Table 2.1. The set of initial states is denoted by $\mathsf{init} = (pc = l_1)$. The domain of x is $\mathbb{N}$. The system is given on Figure 2.1.

	Guard	Updates
c_1	$pc = l_1$	$pc' = l_2, x' = 0$
c_2	$pc = l_2 \wedge x = 0$	$pc' = pc, x' = x$
c_3	$pc = l_2 \wedge x > 0$	$pc' = l_3, x' = x$

Table 2.1: Guarded commands of the simple program

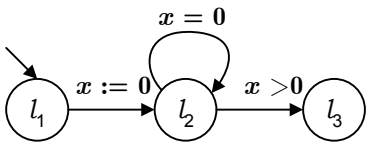


Figure 2.1: Simple transition system

As in the symbolic algorithm sets of states are represented by formulas, we need to translate the transformer functions defined for sets to operators on formulas. This we do as follows.

Definition 2.24 (**Predicate transformers**).
Let $c \in \mathcal{C}$ and $\sigma = c^1, \ldots, c^m \in \mathcal{C}^*$.

$$\mathsf{post}_{c_i}(\varphi) = (\exists X.\ \varphi(X) \wedge g_i(X) \wedge x_1' = e_1^i(X) \wedge \ldots \wedge x_n' = e_n^i(X))[X/X']$$

$$\mathsf{pre}_{c_i}(\varphi) = g_i(X) \wedge \varphi[e_1^i(X), \ldots, e_n^i(X)/x_1, \ldots, x_n]$$

$$\mathsf{post}(\varphi) = \bigvee_{c \in \mathcal{C}} \mathsf{post}_c(\varphi)$$

$$\mathsf{pre}(\varphi) = \bigvee_{c \in \mathcal{C}} \mathsf{pre}_c(\varphi)$$

$$\mathsf{post}_\sigma(\varphi) = \mathsf{post}_{c^m}(\ldots \mathsf{post}_{c^1}(\varphi)\ldots)$$

$$\mathsf{pre}_\sigma(\varphi) = \mathsf{pre}_{c^1}(\ldots \mathsf{pre}_{c^m}(\varphi)\ldots)$$

The definitions conform with the ones on sets in the sense that they reflect the fact that a formula φ denotes a set of states. Thus,

- $[\![\mathsf{post}_c(\varphi)]\!] = \mathsf{post}_c([\![\varphi]\!])$
- $[\![\mathsf{pre}_c(\varphi)]\!] = \mathsf{pre}_c([\![\varphi]\!])$
- $[\![\mathsf{post}(\varphi)]\!] = \mathsf{post}([\![\varphi]\!])$

- $[\![\mathsf{pre}(\varphi)]\!] = \mathsf{pre}([\![\varphi]\!])$
- $[\![\mathsf{post}_\sigma(\varphi)]\!] = \mathsf{post}_\sigma([\![\varphi]\!])$
- $[\![\mathsf{pre}_\sigma(\varphi)]\!] = \mathsf{pre}_\sigma([\![\varphi]\!])$

Proposition 2.25. *For formulas φ and ψ, and $c \in \mathcal{C}$ it holds that*

$$\varphi \leq \neg \mathit{pre}_c(\psi) \text{ iff } \mathit{post}_c(\varphi) \leq \neg\psi.$$

Proof. Let $\varphi \leq \neg\mathsf{pre}_c(\psi)$. Let $s \in [\![\mathsf{post}_c(\varphi)]\!]$. Then there is a state $t \in [\![\varphi]\!]$ such that $t \xrightarrow{c} s$. We have that $t \notin [\![\mathsf{pre}_c(\psi)]\!]$ since $\varphi \leq \neg\mathsf{pre}_c(\psi)$. Assume that $s \in [\![\psi]\!]$. Then $t \in [\![\mathsf{pre}_c(\psi)]\!]$. This is a contradiction and hence $s \in [\![\neg\psi]\!]$.

Let $\mathsf{post}_c(\varphi) \leq \neg\psi$. Let $s \in [\![\varphi]\!]$. Assume that $s \in [\![\mathsf{pre}_c(\psi)]\!]$. Then Then there is a state $t \in [\![\psi]\!]$ such that $s \xrightarrow{c} t$. Hence, $t \in [\![\mathsf{post}_c(\varphi)]\!]$. Since $\mathsf{post}_c(\varphi) \leq \neg\psi$, $t \notin [\![\psi]\!]$. This is a contradiction and hence $s \in [\![\neg\mathsf{pre}_c(\psi)]\!]$. □

2.4.2 Iteration and Invariants

Let $\mathcal{S} = \langle X, \mathsf{init}, \delta \rangle$ be a program and the following formulas be given

unsafe, such that $[\![\mathsf{unsafe}]\!] = \mathsf{bad}$

safe, such that $[\![\mathsf{safe}]\!] = S \backslash \mathsf{bad}$

The formula unsafe denotes the set bad of *error states* and its negation safe denotes the set of *safe states*.

Definition 2.26 (**Reachability**)**.** We say that a state t is *reachable* from a state s if $s \rightarrow^* t$. For two sets of states I and B we say that B is *reachable* from I if for some states $s \in I$ and $t \in B$ t is reachable form s. The set B is reachable if it is reachable from the set of initial states.

In the case when $\mathcal{S}$ is a WSTS with ordering $\preceq$ and the set B is upward-closed, checking reachability amounts to the coverability problem as it is formulated in [10].

Definition 2.27 (**Coverability**)**.** The *coverability problem* is "Given an upward-closed set B, is B reachable from I, the set of initial states of $\mathcal{S}$?".

A program is said to be *correct*(*safe*) if no error state is reachable from an initial state. The least fixpoint of the operator pre, computed from the basis unsafe is

$$\mathsf{lfp}(\mathsf{pre},\mathsf{unsafe}) = \mathsf{lfp}(\lambda\varphi.\mathsf{unsafe} \vee \mathsf{pre}(\varphi)).$$

Similarly, the least fixpoint of the operator post above init is

$$\mathsf{lfp}(\mathsf{post},\mathsf{init}) = \mathsf{lfp}(\lambda\varphi.\mathsf{init} \vee \mathsf{post}(\varphi)).$$

Note that they might not be elements of the domain of formulas. In the sequel with $\mathsf{lfp}(\mathsf{pre},\mathsf{unsafe})$ we denote arbitrary formula φ such that $[\![\varphi]\!] = \bigcup_{i=0}^{\infty}[\![\mathsf{pre}^i(\mathsf{unsafe})]\!]$ if such exists. Similarly for $\mathsf{lfp}(\mathsf{post},\mathsf{init})$. The definition of correctness given above can be formalized as follows. The program $\mathcal{S}$ is safe if

$$\mathsf{lfp}(\mathsf{post},\mathsf{init}) \leq \mathsf{safe}$$

or

$$\mathsf{lfp}(\mathsf{pre},\mathsf{unsafe}) \leq \mathsf{nonInit},$$

where $\mathsf{nonInit}$ denotes the complement of the set of initial states.

Definition 2.28 (**Forward invariant**)**.** A *forward invariant* is a formula φ such that $\mathsf{init} \leq \varphi$ and $\mathsf{post}(\varphi) \leq \varphi$. If in addition $\varphi \leq \mathsf{safe}$ then φ is called *safe forward invariant*.

Proving the correctness of the program can be done by computing a forward invariant and checking whether it is safe. If we construct an over-approximation $\mathsf{post}^{\#}$ of the operator post and compute the least fixpoint $\mathsf{lfp}(\mathsf{post}^{\#},\mathsf{init})$, then we obtain a forward invariant. By proving its safety we can show that the system is correct.

2.5 Symbolic Representation of WSTS

Let $\preceq$ be a preorder over the set of states of the transition system $\mathcal{S} = \langle S, I, \mathcal{C}, \delta\rangle$ that corresponds to a program $\mathcal{S}$ and $(\mathcal{S}, \preceq)$ is a WSTS. We assume that $\preceq$ can be expressed over $\mathcal{AP}$, i.e. there is a formula $\varphi_{\preceq}(X', X)$ in $\mathcal{B}(\mathcal{AP})$, such that $s' \preceq s$ iff $\varphi_{\preceq}[s'/X', s/X]$.

Definition 2.29 (**Operator $\uparrow$ on formulas**)**.** Let $\varphi(X)$ be a formula. With $\varphi{\uparrow}(X)$ we denote the formula obtained by quantifier elimination from the formula $\exists X' : \varphi_{\preceq}(X', X) \wedge \varphi[X'/X]$.

The formula $\varphi{\uparrow}(X)$ is such that $[\![\varphi{\uparrow}]\!] = [\![\varphi]\!]{\uparrow}$.

Now we give the definitions of the operators ucpre_c, ucpre and ucpre_σ on formulas in a way that

- $[\![\mathsf{ucpre}_c(\varphi)]\!] = \mathsf{ucpre}_c([\![\varphi]\!])$ and
- $[\![\mathsf{ucpre}(\varphi)]\!] = \mathsf{ucpre}([\![\varphi]\!])$ and
- $[\![\mathsf{ucpre}_\sigma(\varphi)]\!] = \mathsf{ucpre}_\sigma([\![\varphi]\!])$.

Definition 2.30 (**ucpre on formulas**)**.** Let $c \in \mathcal{C}$, $\sigma = c^1 \dots c^m \in \mathcal{C}^*$ and φ be a formula. We define:

$$\mathsf{ucpre}_c(\varphi) = (\mathsf{pre}_c(\varphi)){\uparrow}$$

$$\mathsf{ucpre}(\varphi) = \bigvee\nolimits_{c \in \mathcal{C}} \mathsf{ucpre}_c(\varphi)$$

$$\mathsf{ucpre}_\sigma(\varphi) = \mathsf{ucpre}_{c^1}(\dots \mathsf{ucpre}_{c^m}(\varphi) \dots)$$

Proposition 2.31. *For every formula φ we have $\mathsf{pre}(\varphi) \leq \mathsf{ucpre}(\varphi)$.*

Proof. $[\![\mathsf{pre}(\varphi)]\!] \subseteq [\![\mathsf{pre}(\varphi)]\!]{\uparrow} = [\![\mathsf{pre}(\varphi){\uparrow}]\!] = [\![\mathsf{ucpre}(\varphi)]\!]$ □

The following lemma says that the set of all states that can reach an upward-closed set is also upward-closed.

Lemma 2.32. *If $[\![\mathsf{unsafe}]\!]$ is an upward-closed set, then $[\![\mathsf{lfp}(\mathsf{pre}, \mathsf{unsafe})]\!]$ is upward-closed.*

Proof. Let $s_1 \in [\![\mathsf{lfp}(\mathsf{pre}, \mathsf{unsafe})]\!]$ and $s_1 \preceq t_1$. We consider the two cases:

1. $s_1 \in [\![\mathsf{unsafe}]\!]$
   Since $[\![\mathsf{unsafe}]\!]$ is upward-closed, it holds that $t_1 \in [\![\mathsf{unsafe}]\!]$. Thus, $t_1 \in [\![\mathsf{lfp}(\mathsf{pre}, \mathsf{unsafe})]\!]$.
2. $s_1 \notin [\![\mathsf{unsafe}]\!]$
   Thus, since $s_1 \in [\![\mathsf{lfp}(\mathsf{pre}, \mathsf{unsafe})]\!]$, there exist $s_2, \dots, s_n$ such that $s_1 \to s_2 \to \dots \to s_{n-1} \to s_n$ and $s_n \in [\![\mathsf{unsafe}]\!]$. By the compatibility of $\preceq$, there exist $t_2, \dots, t_n$, such that $t_1 \to^* t_2 \to^* \dots \to^* t_{n-1} \to^* t_n$ and $s_n \preceq t_n$. Since $[\![\mathsf{unsafe}]\!]$ is upward-closed, it holds that $t_n \in [\![\mathsf{unsafe}]\!]$. Thus, $t_1 \in [\![\mathsf{lfp}(\mathsf{pre}, \mathsf{unsafe})]\!]$

□

Proposition 2.33. *For all $i \geq 0$ it holds that $\mathit{pre}^i(\mathit{unsafe}) \leq \mathit{ucpre}^i(\mathit{unsafe})$.*

Proof. We do induction on i.

- **Base case**
  $\mathsf{pre}^0(\mathsf{unsafe}) = \mathsf{unsafe} = \mathsf{ucpre}^0(\mathsf{unsafe})$

- **Induction step**
  $\mathsf{pre}^{i+1}(\mathsf{unsafe}) = \mathsf{pre}(\mathsf{pre}^i(\mathsf{unsafe})) \leq \mathsf{pre}(\mathsf{ucpre}^i(\mathsf{unsafe})) \leq \mathsf{ucpre}^{i+1}(\mathsf{unsafe})$

□

Proposition 2.34. *If $[\![\mathit{unsafe}]\!]$ is an upward-closed set, then for all $i \geq 0$ it holds that $\mathit{ucpre}^i(\mathit{unsafe}) \leq \mathit{lfp}(\mathit{pre}, \mathit{unsafe})$.*

Proof. We do induction on i.

- **Base case**
  $\mathsf{ucpre}^0(\mathsf{unsafe}) = \mathsf{unsafe} \leq \mathsf{lfp}(\mathsf{pre}, \mathsf{unsafe})$

- **Induction step**
  $\mathsf{ucpre}^{i+1}(\mathsf{unsafe}) = \mathsf{pre}(\mathsf{ucpre}^i(\mathsf{unsafe}))\uparrow$
  Let $s \in [\![\mathsf{pre}(\mathsf{ucpre}^i(\mathsf{unsafe}))\uparrow]\!]$. Then there exists t, such that $t \preceq s$ and $t \in [\![\mathsf{pre}(\mathsf{ucpre}^i(\mathsf{unsafe}))]\!]$. Thus, there exist t', such that $t \to t'$ and $t' \in [\![\mathsf{ucpre}^i(\mathsf{unsafe})]\!]$. There exist s', such that $s \to^* s'$ and $t' \preceq s'$. Since $[\![\mathsf{ucpre}^i(\mathsf{unsafe})]\!] \subseteq [\![\mathsf{lfp}(\mathsf{pre}, \mathsf{unsafe})]\!]$ and $[\![\mathsf{lfp}(\mathsf{pre}, \mathsf{unsafe})]\!]$ is upward-closed, it holds that $s' \in [\![\mathsf{lfp}(\mathsf{pre}, \mathsf{unsafe})]\!]$. Hence, $s \in [\![\mathsf{lfp}(\mathsf{pre}, \mathsf{unsafe})]\!]$.

□

The following lemma states that the set of states from which an upward-closed set $[\![\mathsf{unsafe}]\!]$ can be reached, is actually the set of states denoted by the least fixpoint of ucpre above unsafe. This fact ensures that backward exploration of the state space of a WSTS by iterating ucpre staring from an upward-closed set produces as a result exactly the set of all states from which an error state is reachable. However, we do not make use of it here in this exact way. For our further discussions it is important that the formula $\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe})$ is a backward invariant. Hence, if a set of predicates contains the negations of the atoms that appear in $\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe})$, then the abstraction constructed w.r.t. this set of predicates is precise enough to prove correctness of the abstract system, in case the concrete system is safe.

Lemma 2.35. *If* $[\![\mathit{unsafe}]\!]$ *is an upward-closed set, then* $[\![\mathit{lfp}(\mathit{pre}, \mathit{unsafe})]\!] = [\![\mathit{lfp}(\mathit{ucpre}, \mathit{unsafe})]\!]$.

Proof.

$$[\![\mathsf{lfp}(\mathsf{pre}, \mathsf{unsafe})]\!] = \bigcup_{i=0}^{\infty} [\![\mathsf{pre}^i(\mathsf{unsafe})]\!]$$

$$[\![\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe})]\!] = \bigcup_{i=0}^{\infty} [\![\mathsf{ucpre}^i(\mathsf{unsafe})]\!]$$

By Proposition 2.33, $[\![\mathsf{lfp}(\mathsf{pre}, \mathsf{unsafe})]\!] \subseteq [\![\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe})]\!]$.
By Proposition 2.34, $[\![\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe}) \subseteq [\![\mathsf{lfp}(\mathsf{pre}, \mathsf{unsafe})]\!]$. □

Chapter 3

Predicate Abstraction with Refinement

As explained in the previous chapter we are interested in computing a safe forward invariant for a given transition system or proving the program's incorrectness. We use predicate abstraction with respect to finite set of predicates P to map the infinite set of formulas $\mathcal{B}(\mathcal{AP})$ into a finite one. In the finite abstract domain the iterative computation of the fixpoint of the abstract post operator is guaranteed to converge. Whenever the approximation is too coarse, the abstraction is refined and the computation of the invariant is resumed.

3.1 Predicate Abstraction

In this section we define the abstract domain w.r.t. a finite set of predicates together with abstraction and concretization functions. Since the algorithm considers different abstract domains at the same time, we assign each formula in the abstract domain a finite set of atoms - the set of predicates that determines the corresponding abstract domain. These pairs of formulas and sets of atoms we call *regions*. Then we define the *abstract post operator* $\mathsf{post}^{\#}$ on regions, which is an overapproximation of the concrete post operator in the sense that $\mathsf{post}^{\#}((\varphi, P))$ is a superset of $\mathsf{post}(\varphi)$.

We fix a program $\mathcal{S} = \langle X, \mathsf{init}, \delta \rangle$ and a preorder $\preceq$ given by the formula $\varphi_{\preceq}$, which is a well-quasi ordering on the set of states of the corresponding transition system.

3.1.1 Concrete and Abstract Domain

The *concrete domain* is the infinite formula space $\mathcal{B}(\mathcal{AP})$, more formally the complete lattice $\langle \mathcal{B}(\mathcal{AP}), \leq, \mathsf{false}, \mathsf{true}, \vee, \wedge \rangle$. We assume that all formulas are in the form

$$\bigvee_{i \in I} \bigwedge_{j \in J_i} \varphi_{ij}$$

where φ_{ij} are atoms.

The abstract domain is parameterized by a finite set of atoms $P \subseteq \mathcal{AP}$. The *abstract domain* w.r.t. P is $\mathcal{L}(P)$, the finite free distributive complete lattice generated by the set of predicates P, with bottom element false and top element true (which are assumed to be elements of $\mathcal{L}(P)$) and the operators $\vee$ and $\wedge$. Each lattice element can be written in its disjunctive normal form. The partial order $\sqsubseteq$ on $\mathcal{L}(P)$ is defined as follows.

Definition 3.1 (**Order relation $\sqsubseteq$**)**.**

$$\bigvee_{i \in I} \bigwedge_{j \in J_i} \varphi_{ij} \sqsubseteq \bigvee_{k \in K} \bigwedge_{j \in J'_k} \varphi'_{kj} \text{ if } \forall i \in I\ \exists k \in K\ \{\varphi_{ij} \mid j \in J_i\} \supseteq \{\varphi'_{kj} \mid j \in J'_k\}$$

Note that according to this ordering predicates are pairwise incomparable. The abstract relation $\sqsubseteq$ implies the concrete one, but the converse is not true.

Proposition 3.2. *If $\varphi_1 \sqsubseteq \varphi_2$ then $\varphi_1 \leq \varphi_2$.*

3.1.2 Abstraction and Concretization Functions

We use the framework of abstract interpretation [6] to construct the *best abstraction* of post w.r.t. P. To this end we define abstraction and concretization functions that form a *Galois connection*. The *abstraction function* $\alpha_P : \mathcal{B}(\mathcal{AP}) \to \mathcal{L}(P)$ is defined as:

$$\alpha_P(\varphi) = \mu_{\sqsubseteq} \psi \in \mathcal{L}(P).\ \varphi \leq \psi.$$

It maps a formula φ to the smallest w.r.t. $\sqsubseteq$ formula φ' in $\mathcal{L}(P)$ that is larger than φ w.r.t. $\leq$. The *concretization function* γ_P is defined to be the identity.

Theorem 3.3. *The pair of functions α_P and γ_P is a Galois connection, i.e. for all $\varphi \in \mathcal{B}(\mathcal{AP})$ and $\psi \in \mathcal{L}(P)$ it holds that*

$$\varphi \leq \gamma_P(\psi) \text{ iff } \alpha_P(\varphi) \sqsubseteq \psi.$$

As a consequence we have that α_P and γ_P are monotone and $\varphi \leq \gamma_P(\alpha_P(\varphi))$. For $\psi \in P$ we often write ψ instead of $\gamma_P(\psi)$ using the fact that γ_P is the identity. In the same way for $\psi_1, \psi_2 \in \mathcal{L}(P)$ we write $\psi_1 \leq \psi_2$ when $\psi_1 \sqsubseteq \psi_2$ having in mind that $\sqsubseteq$ entails $\leq$.

Proposition 3.4. *If $\mathcal{L}(P)$ contains a formula that is equivalent to a formula φ, then the abstraction of φ is equivalent to φ, formally*

$$\gamma_P(\alpha_P(\varphi)) \equiv \varphi.$$

If the finite sets of atoms P and Q are such that $P \subseteq Q$, then for every formula φ it holds that $\gamma_Q(\alpha_Q(\varphi)) \leq \gamma_P(\alpha_P(\varphi))$, i.e. the more elements the set of predicates contains, the more precise is the abstraction.

3.1.3 The Region Structure

Definition 3.5 (**Region**)**.** A *region* is a pair (ψ, P) where P is a finite set of predicates, which we call *support predicates* and $\psi \in \mathcal{L}(P)$.

Definition 3.6 (**Operators on regions**)**.** Let $c \in \mathcal{C}$ and $\sigma = c^1 \dots c^m \in \mathcal{C}^*$. We define the following operators on regions:

$$\mathsf{post}^{\#}_c((\psi, P)) = (\alpha_P(\mathsf{post}_c(\psi)), P)$$

$$\mathsf{post}^{\#}((\psi, P)) = (\textstyle\bigvee_{c \in C} \alpha_P(\mathsf{post}_c(\psi)), P)$$

$$\mathsf{post}^{\#}_\sigma((\psi, P)) = \mathsf{post}^{\#}_{c^m}(\dots \mathsf{post}^{\#}_{c^1}((\psi, P)) \dots)$$

$$[\![(\psi, P)]\!] = [\![\psi]\!]$$

$$[(\psi, P)] = \psi.$$

3.1.4 Abstract $\mathsf{post}^{\#}_P$

Let $P \subseteq \mathcal{AP}$. The abstract post-operator w.r.t. P, $\mathsf{post}^{\#}_P : \mathcal{L}(P) \to \mathcal{L}(P)$ we define as

$$\mathsf{post}^{\#}_P = \alpha_P \circ \mathsf{post} \circ \gamma_P.$$

Then it holds that

$$\mathsf{post}^{\#}_P(\psi) = [\mathsf{post}^{\#}((\psi, P))].$$

3.2 Abstract Reachability Analysis

In this section we give a schema of a semi-algorithm that iteratively computes a safe invariant for a program $\mathcal{S}$ or proves its incorrectness. We show that if it terminates, then the result is correct. If it reports that the program is not correct then bad is reachable, and if it gives as result a formula, then this formula is a safe forward invariant, which means that the program is correct. This entails the partial correctness of every instantiation of this schema.

Here we give the steps of the algorithm in pseudocode and describe the procedure, explaining the basic performed operations.

3.2.1 Abstract Reachability Tree

The *abstract reachability tree* is a rooted directed tree where each node is labeled with a region and each edge is labeled with a guarded command. If a node $\mathbf{n}$ is labeled with (ψ, P), we write $\mathbf{n}{:}(\psi, P)$. The formula ψ is called the *reachable region* of the node.

The elements of $\mathcal{C}^*$, i.e. the finite sequences of guarded commands we call *traces*. A node $\mathbf{n}$ is characterized by the trace σ labeling the path from the root to $\mathbf{n}$. When there is a path in the tree from a node $\mathbf{m}$ to a node $\mathbf{n}$ the edges of which are labeled with the elements of the trace σ, we write $\mathbf{m} \xrightarrow{\sigma} \mathbf{n}$. We call a node $\mathbf{n}{:}(\psi, P)$

- an *error node* if $\psi \not\leq \mathsf{safe}$,
- *safe* if $\psi \leq \mathsf{safe}$.

The algorithm that we describe here is a standard symbolic forward abstract iteration algorithm with backward refinement of the abstraction. It is similar to the corresponding procedure from [13]. The schema is parameterized by:

- the procedure used to determine the *pivot node* (actually by the definition of the *error region* of a node) and
- by the procedure used to determine the *refinement predicates*,

i.e. by the way abstract counterexamples are analyzed. In the next two chapters we provide two instantiations of this schema and prove their termination for two particular classes of systems.

Require: Program $\mathcal{S}$, formula unsafe
  create root $\mathbf{r}$ labeled with (init, P_0); $F := \emptyset$; $L := \{\mathbf{r}\}$
  while $L \neq \emptyset$ **do**
    remove $\mathbf{n}$:(ψ, P) from L
    if $\psi \leq \mathsf{safe}$ **then**
      $\varphi := \bigvee_{\theta \in F} \theta$
      if $\psi \sqsubseteq \varphi$ **then**
        mark $\mathbf{n}$ as *covered*
      else
        for all command $c \in \mathcal{C}$ **do**
          $(\psi', P) := \mathsf{post}^{\#}_c((\psi, P))$
          if $\psi' \not\leq \mathsf{false}$ **then**
            construct a child $\mathbf{n}'$ of $\mathbf{n}$ with label (ψ', P)
            label the arc $\mathbf{n} \rightarrow \mathbf{n}'$ with c
            mark $\mathbf{n}'$ as *unprocessed*; add $\mathbf{n}'$ to L
            $F := F \cup \{\psi\}$
            mark $\mathbf{n}$ as *uncovered*
          end if
        end for
      end if
    else
      $\mathbf{m}$:$(\varphi, Q) := \mathsf{pivot}(\mathbf{n})$; $\sigma :=$ the trace from $\mathbf{m}$ to $\mathbf{n}$
      if $\mathbf{m}$==NULL **then**
        return NOT CORRECT
      else
        relabel $\mathbf{m}$ with $(\varphi, Q \cup \mathsf{predicates}(\sigma))$
        // $\mathbf{m}$ is refined with $\mathsf{predicates}(\sigma)$ w.r.t. the trace σ
        remove the subtrees starting at the children of $\mathbf{m}$
        change the mark of $\mathbf{m}$ to *unprocessed*; add $\mathbf{m}$ to L
        remove from F the formulas corresponding to
          the deleted nodes and $\mathbf{m}$
        mark as *unprocessed* and add to L
          all nodes marked as covered after $\mathbf{m}$ was processed
      end if
    end if
  end while
  return $\varphi := \bigvee_{\theta \in F} \theta$

Algorithm 1: Abstract Iteration with Abstraction Refinement

Definition 3.7 (**Error region**). The *error region* of a node $\mathbf{m}$ with reachable region φ for an error node $\mathbf{n}$ with $\mathbf{m} \xrightarrow{\sigma} \mathbf{n}$ is a formula χ, which satisfies the following two conditions:

- If $\varphi \leq \neg\chi$ then $[\![\varphi]\!] \cap [\![\mathsf{pre}_\sigma(\mathsf{unsafe})]\!] = \emptyset$, i.e. every σ-successor of a state in $[\![\varphi]\!]$ is in $[\![\mathsf{safe}]\!]$.

- If $\varphi \not\leq \neg\chi$ then $[\![\varphi]\!] \cap [\![\mathsf{lfp}(\mathsf{pre}, \mathsf{unsafe})]\!] \neq \emptyset$, i.e. there is a state in $[\![\varphi]\!]$, from which a state in bad is reachable.

3.2.2 The Algorithm Schema

The algorithm iteratively constructs an abstract reachability tree, by constructing a sequence of trees $T_0, T_1, \ldots$. The initial tree T_0 consists of a single node $\mathbf{r}$ labeled with (init, P_0), where $P_0 = \mathsf{atoms}(\neg\mathsf{unsafe}) \cup \mathsf{atoms}(\mathsf{init})$. At the $k+1$-th iteration of the loop the procedure constructs the tree T_{k+1} from the tree T_k. It maintains a list L of nodes to be explored and at each step processes a node from L until it becomes empty.

Each node in the current tree T_k has a *mark* which can be *unprocessed*, *covered* or *uncovered*. A node $\mathbf{n}{:}(\psi, P)$ is marked as

- *unprocessed* if it is in L,

- *covered* if it was processed in T_i with $i < k$ and found to be covered, i.e. if $\psi \sqsubseteq \varphi$, where
  $\varphi = \bigvee\{\varphi' | \exists \mathbf{n}'{:}(\varphi', P') \in T_i$ such that the mark of $\mathbf{n}'$ is *uncovered*$\}$

- *uncovered* if it was processed in T_i where $i < k$ and its successors were computed because $\psi \not\sqsubseteq \varphi$, where
  $\varphi = \bigvee\{\varphi' | \exists \mathbf{n}'{:}(\varphi', P') \in T_i$ such that the mark of $\mathbf{n}'$ is *uncovered*$\}$.

The tree T_{k+1} is constructed from the tree T_k by one of the following operations:

- a node from L is processed and found to be covered or

- a node from L is processed, it is not covered and so its children are added to the tree and to L or

- a node in T_k is refined and the other nodes in its subtree are deleted.

When a node is processed, it is checked whether it is safe or not. In the case when the node is safe, it is either discovered to be covered and its children are not generated (its subtree is guaranteed to be processed elsewhere in the tree) or is not covered in which case its children are generated and added to the abstract reachability tree and to L. If the node is not safe, then there is an abstract counterexample - a path from the root of the abstract reachability tree to an error node. This abstract counterexample is analyzed backwards, starting from the error node and finding the first node on the path from it to the root, such that the intersection of the reachable region and the error region of the node is empty. This node is called the *pivot node*. If no such node is found, the procedure terminates with the answer that the program is not correct. Otherwise, this node is *refined* by adding new atoms to its set of support predicates. The subtree below the pivot node is deleted and the construction is resumed from the pivot node onwards, where the abstraction is w.r.t. the new set of support predicates of the pivot node. In addition, all nodes that are marked as covered after the pivot node was processed should be marked as unprocessed and added to L since they might not be covered any more. The new predicates are such, that it is guaranteed that the same abstract counterexample does not appear again - the node that is reachable from the pivot node via the same trace is guaranteed to be safe.

The algorithm schema described informally above is given in pseudocode as Algorithm 1. If the algorithm terminates with a tree T, it either returns NOT CORRECT, or the formula

$$\varphi = \bigvee\{\varphi' | \exists \mathbf{n}' : (\varphi', P') \in T \text{ such that the mark of } \mathbf{n}' \text{ is } \textit{uncovered}\},$$

which is a forward safe invariant.

The following three propositions state some properties, that are satisfied by the regions labeling a node in some of the trees and a node in its subtree.

Proposition 3.8. *If* $\boldsymbol{n}:(\psi, P)$ *and* $\boldsymbol{m}:(\psi', P')$ *are nodes in some* T_i *such that* $\boldsymbol{m}$ *is in the subtree of* $\boldsymbol{n}$ *then* $P \subseteq P'$.

Proof. The proof goes by induction on the length l of the path between $\mathbf{n}$ and $\mathbf{m}$.

1. **Base case:**
   If l=0 then $\mathbf{m}$=$\mathbf{n}$ and so, $P = P'$.

2. **Induction step:**
   Let the statement hold for some length l and the length of the path from $\mathbf{n}$ to $\mathbf{m}$ be l+1. Let $\mathbf{r}$:(φ, Q) be the parent of $\mathbf{m}$. We consider the following two cases:

   (a) $\mathbf{m}$ has not been refined after it was added as a child of $\mathbf{r}$.
   Then $P' = Q$ and by induction hypothesis $P \subseteq Q$. Thus, $P \subseteq P'$.

   (b) $\mathbf{m}$ has been refined after it was added as a child of $\mathbf{r}$.
   Then $P' \supseteq Q$ and by induction hypothesis $P \subseteq Q$. Thus, $P \subseteq P'$.

□

Proposition 3.9. *If* $\boldsymbol{n}$*:*(ψ, P) *and* $\boldsymbol{m}$*:*(ψ', P') *are nodes in some* T_i *such that* $\boldsymbol{n} \xrightarrow{\sigma} \boldsymbol{m}$ *then* $\psi' \leq [\mathit{post}^{\#}_{\sigma}((\psi, P))]$.

Proof. The proof goes by induction on the length of σ.

1. **Base case:**
   The length of σ is 0, i.e. $\mathbf{m}$=$\mathbf{n}$. Then $\psi' = \psi = [\mathsf{post}^{\#}_{\sigma}((\psi, P))]$.

2. **Induction step:** Let σ have length $l + 1$, i.e. $\sigma = \sigma' * c$, where the length of σ' is l and $c \in \mathcal{C}$. Let $\mathbf{r}$:(φ, Q) be the parent of $\mathbf{m}$. Then $\mathbf{n} \xrightarrow{\sigma'} \mathbf{r}$. By induction hypothesis $\varphi \leq [\mathsf{post}^{\#}_{\sigma'}((\psi, P))]$. Since α_Q and post_c are monotone, $\alpha_Q(\mathsf{post}_c(\varphi)) \leq \alpha_Q(\mathsf{post}_c([\mathsf{post}^{\#}_{\sigma'}((\psi, P))]))$. Since $P \subseteq Q$, $\alpha_Q(\mathsf{post}_c([\mathsf{post}^{\#}_{\sigma'}((\psi, P))])) \leq \alpha_P(\mathsf{post}_c([\mathsf{post}^{\#}_{\sigma'}((\psi, P))]))$. We have that $\psi' = \alpha_Q(\mathsf{post}_c(\varphi)) \leq \alpha_P(\mathsf{post}_c([\mathsf{post}^{\#}_{\sigma'}((\psi, P))])) = [\mathsf{post}^{\#}_{\sigma' * c}((\psi, P))] = [\mathsf{post}^{\#}_{\sigma}((\psi, P))]$.

□

Proposition 3.10. *If* $\boldsymbol{n}$*:*(ψ, P) *and* $\boldsymbol{m}$*:*(ψ', P') *are nodes in some* T_i *such that* $\boldsymbol{n} \xrightarrow{\sigma} \boldsymbol{m}$ *then* $\mathit{post}_{\sigma}(\psi) \leq \psi'$.

Proof. The proof goes by induction on the length of σ.

1. **Base case:**
   The length of σ is 0, i.e. $\mathbf{m}$=$\mathbf{n}$. Then $\psi' = \psi = \mathsf{post}_{\sigma}(\psi)$.

2. **Induction step:** Let σ have length $l+1$, i.e. $\sigma = \sigma' * c$, where the length of σ' is l and $c \in \mathcal{C}$. Let $\mathbf{r}{:}(\varphi, Q)$ be the parent of $\mathbf{m}$. Then $\mathbf{n} \xrightarrow{\sigma'} \mathbf{r}$. By induction hypothesis $\mathsf{post}_{\sigma'}(\psi) \leq \varphi$ and thus, $\mathsf{post}_c(\mathsf{post}_{\sigma'}(\psi)) \leq \mathsf{post}_c(\varphi)$. $\psi' = \alpha_Q(\mathsf{post}_c(\varphi))$ and so, $\mathsf{post}_c(\varphi) \leq \psi'$. Hence, $\mathsf{post}_\sigma(\psi) \leq \psi'$.

□

Proposition 3.11. *Let a node $\boldsymbol{n}_1{:}(\psi_1, P_1)$ be refined in T_i with the set of atoms Q. Let $k > i$ be such that $\boldsymbol{n}_1$ is not deleted in any of the trees T_j where $i < j < k$. Let $\boldsymbol{n}_2{:}(\psi_2, P_2)$ be a node in T_k in the subtree of $\boldsymbol{n}_1$ ($\boldsymbol{n}_2$ can be also $\boldsymbol{n}_1$). Then $Q \subseteq P_2$.*

Proof. Let the label of $\mathbf{n}_1$ in T_k be (ψ_1', P_1'). Then $Q \subseteq P_1'$ and $\psi_1' = \psi_1$ since $\mathbf{n}_1$ is not deleted in any of the trees T_j where $i < j < k$. By Proposition 3.8 $P_1' \subseteq P_2$ and hence $Q \subseteq P_2$. □

3.3 Soundness of the Algorithm

Theorem 3.12. *If the algorithm terminates then:*

1. *if it returns NOT CORRECT, then the set of states bad is reachable.*
2. *if it returns a formula φ, it holds that:*
   - $\mathit{init} \leq \varphi$,
   - $\mathit{post}(\varphi) \leq \varphi$,
   - $\varphi \leq \mathit{safe}$.

Proof.

1. Suppose that the algorithm terminates and returns NOT CORRECT. We know that the intersection of the reachable region and the error region of the root node is not empty. This means that there is an initial state, from which a state in bad is reachable, formally $[\![\mathsf{init}]\!] \cap [\![\mathsf{lfp}(\mathsf{pre}, \mathsf{unsafe})]\!] \neq \emptyset$.
2. Suppose that the algorithm terminates and returns a formula φ which is $\bigvee_{\theta \in F} \theta$.

- The root node is labeled with (init, P) and marked as uncovered. Hence, $\mathsf{init} \sqsubseteq \varphi$ and thus, $\mathsf{init} \leq \varphi$.
- Let $s \in [\![\mathsf{post}(\varphi)]\!]$. Then, there exist $c \in \mathcal{C}$ and $s' \in [\![\varphi]\!]$ such that $s' \xrightarrow{c} s$. Since $s' \in [\![\varphi]\!]$, there exists $\theta \in F$ such that $s' \in [\![\theta]\!]$. There is a node $\mathbf{n}'{:}(\theta, P')$ in the current tree marked as uncovered. Since L is empty and $[\![\mathsf{post}_c(\theta)]\!] \neq \emptyset$ because $s \in [\![\mathsf{post}_c(\theta)]\!]$, there exists a node $\mathbf{n}{:}(\psi, P)$ in the current tree such that $s \in [\![\psi]\!]$ and $\mathbf{n}' \xrightarrow{c} \mathbf{n}$. If $\mathbf{n}$ is marked as uncovered, then $\psi \in F$ and hence, $s \in [\![\varphi]\!]$. Otherwise $\psi \sqsubseteq \bigvee_{\theta \in F} \theta$ and hence, there is an uncovered node $\mathbf{m}{:}(\psi'', P'')$ such that $\psi'' \in F$ and $\psi \sqsubseteq \psi''$. We have that $s \in [\![\psi'']\!]$ and so, $s \in [\![\varphi]\!]$.
- It is clear that $\varphi \leq \mathsf{safe}$ because $\varphi = \bigvee_{\theta \in F} \theta$ and for all elements θ of F, $\theta \leq \mathsf{safe}$.

□

Chapter 4

Forward Abstract Iteration with Backward Abstraction Refinement for WSTS

4.1 The Algorithm

In this chapter we give an instantiation of the schema from the previous chapter and prove that the obtained algorithm is guaranteed to terminate on WSTS.

We assume that $\mathcal{S}$ is a transition system represented as a program and that $\preceq$ is a well-quasi ordering between states that is expressible with a formula $\varphi_{\preceq}$. Moreover, we assume that the underlying theory admits quantifier elimination and that entailment is decidable. These assumptions guarantee that the images of the operators post, pre and ucpre on formulas are computable. For this algorithm we use the following definition of error region.

Definition 4.1 (**Error region**)**.** If $\mathbf{m}$ and $\mathbf{n}$ are nodes in some T_i and $\mathbf{m} \xrightarrow{\sigma} \mathbf{n}$, $|\sigma| = l$, and $\mathbf{n}$ is an error node, then $\bigvee_{i=0}^{l} \mathsf{ucpre}^i(\mathsf{unsafe})$ is called the *error region* of $\mathbf{m}$ for $\mathbf{n}$.

Now we show that the above definition satisfies the required properties.

Proposition 4.2. *Let the reachable region of a node $\boldsymbol{m}$ be φ, $\boldsymbol{n}$ be an error node with $\boldsymbol{m} \xrightarrow{\sigma} \boldsymbol{n}$ and χ be the error region of $\boldsymbol{m}$ for $\boldsymbol{n}$.*

- *If $\varphi \leq \neg\chi$ then $[\![\varphi]\!] \cap [\![\mathit{pre}_\sigma(\mathit{unsafe})]\!] = \emptyset$.*

- *If* $\varphi \not\leq \neg\chi$ *then* $[\![\varphi]\!] \cap [\![\mathit{lfp}(\mathit{pre}, \mathit{unsafe})]\!] \neq \emptyset$.

Proof. The error region is $\chi = \bigvee_{i=0}^{l} \mathsf{ucpre}^i(\mathsf{unsafe})$, where $l = |\sigma|$.

- If $[\![\varphi]\!] \cap [\![\bigvee_{i=0}^{l} \mathsf{ucpre}^i(\mathsf{unsafe})]\!] = \emptyset$ then it holds that $[\![\varphi]\!] \cap [\![\mathsf{ucpre}^l(\mathsf{unsafe})]\!] = \emptyset$. Hence, $[\![\varphi]\!] \cap [\![\mathsf{pre}_\sigma(\mathsf{unsafe})]\!] = \emptyset$.
- If $[\![\varphi]\!] \cap [\![\bigvee_{i=0}^{l} \mathsf{ucpre}^i(\mathsf{unsafe})]\!] \neq \emptyset$ then by Proposition 2.34 it holds that $[\![\varphi]\!] \cap [\![\mathsf{lfp}(\mathsf{pre}, \mathsf{unsafe})]\!] \neq \emptyset$.

□

The pivot node is determined as the first node on the path from the error node backwards towards the root for which the intersection of the reachable region and the error region is empty. Procedure 2 determines the pivot node according to the definition of an error region given above for an error node **n** given as an argument.

Require: Program $\mathcal{S}$, tree T, formula unsafe, error node **n**
  $\mathbf{n}'' := \mathbf{n}$; $\psi :=$ unsafe; $\chi :=$ unsafe; $\psi'' :=$ unsafe
  while $\psi'' \not\leq \neg\chi$ **do**
    if $\mathbf{n}''$ is the root **then**
      return NULL
    else
      $\mathbf{n}''{:}(\psi'', \mathrm{P}'') :=$parent$(\mathbf{n}'')$
      $\psi :=$ ucpre(ψ)
      $\chi := \chi \vee \psi$
    end if
  end while
  return $\mathbf{n}''{:}(\psi'', \mathrm{P}'')$

Procedure 2: pivot-Counterexample Analysis for WSTS

If there is no pivot node, i.e. if the intersection of the reachable region of the root with the error region of the root is not empty, then the procedure pivot returns NULL and the abstract interpretation algorithm terminates with NOT CORRECT. Otherwise the pivot node is refined and the predicates that are added to its set of support predicates are determined by the procedure predicates which for a trace σ returns the set of negations of the atoms that appear in $\bigvee_{k=0}^{|\sigma|} \mathsf{ucpre}^k(\mathsf{unsafe})$.

Require: Program $\mathcal{S}$, formula unsafe
  create root $\mathbf{r}$ labeled with (init, P_0) $//P_0 = \mathsf{atoms}(\mathsf{init}) \cup \mathsf{atoms}(\neg\mathsf{unsafe})$
  $L := \{\mathbf{r}\}$; $F := \emptyset$
  while $L \neq \emptyset$ **do**
    pop $\mathbf{n}$:(ψ, P) from L
    if $\psi \leq \mathsf{safe}$ **then**
      $\varphi := \bigvee_{\theta \in F} \theta$
      if $\psi \sqsubseteq \varphi$ **then**
        mark $\mathbf{n}$ as *covered*
      else
        for all command $c \in \mathcal{C}$ **do**
          (ψ', P):=$\mathsf{post}_c^{\#}((\psi, P))$
          if $\psi' \not\leq \mathsf{false}$ **then**
            construct a child $\mathbf{n}'$ of $\mathbf{n}$ with label (ψ', P)
            label the arc $\mathbf{n} \rightarrow \mathbf{n}'$ with c
            mark $\mathbf{n}'$ as *unprocessed*; push $\mathbf{n}'$ to L
            $F := F \cup \{\psi\}$
            mark $\mathbf{n}$ as *uncovered*
          end if
        end for
      end if
    else
      $\mathbf{m}$:(φ, Q) := $\mathsf{pivot}(\mathbf{n})$; σ := the trace from $\mathbf{m}$ to $\mathbf{n}$
      if $\mathbf{m}$==NULL **then**
        return NOT CORRECT
      else
        relabel $\mathbf{m}$ with $(\varphi, Q \cup \mathsf{predicates}(\sigma))$
        // We say that the node $\mathbf{m}$ is refined with
        // $\mathsf{atoms}(\neg \bigvee_{k=0}^{|\sigma|} \mathsf{ucpre}^k(\mathsf{unsafe}))$ w.r.t. the trace σ
        remove the subtrees starting at the children of $\mathbf{m}$
        change the mark of $\mathbf{m}$ to *unprocessed*; push $\mathbf{m}$ to L
        remove from F the formulas corresponding to
          the deleted nodes and $\mathbf{m}$
      end if
    end if
  end while
  return $\varphi := \bigvee_{\theta \in F} \theta$

Algorithm 3: Predicate Abstraction for WSTS

The procedure for abstract iteration with abstraction refinement is given as Algorithm 3. Here the list of unprocessed nodes L is interpreted as a stack, i.e. the state space is explored in depth-first order. Note that in this case all nodes processed after the pivot node are in its subtree. Hence, when the pivot node is refined all nodes that have been marked as covered after it has been processed are deleted. So, they need not to be marked as unprocessed. If the state space is explored in breadth-first order, then all nodes that are marked as covered after the pivot node was processed should be marked as unprocessed. Then all results also hold in that case. Here we consider depth-first exploration just to avoid the complication mentioned above.

Assume that the program $\mathcal{S} = \langle X, \mathsf{init}, \delta \rangle$ is such that the corresponding transition system $(\mathcal{S}, \preceq)$ is a WSTS. Moreover, let the set $[\![\mathsf{unsafe}]\!]$ be upward-closed. Now we prove that under these assumptions the procedure terminates.

4.2 Termination

The termination of the algorithm described in the previous section we prove by contradiction. Each of the iterations of the loop is terminating, so if we assume that the loop does not terminate then it generates an infinite sequence of trees

$$T_0, T_1, \ldots, T_k, \ldots.$$

First we prove several lemmas and propositions that will be useful for the proof of the main theorem.

The following lemma and its corollary formalize the fact, that if a node is refined with the set of negations of the atoms appearing in $\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe})$, then in the next iterations as long as this node is not deleted all the nodes in its subtree are safe, i.e. the node is not refined.

Lemma 4.3. *Let* $\boldsymbol{n}{:}(\psi, P)$ *be a node such that* $\mathit{atoms}(\neg \mathit{lfp}(\mathit{ucpre}, \mathit{unsafe})) \subseteq P$ *and* $\psi \leq \neg \mathit{lfp}(\mathit{ucpre}, \mathit{unsafe})$*. Then, for every* $j \geq 0$ *it holds that*

$$[(\mathit{post}^{\#})^j((\psi, P))] \leq \mathit{safe}$$

Proof. We prove by induction on j that

$$[(\mathsf{post}^{\#})^j((\psi, P))] \leq \neg \mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe}).$$

- **Base case:** $j = 0$
  We have that $(\mathsf{post}^{\#})^0((\psi, P)) = (\psi, P)$. Also, $\psi \leq \neg\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe})$ by assumption.

- **Induction step:** $(j + 1)$
  Induction hypothesis: $[(\mathsf{post}^{\#})^j((\psi, P))] \leq \neg\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe})$.
  Let $[(\mathsf{post}^{\#})^j((\psi, P))] = \chi$ and $[(\mathsf{post}^{\#})^{j+1}((\psi, P))] = \chi'$.
  Since $\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe}) \equiv \mathsf{ucpre}(\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe}))$,

  $\chi \leq \neg\mathsf{ucpre}(\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe}))$
  $\Rightarrow \quad \chi \leq \neg\mathsf{pre}(\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe}))$ (by Prop. 2.31)
  $\Rightarrow \quad \mathsf{post}(\chi) \leq \neg\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe})$ (by Prop. 2.25)
  $\Rightarrow \quad \alpha_P(\mathsf{post}(\chi)) \leq \alpha_P(\neg\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe}))$ (by monotonicity of α_P)
  $\Rightarrow \quad \alpha_P(\mathsf{post}(\chi)) \leq \neg\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe})$ (by Prop. 3.4)
  $\Rightarrow \quad \chi' \leq \neg\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe})$ (by Def. of $\mathsf{post}^{\#}$)

Hence, for all $j \geq 0$

$$[(\mathsf{post}^{\#})^j((\psi, P))] \leq \neg\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe}) \leq \neg\mathsf{unsafe} \equiv \mathsf{safe}.$$

□

Corollary 4.4.
Let a node $\boldsymbol{n}$:(ψ, P) be refined in T_i with $\mathsf{atoms}(\neg\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe}))$. Then, for every $k > i$, such that $\boldsymbol{n}$ is not deleted in any of the trees T_j where $i < j < k$, if $\boldsymbol{m}$:(φ, Q) is a node in T_k with $\boldsymbol{n} \xrightarrow{\sigma} \boldsymbol{m}$, it holds that $\varphi \leq \mathsf{safe}$.

Proof. Let the label of $\mathbf{n}$ in T_k be (ψ', P'). By Proposition 3.11, $\mathsf{atoms}(\neg\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe})) \subseteq P'$. Since $\mathbf{n}$ is refined in T_i with $\mathsf{atoms}(\neg\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe}))$ and not deleted in the trees T_j where $i < j < k$, $\psi' = \psi \leq \neg\mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe})$. If l is the length of σ then by Lemma 4.3 $[(\mathsf{post}^{\#})^l((\psi', P'))] \leq \mathsf{safe}$. Form Proposition 3.9 it follows that $\varphi \leq [\mathsf{post}^{\#}_{\sigma}((\psi', P'))]$. Thus, $\varphi \leq \mathsf{safe}$. □

To guarantee the termination of the procedure we ensure that a node cannot be refined more than once w.r.t. a trace of particular length. Thus, if we assume that a node in the tree is refined infinitely often then for every l it is eventually refined with respect to traces of length greater than l. This fact is used for proving that a node cannot be refined infinitely many times.

Lemma 4.5. *Let $\boldsymbol{n}$:(ψ, P) be a node such that $\mathsf{atoms}(\neg\chi_i) \subseteq P$ and $\psi \leq \neg\chi_i$, where $\chi_i = \bigvee_{k=0}^{i} \mathsf{ucpre}^k(\mathsf{unsafe})$. Then*

$$[(\mathsf{post}^{\#})^i((\psi, P))] \leq \mathsf{safe}.$$

Proof. We prove by induction on j that if $j \leq i$ then

$$[(\mathsf{post}^{\#})^j((\psi, P))] \leq \neg\mathsf{ucpre}^{i-j}(\mathsf{unsafe}).$$

- **Base case:** $j = 0$
  We have that $(\mathsf{post}^{\#})^0((\psi, P)) = (\psi, P)$. Since $\chi_i = \bigvee_{k=0}^{i} \mathsf{ucpre}^k(\mathsf{unsafe})$ and $\psi \leq \neg\chi_i$, $\psi \leq \neg\mathsf{ucpre}^i(\mathsf{unsafe})$.

- **Induction step:** $(j+1) \leq i$
  Induction hypothesis: $[(\mathsf{post}^{\#})^j((\psi, P))] \leq \neg\mathsf{ucpre}^{i-j}(\mathsf{unsafe})$.
  Let $[(\mathsf{post}^{\#})^j((\psi, P))] = \chi$ and $[(\mathsf{post}^{\#})^{j+1}((\psi, P))] = \chi'$.
  Then $\chi \leq \neg\mathsf{ucpre}^{i-j}(\mathsf{unsafe})$
  $\Rightarrow$ $\chi \leq \neg\mathsf{ucpre}(\mathsf{ucpre}^{i-j-1}(\mathsf{unsafe}))$
  $\Rightarrow$ $\chi \leq \neg\mathsf{pre}(\mathsf{ucpre}^{i-j-1}(\mathsf{unsafe}))$ (by Prop. 2.31)
  $\Rightarrow$ $\mathsf{post}(\chi) \leq \neg\mathsf{ucpre}^{i-j-1}(\mathsf{unsafe})$ (by Prop. 2.25)
  $\Rightarrow$ $\alpha_P(\mathsf{post}(\chi)) \leq \alpha_P(\neg\mathsf{ucpre}^{i-(j+1)}(\mathsf{unsafe}))$ (by monotonicity of α_P)
  $\Rightarrow$ $\alpha_P(\mathsf{post}(\chi)) \leq \neg\mathsf{ucpre}^{i-(j+1)}(\mathsf{unsafe})$ (by Prop. 3.4)
  $\Rightarrow$ $\chi' \leq \neg\mathsf{ucpre}^{i-(j+1)}(\mathsf{unsafe})$ (by Def. of $\mathsf{post}^{\#}$)

Thus, for $j = i$ we have

$$[(\mathsf{post}^{\#})^i((\psi, P))] \leq \neg\mathsf{ucpre}^0(\mathsf{unsafe}) = \neg\mathsf{unsafe} \equiv \mathsf{safe}.$$

□

Corollary 4.6. *Let a node $\boldsymbol{n}_1$:(ψ_1, P_1) be refined in T_i w.r.t. a trace σ. Let $k > i$ be such that $\boldsymbol{n}_1$ is not deleted in any of the trees T_j where $i < j < k$ and let $\boldsymbol{n}_2$:(ψ_2, P_2) be a node in T_k in the subtree of $\boldsymbol{n}_1$. Then, for any trace τ such that $|\tau| = |\sigma| = l$, $\boldsymbol{n}_2$ is not refined in T_k w.r.t. τ.*

Proof. Assume that $\mathbf{n}_2$ is refined in T_k w.r.t. τ where $|\tau| = l$ and $\mathbf{m}$:(φ, Q) is the corresponding error node in T_k. Then, $\psi_2 \leq \neg\bigvee_{k=0}^{l} \mathsf{ucpre}^k(\mathsf{unsafe})$. According to Proposition 3.11 $\mathsf{atoms}(\neg\bigvee_{k=0}^{l} \mathsf{ucpre}^k(\mathsf{unsafe})) \subseteq P_2$. Hence, by Lemma 4.5 $[(\mathsf{post}^{\#})^l((\psi_2, P_2))] \leq \mathsf{safe}$. From Proposition 3.9 it follows that $\varphi \leq [\mathsf{post}^{\#}_{\sigma}((\psi_2, P_2))]$. Thus, $\varphi \leq \mathsf{safe}$ which contradicts to the fact that $\mathbf{m}$ is an error node. □

Lemma 4.7. *Let a node* $\boldsymbol{n}$ *be deleted only finitely many times and refined infinitely many times, i.e. there is* $k \geq 0$ *such that* $\boldsymbol{n}$ *is not deleted from any of the trees* $T_k, T_{k+1}, \ldots$ *and in infinitely many of these trees* $\boldsymbol{n}$ *is refined. Then, there is an infinite subsequence* $T_{k_0}, T_{k_1}, \ldots$ *of the above sequence, and a sequence of traces* $\sigma_0, \sigma_1, \ldots$*, such that*

- $|\sigma_i| < |\sigma_{i+1}|$ *and*
- *in* T_{k_i} $\boldsymbol{n}$ *is refined w.r.t.* σ_i*.*

Proof. Let a node $\mathbf{n}$ be deleted only finitely many times and refined infinitely many times, i.e. there is $k \geq 0$ such that $\mathbf{n}$ is not deleted from any of the trees $T_k, T_{k+1}, \ldots$ and in infinitely many of these trees $\mathbf{n}$ is refined. Then there is an infinite subsequence of the above sequence $T_{j_0}, T_{j_1}, \ldots$ and a sequence of traces $\tau_0, \tau_1, \ldots$ such that $\mathbf{n}$ is refined in T_{j_i} with respect to τ_i. According to Corollary 4.6 (with $n_1 = n_2 = n$) all members of the sequence $\tau_0, \tau_1, \ldots$ have different lengths. Thus, we can choose a subsequence $T_{k_0}, T_{k_1}, \ldots$ of the above sequence of trees and a corresponding subsequence $\sigma_0, \sigma_1, \ldots$ of $\tau_0, \tau_1, \ldots$, such that $|\sigma_i| < |\sigma_{i+1}|$ and $\mathbf{n}$ is refined in T_{k_i} w.r.t σ_i. □

Lemma 4.8. *If a node is deleted finitely many times, it is refined only finitely many times.*

Proof. Assume for contradiction, that $\mathbf{n}$ is deleted only finitely many times and refined infinitely many times, i.e. there is $k \geq 0$, such that $\mathbf{n}$ is not deleted from any of the trees $T_k, T_{k+1}, \ldots$ and in infinitely many of these trees $\mathbf{n}$ is refined. Then according to the previous lemma there is an infinite subsequence $T_{k_0}, T_{k_1}, \ldots$ of the above sequence and a sequence of traces $\sigma_0, \sigma_1, \ldots$, such that

- $|\sigma_i| < |\sigma_{i+1}|$
- $\mathbf{n}$ is refined in T_{k_i} with respect to σ_i.

Let $|\sigma_i| = l_i$. At each of the corresponding refinements the algorithm considers the formula $\chi_{l_i} = \bigvee_{k=0}^{l_i} \mathsf{ucpre}^k(\mathsf{unsafe})$. Then

$$\chi_{l_0} \leq \chi_{l_1} \leq \cdots$$

and each set $[\![\chi_{l_i}]\!]$ is upward-closed. According to Lemma 2.12, there exists an index j, such that

$$[\![\chi_{l_j}]\!] = [\![\chi_{l_{j+1}}]\!] = \ldots.$$

This means that,

$$\chi_{l_j} \equiv \chi_{i_{j+1}} \equiv \dots.$$

Since $l_0 < l_1 < \dots$, $\bigvee_{i=0}^{\infty} \chi_{l_i} \equiv \bigvee_{i=0}^{\infty} \mathsf{ucpre}^i(\mathsf{unsafe}) \equiv \mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe})$. Then, $\chi_{l_j} \equiv \mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe})$. Let $\mathbf{m}{:}(\varphi, Q)$ be the node for which $\mathbf{n} \overset{\sigma_{j+1}}{\rightarrow} \mathbf{m}$ in $T_{k_{j+1}}$. Then $\varphi \not\leq \mathsf{safe}$ by the choice of σ_{j+1}. By Corollary 4.4, $\varphi \leq \mathsf{safe}$. This is a contradiction. □

Lemma 4.9. *A node cannot be deleted infinitely many times.*

Proof. A node $\mathbf{n}$ is deleted form some tree in the sequence only if a node on the path from the root to $\mathbf{n}$ that is different from $\mathbf{n}$ is refined in this tree. For every node we show that it is deleted only finitely many times. The proof goes by induction on the depth of the node.

- **Base case:**
  The root node is never deleted.

- **Induction step:**
  Let $l > 0$. Assume that every node with depth less than l is deleted only finitely many times. Let $\mathbf{n}$ be a node with depth l. By induction hypothesis and by Lemma 4.8 we have that every node on the path from the root to $\mathbf{n}$ is refined only finitely many times. Thus, there is a k such that none of the nodes on the path from the root to $\mathbf{n}$ is refined in the trees $T_k, T_{k+1}, \dots$. Then $\mathbf{n}$ is not deleted from $T_k, T_{k+1}, \dots$, i.e. it is deleted only finitely many times.

□

Lemma 4.10. *Let infinitely many different nodes be refined by the procedure. Then there is an infinite sequence of different nodes $\boldsymbol{n}_0, \boldsymbol{n}_1, \dots, \boldsymbol{n}_i, \dots$ and an infinite sequence of trees $T_{k_0}, T_{k_1}, \dots, T_{k_i}, \dots$, $(k_i < k_{i+1})$ such that:*

- *$\boldsymbol{n}_{i+1}$ is in the subtree of $\boldsymbol{n}_i$,*

- *$\boldsymbol{n}_i$ is refined in T_{k_i},*

- *none of the ancestors of $\boldsymbol{n}_i$ (including $\boldsymbol{n}_i$) is refined in the trees T_j with $j > k_i$.*

Proof. Let infinitely many different nodes be refined. We construct inductively the infinite sequence of nodes $\mathbf{n}_0, \mathbf{n}_1, \ldots, \mathbf{n}_i, \ldots$ and the infinite sequence of trees $T_{k_0}, T_{k_1}, \ldots, T_{k_i}, \ldots$ such that for every i for the finite subsequence $\mathbf{n}_0, \mathbf{n}_1, \ldots, \mathbf{n}_i$ it holds that:

- for all $l \leq i$, $\mathbf{n}_l$ is refined in T_{k_l},
- for all $l < i$ $\mathbf{n}_{l+1}$ is in the subtree of $\mathbf{n}_l$,
- infinitely many nodes in the subtree of $\mathbf{n}_i$ are refined in the trees T_j with $j > k_i$,
- for all $l \leq i$ none of the ancestors of $\mathbf{n}_l$ (including $\mathbf{n}_l$) is refined in the trees T_j with $j > k_l$.

The construction is the following.

- There is a node in the subtree of which infinitely many nodes are refined since $\mathcal{C}$ is finite. We define $\mathbf{n}_0$ to be the node with the smallest depth, such that $\mathbf{n}_0$ is refined by the procedure and infinitely many nodes in the subtree of $\mathbf{n}_0$ are refined. Let k_0 be the largest index such that $\mathbf{n}_0$ is refined in T_{k_0}. It is clear, that the sequence $\mathbf{n}_0$ satisfies the above conditions because by the choice of $\mathbf{n}_0$ we have that none of its ancestors is ever refined.
- Let us assume that we have constructed the sequences $\mathbf{n}_0, \mathbf{n}_1, \ldots, \mathbf{n}_i$ and $T_{k_0}, T_{k_1}, \ldots, T_{k_i}$ that satisfy the above conditions. Since there are infinitely many nodes in the subtree of $\mathbf{n}_i$ that are refined in the trees T_j with $j > k_i$ and $\mathcal{C}$ is finite, it holds that there is a node in the subtree of $\mathbf{n}_i$, different from $\mathbf{n}_i$, such that infinitely many nodes in its subtree are refined in the trees T_j with $j > k_i$. Let $\mathbf{n}_{i+1}$ be the node with the smallest depth in the subtree of $\mathbf{n}_i$ such that
  - $\mathbf{n}_{i+1}$ is refined in a tree T_k such that $k > k_i$ and
  - infinitely many nodes in the subtree of $\mathbf{n}_{i+1}$ are refined in the trees T_j with $j > k_i$.

  Let k_{i+1} be the largest index such that $\mathbf{n}_{i+1}$ is refined in $T_{k_{i+1}}$. Then we are sure that none of the nodes on the path from $\mathbf{n}_i$ to $\mathbf{n}_{i+1}$ (including $\mathbf{n}_{i+1}$) are refined in the trees T_j with $j > k_{i+1}$ by the choice of $\mathbf{n}_{i+1}$.

By the induction hypothesis we have that none of the ancestors of $\mathbf{n}_i$ (including $\mathbf{n}_i$) is refined in some of the trees T_j with $j > k_i$. Hence, none of the ancestors of $\mathbf{n}_{i+1}$ (including $\mathbf{n}_{i+1}$) is refined in the trees T_j with $j > k_{i+1}$. According to the choice of $\mathbf{n}_{i+1}$, infinitely many nodes in its subtree are refined in the trees T_j with $j > k_{i+1}$. Hence, the sequence of nodes $\mathbf{n}_0, \mathbf{n}_1, \ldots, \mathbf{n}_{i+1}$ satisfies the requirements. The properties of the finite subsequences of the infinite sequence constructed above imply that it satisfies the desired conditions.

□

Theorem 4.11. *The abstraction refinement procedure given as Algorithm 3 terminates on every WSTS $(\mathcal{S}, \preceq)$ and upward-closed set of error states.*

Proof. Assume for contradiction that the procedure does not terminate. This means that the loop is executed infinitely many times, i.e. the stack of unprocessed nodes L never becomes empty and all abstract counterexamples are spurious. At each iteration of the loop the current tree constructed by the procedure we denote with T_k. So, we have the sequence of finite trees

$$T_0, T_1, \ldots.$$

Assume that for some $k \geq 0$ for all $i \geq k$ no more refinements are performed, i.e. T_{i+1} is obtained from T_i by processing a safe node from L. Let $\mathbf{n}{:}(\psi, P)$ be an arbitrary unprocessed node in T_k. There are two possible cases:

- $\mathbf{n}$ is found to be covered, so none of the children of $\mathbf{n}$ are added to the tree
- $\mathbf{n}$ is not covered in which case the children of $\mathbf{n}$ are generated and added to the tree and to L.

$\mathbf{n}$ is not refined in $T_k, T_{k+1}, \ldots$, which means that for $i \geq k$ in T_i the set of support predicates of $\mathbf{n}$ is P. The application of $\mathsf{post}_c^{\#}$ preserves the set of support predicates and the nodes added to the subtree of $\mathbf{n}$ are not refined in $T_k, T_{k+1}, \ldots$ either. Thus, all nodes that appear in the subtree of $\mathbf{n}$ in $T_k, T_{k+1}, \ldots$ have a set of support predicates P. Hence, all these nodes are labeled with formulas that belong to $\mathcal{L}(P)$ which is finite. Since a node is added to the tree only if it is not covered, only finitely many nodes may be

added to the subtree of $\mathbf{n}$. The tree T_k has only finitely many unprocessed nodes, so only finitely many new nodes are added to L. Hence, at some point $L = \emptyset$ and the algorithm terminates which contradicts to our assumption. Thus, infinitely many refinements are performed.

By Lemma 4.9 no node is deleted infinitely often and thus, according to Lemma 4.8 no node is refined infinitely often. Thus, infinitely many different nodes are refined. By Lemma 4.10, there is an infinite sequence of nodes $\mathbf{n}_0, \mathbf{n}_1, \ldots, \mathbf{n}_i, \ldots$ and an infinite sequence of trees $T_{k_0}, T_{k_1}, \ldots, T_{k_i}, \ldots$, such that

- $\mathbf{n}_{i+1}$ is in the subtree of $\mathbf{n}_i$,
- $\mathbf{n}_i$ is refined in T_{k_i},
- none of the ancestors of $\mathbf{n}_i$ (including $\mathbf{n}_i$) is refined in the trees T_j with $j > k_i$.

Let $\sigma_0, \sigma_1, \ldots$ be the corresponding sequence of traces with respect to which the nodes from the above sequence are refined ($\mathbf{n}_i$ is refined w.r.t. σ_i in T_{k_i}). By the third property of the sequence of nodes, for each i the node $\mathbf{n}_i$ is not deleted in any of the trees T_j with $j > k_i$. According to Corollary 4.6 $\sigma_0, \sigma_1, \ldots$ should have different lengths. This means that there are arbitrarily long sequences among them. Thus, we can choose an infinite subsequence of the above sequence of nodes, $\mathbf{m}_0, \mathbf{m}_1, \ldots$, such that for the corresponding subsequence of traces $\tau_0, \tau_1, \ldots$ of $\sigma_0, \sigma_1, \ldots$ it holds that $|\tau_i| < |\tau_{i+1}|$. Let $|\tau_i| = l_i$. The algorithm considers the sequence of formulas

$$\chi_{l_0}, \chi_{l_1}, \ldots,$$

where

$$\chi_{l_i} \equiv \bigvee_{k=0}^{l_i} \mathsf{ucpre}^k(\mathsf{unsafe}).$$

Thus, for every i $[\![\chi_{l_i}]\!]$ is an upward-closed set and $\chi_{l_i} \leq \chi_{l_{i+1}}$. According to Lemma 2.12 there is an index j, such that $\chi_{l_j} \equiv \mathsf{lfp}(\mathsf{ucpre}, \mathsf{unsafe})$. Thus, according to Corollary 4.4 after $\mathbf{m}_j$ is refined w.r.t. τ_j, in the subsequent trees all nodes in the subtree of $\mathbf{m}_j$ are safe and hence, in all subsequent trees all nodes in the subtree of $\mathbf{m}_{j+1}$ are safe. This contradicts to the fact that after $\mathbf{m}_j$ is refined w.r.t. τ_j, $\mathbf{m}_{j+1}$ is refined. This concludes the proof. □

Chapter 5

Forward Abstract Iteration with Backward Abstraction Refinement for WSLTS

In this chapter we consider WSTS with strong compatibility with the transition labels taken into account, i.e. WSLTS. Some families of WSTS such as various extensions of Petri nets or real time automata (see [1]) belong to this class. This class of transition systems is a subclass of the class of WSTS, but we obtain a possibly more efficient algorithm for predicate abstraction with abstraction refinement that is guaranteed to terminate on members of this class.

For WSLTS the set of predecessors of an upward-closed set of states is also upward-closed. This is formalized in the following lemma.

Lemma 5.1. *If $[\![\varphi]\!]$ is upward-closed and c is a guarded command, then the set $[\![\mathit{pre}_c(\varphi)]\!]$ is also upward-closed.*

Proof. Let $[\![\varphi]\!]$ be upward-closed. Let $s \in [\![\mathsf{pre}_c(\varphi)]\!]$ and $s \preceq t$. There exists a state $u \in [\![\varphi]\!]$ such that $s \xrightarrow{c} u$. Hence, there is a sate v such that $t \xrightarrow{c} v$ and $u \preceq v$. Thus, $v \in [\![\varphi]\!]$ and hence, $t \in [\![\mathsf{pre}_c(\varphi)]\!]$. □

5.1 The Algorithm

We assume that $\mathcal{S}$ is a transition system represented as a program. Also, entailment should be decidable in the underlying theory and it should admit

quantifier elimination. This guarantees that the images of the operators post and pre on formulas are computable.

Here we give an instantiation of the schema from Chapter 3 and show that the obtained algorithm is guaranteed to terminate on WSLTS. This instantiation results in a possibly more efficient algorithm since fewer predicates are added to the support set of the pivot node. The definition of error region that we use in this chapter is the following.

Definition 5.2 (**Error region**)**.** If $\mathbf{m}$ and $\mathbf{n}$ are nodes in some T_i with $\mathbf{m} \xrightarrow{\sigma} \mathbf{n}$ and $\mathbf{n}$ is an error node then $\mathsf{pre}_\sigma(\mathsf{unsafe})$ is called the *error region* of $\mathbf{m}$ for $\mathbf{n}$.

Proposition 5.3. *Let the reachable region of a node $\boldsymbol{m}$ be φ, $\boldsymbol{n}$ be an error node with $\boldsymbol{m} \xrightarrow{\sigma} \boldsymbol{n}$ and χ be the error region of $\boldsymbol{m}$ for $\boldsymbol{n}$.*

- *If $\varphi \leq \neg\chi$ then $[\![\varphi]\!] \cap [\![\mathit{pre}_\sigma(\mathit{unsafe})]\!] = \emptyset$.*
- *If $\varphi \not\leq \neg\chi$ then $[\![\varphi]\!] \cap [\![\mathit{lfp}(\mathit{pre}, \mathit{unsafe})]\!] \neq \emptyset$.*

Note that by the definition of error region we have that if $\varphi \not\leq \neg\chi$, then $[\![\varphi]\!] \cap [\![\mathit{pre}_\sigma(\mathit{unsafe})]\!] \neq \emptyset$.

Require: Program $\mathcal{S}$, tree T, formula unsafe, node $\mathbf{n}$
  $\mathbf{n}' := \mathbf{n}$; $\mathbf{n}'' := \mathbf{n}$; $\psi :=$ unsafe; $\psi'' :=$ unsafe;
  while $\psi'' \not\leq \neg\psi$ **do**
    if $\mathbf{n}''$ is the root **then**
      return NULL
    else
      $\mathbf{n}' := \mathbf{n}''$
      $\mathbf{n}'':(\psi'', \mathrm{P}'') :=$parent$(\mathbf{n}'')$
      c:= the label on $\mathbf{n}'' \rightarrow \mathbf{n}'$
      $\psi := \mathsf{pre}_c(\psi)$
    end if
  end while
  return $\mathbf{n}'':(\psi'', P'')$

Procedure 4: pivot-Counterexample Analysis for WSLTS

The set of refinement predicates added to the set of support predicates of the pivot node is changed accordingly. It is computed by the procedure

predicates which takes as an argument the trace σ between the pivot node and the error node and returns $\mathsf{atoms}(\neg \bigvee_{k=1}^{|\sigma|+1} \mathsf{pre}_{[k,|\sigma|+1)}(\mathsf{unsafe}))$. Note that in this version the refinement is really guided by the counterexample since the pivot node and the refinement predicates depend on the particular trace σ. The procedure pivot that determines the pivot node is given as Procedure 4. The abstraction refinement procedure, which is similar to the algorithm in [13], is given in pseudocode as Algorithm 5.

Assume that the program $\mathcal{S} = \langle X, \mathsf{init}, \delta \rangle$ is such that the corresponding transition system $\mathcal{S} = \langle S, I, \mathcal{C}, \delta \rangle$ is a WSLTS with an ordering $\preceq$. Moreover, let the set $[\![\mathsf{unsafe}]\!]$ be upward-closed. Now we prove that under these assumptions the algorithm terminates.

5.2 Termination

In order to prove the termination of the procedure, we first show in the following lemma and its corollary that a node cannot be refined more than once w.r.t. the same trace σ.

Lemma 5.4. *Let $\boldsymbol{n}{:}(\varphi, P)$ be a node and $\sigma = c_1 \dots c_n$ be a trace. Let*

$$\chi_n = \bigvee_{i=1}^{n+1} \mathit{pre}_{\sigma[i,n+1)}(\mathit{unsafe}).$$

If $\mathit{atoms}(\neg\chi_n) \subseteq P$ and $\varphi \leq \neg \mathit{pre}_\sigma(\mathit{unsafe})$ then $[\mathit{post}^\#_\sigma((\varphi, P))] \leq \mathit{safe}$.

Proof. We prove by induction on i that if $i \leq n$ then

$$[\mathsf{post}^\#_{\sigma[1,i+1)}((\varphi, P))] \leq \neg \mathsf{pre}_{\sigma[i+1,n+1)}(\mathsf{unsafe}).$$

- **Base case:** $i = 0$
  $\mathsf{post}^\#_{\sigma[1,1)}((\varphi, P)) = (\varphi, P)$, $\varphi \leq \neg \mathsf{pre}_{\sigma[1,n+1)}(\mathsf{unsafe})$. Hence,
  $[\mathsf{post}^\#_{\sigma[1,1)}((\varphi, P))] \leq \neg \mathsf{pre}_{\sigma[1,n+1)}(\mathsf{unsafe})$.

- **Induction step:** $(i+1) \leq n$
  Induction hypothesis: $[\mathsf{post}^\#_{\sigma[1,i+1)}((\varphi, P))] \leq \neg \mathsf{pre}_{\sigma[i+1,n+1)}(\mathsf{unsafe})$
  Let $[\mathsf{post}^\#_{\sigma[1,i+1)}((\varphi, P))] = \chi$ and $[\mathsf{post}^\#_{\sigma[1,i+2)}((\varphi, P))] = \chi'$.

Then $\chi \leq \neg\mathsf{pre}_{\sigma[i+1,n+1)}(\mathsf{unsafe})$
$\Rightarrow \quad \chi \leq \neg\mathsf{pre}_{c_{i+1}}(\mathsf{pre}_{\sigma[i+2,n+1)}(\mathsf{unsafe}))$
$\Rightarrow \quad \mathsf{post}_{c_{i+1}}(\chi) \leq \neg\mathsf{pre}_{\sigma[i+2,n+1)}(\mathsf{unsafe})$ (by Prop 2.25)
$\Rightarrow \quad \alpha_P(\mathsf{post}_{c_{i+1}}(\chi)) \leq \alpha_P(\neg\mathsf{pre}_{\sigma[i+2,n+1)}(\mathsf{unsafe}))$
$\Rightarrow \quad \alpha_P(\mathsf{post}_{c_i+1}(\chi)) \leq \neg\mathsf{pre}_{\sigma[(i+1)+1,n+1)}(\mathsf{unsafe})$ (by Prop. 3.4)
$\Rightarrow \quad \chi' \leq \neg\mathsf{pre}_{\sigma[(i+1)+1,n+1)}(\mathsf{unsafe})$ (by Def. of $\mathsf{post}^{\#}$)

Thus, for $i = n$ we have

$$[\mathsf{post}^{\#}_{\sigma[1,n+1)}((\varphi, P))] \leq \neg\mathsf{pre}_{\sigma[n+1,n+1)}(\mathsf{unsafe}) = \neg\mathsf{unsafe} \equiv \mathsf{safe}.$$

□

Corollary 5.5. *Let a node $\boldsymbol{n}_1{:}(\psi_1, P_1)$ be refined in T_i w.r.t. a trace σ. Let $k > i$ be such that $\boldsymbol{n}_1$ is not deleted in any of the trees T_j where $i < j < k$ and let $\boldsymbol{n}_2{:}(\psi_2, P_2)$ be a node in T_k in the subtree of $\boldsymbol{n}_1$. Then $\boldsymbol{n}_2$ is not refined in T_k w.r.t. σ.*

Proof. Assume that $\mathbf{n}_2$ is refined in T_k w.r.t. σ and $\mathbf{m}{:}(\varphi, Q)$ is the corresponding error node in T_k. Then, $\psi_2 \leq \neg\mathsf{pre}_{\sigma}(\mathsf{unsafe})$. According to Proposition 3.11 $\mathsf{atoms}(\neg\bigvee_{i=1}^{|\sigma|+1}\mathsf{pre}_{\sigma[i,|\sigma|+1)}(\mathsf{unsafe})) \subseteq P_2$. Hence, by Lemma 5.4 $[\mathsf{post}^{\#}_{\sigma}((\psi_2, P_2))] \leq \mathsf{safe}$. From Proposition 3.9 it follows that $\varphi \leq [\mathsf{post}^{\#}_{\sigma}((\psi_2, P_2))]$. Thus, $\varphi \leq \mathsf{safe}$ which contradicts to the fact that $\mathbf{m}$ is an error node in T_k. □

Lemma 5.6. *Let a node $\boldsymbol{n}$ be deleted only finitely many times, i.e. there is $k \geq 0$, such that $\boldsymbol{n}$ is not deleted from any of the trees $T_k, T_{k+1}, \ldots$. Then $\boldsymbol{n}$ is refined only a finite number of times.*

Proof. Let a node $\mathbf{n}$ be deleted only finitely many times, i.e. there is $k \geq 0$, such that $\mathbf{n}$ is not deleted from any of the trees $T_k, T_{k+1}, \ldots$. Assume that $\mathbf{n}$ is refined in infinitely many of the trees $T_k, T_{k+1}, \ldots$. According to Corollary 5.5 each time $\mathbf{n}$ is refined with respect to a different trace. Let $\mathbf{n}$ be refined in the trees $T_{m_0}, T_{m_1}, \ldots$ with respect to the traces $\sigma_0, \sigma_1, \ldots$. Since $\mathcal{C}$ is finite, there is a $c \in \mathcal{C}$, such that there is an infinite subsequence $T_{r_0}, T_{r_1}, \ldots$ of $T_{m_0}, T_{m_1}, \ldots$ and a corresponding subsequence $\tau_0, \tau_1, \ldots$ of $\sigma_0, \sigma_1, \ldots$, such that for every i, $\tau_i[1] = c$. Let l_i be the length of τ_i and $\tau'_i = \tau_i[2, l_i + 1)$. Let $\mathbf{m}$ be the child of $\mathbf{n}$ with $\mathbf{n} \xrightarrow{c} \mathbf{m}$ and $\varphi_0, \varphi_1, \ldots$ be the reachable regions of $\mathbf{m}$ in $T_{r_0}, T_{r_1}, \ldots$. Then, for all i $[\![\varphi_i]\!] \cap [\![\mathsf{pre}_{\tau'_i}(\mathsf{unsafe})]\!] \neq \emptyset$ (otherwise $\mathbf{m}$ but not $\mathbf{n}$ would be refined in T_{r_i}). Let $s_0, s_1, \ldots$ be a sequence of states

such that $s_i \in [\![\varphi_i]\!] \cap [\![\mathsf{pre}_{\tau_i'}(\mathsf{unsafe})]\!]$. Since $\preceq$ is a well-quasi ordering on S, there are indices $i < j$ such that $s_i \preceq s_j$. According to the choice of s_i we have $s_i \in [\![\mathsf{pre}_{\tau_i'}(\mathsf{unsafe})]\!]$. Thus, $s_j \in [\![\mathsf{pre}_{\tau_i'}(\mathsf{unsafe})]\!]$, since $[\![\mathsf{pre}_{\tau_i'}(\mathsf{unsafe})]\!]$ is upward-closed. After the refinement preformed in T_{r_i} we have that for all p with $i < p$ it holds that $[\![\varphi_p]\!] \cap [\![\mathsf{pre}_{\tau_i'}(\mathsf{unsafe})]\!] = \emptyset$ (follows from the proof of Lemma 5.4). This means that $s_j \notin [\![\varphi_j]\!]$ which contradicts to the choice of s_j. So, $\mathbf{n}$ is refined only finitely many times. □

Proposition 5.7. *If a node $\boldsymbol{n}$:(φ, P) in the tree T_k is refined in this tree w.r.t. the trace σ then for every trace τ which is a prefix of σ and different from σ it holds that $\varphi \leq \neg \mathit{pre}_\tau(\mathit{unsafe})$.*

Proof. Assume for contradiction that $\varphi \not\leq \neg\mathsf{pre}_\tau(\mathsf{unsafe})$.
Thus, $[\![\varphi]\!] \cap [\![\mathsf{pre}_\tau(\mathsf{unsafe})]\!] \neq \emptyset$. Hence, $[\![\mathsf{post}_\tau(\varphi)]\!] \cap [\![\mathsf{unsafe}]\!] \neq \emptyset$. Let $\mathbf{m}$:(ψ, Q) be the node in T_k with $\mathbf{n} \xrightarrow{\tau} \mathbf{m}$. According to Proposition 3.10 we have that $\mathsf{post}_\tau(\varphi) \leq \psi$. Hence, $\psi \not\leq \mathsf{safe}$. This is not possible, since $\mathbf{n}$ is refined in T_k w.r.t. σ and $\mathbf{m}$ is a node on the path from $\mathbf{n}$ to the corresponding error node. This completes the proof by contradiction. □

Theorem 5.8. *The abstraction refinement procedure given as Algorithm 5 terminates on every WSLTS $(\mathcal{S}, \preceq)$ and upward-closed set of error states.*

Proof. Assume for contradiction that the procedure does not terminate. This means that the loop is executed infinitely many times, i.e. the stack of unprocessed nodes L never becomes empty and all counterexamples are spurious. At each iteration of the loop the current tree constructed by the procedure we denote with T_k. So, we have the sequence of finite trees $T_0, T_1, \ldots$. Assume that for some $k \geq 0$ for all $i \geq k$ no more refinements are performed, T_{i+1} is obtained from T_i by processing a safe node from L. Let $\mathbf{n}$:(ψ, P) be an arbitrary unprocessed node in T_k. There are two possible cases: $\mathbf{n}$ is found to be covered, so none of the children of $\mathbf{n}$ are added to the tree or $\mathbf{n}$ is not covered in which case the children of $\mathbf{n}$ are generated and added to the tree and to L. $\mathbf{n}$ is not refined in $T_k, T_{k+1}, \ldots$, which means that for $i \geq k$ in T_i the set of support predicates of $\mathbf{n}$ is P. The application of $\mathsf{post}_c^\#$ preserves the set of support predicates and the nodes added to the subtree of $\mathbf{n}$ are not refined in $T_k, T_{k+1}, \ldots$ either. Thus, all nodes that appear in the subtree of $\mathbf{n}$ in $T_k, T_{k+1}, \ldots$ have a set of support predicates P. Hence, all these nodes are labeled with formulas that belong to $\mathcal{L}(P)$, which is finite. Since a node is added to the tree only if it is not covered, only finitely many nodes may be

added to the subtree of $\mathbf{n}$. The tree T_k has only finitely many unprocessed nodes, so only finitely many new nodes are added to L. Hence, at some point L becomes empty and the algorithm terminates, which contradicts to our assumption. Thus, infinitely many refinements are performed.

Note that the changes in the procedure do not affect the proof of Lemma 4.9, except that Lemma 5.6 is used instead of Lemma 4.8 in this case, and hence it holds for this instantiation of the schema too. Hence, no node is deleted infinitely often and thus according to Lemma 5.6 no node is refined infinitely often. So, infinitely many different nodes are being refined. By Lemma 4.10, there is an infinite sequence of nodes $\mathbf{n}_0, \mathbf{n}_1, \ldots, \mathbf{n}_i, \ldots$ and an infinite sequence of trees $T_{k_0}, T_{k_1}, \ldots, T_{k_i}, \ldots$, such that

- $\mathbf{n}_{i+1}$ is in the subtree of $\mathbf{n}_i$,
- $\mathbf{n}_i$ is refined in T_{k_i},
- none of the ancestors of $\mathbf{n}_i$ (including $\mathbf{n}_i$) is refined in the trees T_j with $j > k_i$.

Let $\sigma_0, \sigma_1, \ldots$ be the corresponding sequence of traces with respect to which the nodes from the above sequence are refined ($\mathbf{n}_i$ is refined w.r.t. σ_i in T_{k_i}). By the third property of the sequence of nodes, for each i the node $\mathbf{n}_i$ is not deleted in any of the trees T_j with $j > k_i$. According to Corollary 5.5 all elements of $\sigma_0, \sigma_1, \ldots$ are different. Since $\mathcal{C}$ is finite, we can choose a subsequence $\mathbf{m}_0, \mathbf{m}_1, \ldots$ of the sequence of nodes $\mathbf{n}_0, \mathbf{n}_1, \ldots$, a subsequence of trees $T_{l_0}, T_{l_1}, \ldots$ and a corresponding subsequence $\tau_0, \tau_1, \ldots$ of $\sigma_0, \sigma_1, \ldots$, such that $\mathbf{m}_i$ is refined in T_{l_i} w.r.t. τ_i and such that for all i τ_i is a prefix of τ_{i+1} and different from τ_{i+1}. Hence, there exists $c \in \mathcal{C}$ such that for all i $\tau_i[1] = c$. Let the sequence of nodes $\mathbf{r}_0, \mathbf{r}_1, \ldots$ be such that $\mathbf{m}_i \xrightarrow{c} \mathbf{r}_i$. Let $\tau'_i = \tau_i[2, |\tau_i| + 1)$. Let φ_i and φ'_i be the reachable regions of $\mathbf{m}_i$ and $\mathbf{r}_i$ in T_{l_i} respectively. We have that $[\![\varphi'_i]\!] \cap [\![\mathsf{pre}_{\tau'_i}(\mathsf{unsafe})]\!] \neq \emptyset$. Let the sequence of states $s_0, s_1, \ldots$ be such that $s_i \in [\![\varphi'_i]\!] \cap [\![\mathsf{pre}_{\tau'_i}(\mathsf{unsafe})]\!]$.

There exist indices i and j, such that $i < j$ and $s_i \preceq s_j$. Since $s_i \in [\![\mathsf{pre}_{\tau'_i}(\mathsf{unsafe})]\!]$ and $[\![\mathsf{pre}_{\tau'_i}(\mathsf{unsafe})]\!]$ is upward-closed, $s_j \in [\![\mathsf{pre}_{\tau'_i}(\mathsf{unsafe})]\!]$.

By Proposition 5.7 we have that $\varphi_j \leq \neg\mathsf{pre}_{\tau_i}(\mathsf{unsafe})$. By Proposition 3.11 $\mathsf{atoms}(\neg \bigvee_{l=1}^{|\tau_i|+1} \mathsf{pre}_{\tau_i[l,|\tau_i|+1)}(\mathsf{unsafe}))$ is a subset of the set of support predicates of $\mathbf{m}_j$ in T_{l_j}. Hence, $[\![\varphi'_j]\!] \cap [\![\mathsf{pre}_{\tau'_i}(\mathsf{unsafe})]\!] = \emptyset$ (by the proof of Lemma 5.4). This contradicts to the fact that $s_j \in [\![\varphi'_j]\!] \cap [\![\mathsf{pre}_{\tau'_i}(\mathsf{unsafe})]\!]$. This completes the proof. □

Require: Program $\mathcal{S}$, formula unsafe
  create root $\mathbf{r}$ labeled with (init, P_0) $//P_0 = \mathsf{atoms}(\mathsf{init}) \cup \mathsf{atoms}(\neg\mathsf{unsafe})$
  $L = \{\mathbf{r}\}$; $F := \emptyset$
  while $L \neq \emptyset$ **do**
    pop $\mathbf{n}$:(ψ, P) from L
    if $\psi \leq \mathsf{safe}$ **then**
      $\varphi := \bigvee_{\theta \in F} \theta$
      if $\psi \sqsubseteq \varphi$ **then**
        mark $\mathbf{n}$ as *covered*
      else
        for all command $c \in \mathcal{C}$ **do**
          (ψ', P):=$\mathsf{post}_c^{\#}((\psi, P))$
          if $\psi' \not\leq \mathsf{false}$ **then**
            construct a child $\mathbf{n}'$ of $\mathbf{n}$ with label (ψ', P)
            label the arc $\mathbf{n} \rightarrow \mathbf{n}'$ with c
            mark $\mathbf{n}'$ as *unprocessed*; push $\mathbf{n}'$ to L
            $F := F \cup \{\psi\}$; mark $\mathbf{n}$ as *uncovered*
          end if
        end for
      end if
    else
      $\mathbf{m}$:(φ, Q) := $\mathsf{pivot}(\mathbf{n})$; σ := the trace from $\mathbf{m}$ to $\mathbf{n}$
      if $\mathbf{m}$==NULL **then**
        return NOT CORRECT
      else
        relabel $\mathbf{m}$ with $(\varphi, Q \cup \mathsf{predicates}(\sigma))$
        // We say that the node $\mathbf{m}$ is refined with
        // $\mathsf{atoms}(\neg \bigvee_{k=1}^{|\sigma|+1} \mathsf{pre}_{\sigma[k,|\sigma|+1)}(\mathsf{unsafe}))$
        //w.r.t. the trace σ
        remove the subtrees starting at the children of $\mathbf{m}$
        change the mark of $\mathbf{m}$ to *unprocessed*; push $\mathbf{m}$ to L
        remove from F the formulas corresponding to
          the deleted nodes and $\mathbf{m}$
      end if
    end if
  end while
  return $\varphi := \bigvee_{\theta \in F} \theta$

Algorithm 5: Predicate Abstraction for WSLTS

Chapter 6

Other Issues

Although the fact that the two procedures from the previous chapters are guaranteed to terminate is a nice theoretical result, they are not very feasible in practice. In this chapter we discuss two particular attempts to optimize them. For the first one we provide an example that shows that the optimized algorithm is not complete for the class of WSTS.

6.1 Refining Only the Spurious Error Trace

Since the construction of the abstract reachability tree is computationally expensive, a practically feasible algorithm should keep more work from previous phases of the refinement loop to the next one when this is possible. One possible improvement is to keep all parts of the subtree with root the pivot node except the spurious error trace. This means that only the regions of the nodes lying on the path between the pivot node and the error node are recomputed with respect to the new set of predicates. In addition, the nodes in this subtree that are marked as covered after the pivot node was processed should be unmarked to preserve soundness. As stated in [13] this optimization does not affect the soundness of the method, but it affects its completeness, i.e. the modified procedure might not terminate. We give an example - a WSTS together with an upward-closed set of error states for which such a modified procedure does not terminate. The example is constructed on the basis of the example in [13].

Example 6.1. Consider the program with set of guarded commands given in Table 2.1 and the corresponding labeled transition system. Let $\preceq$ be defined

as follows:

$$\langle pc_1, x_1\rangle \preceq \langle pc_2, x_2\rangle \text{ iff } pc_1 = pc_2 \text{ and } x_1 \leq x_2.$$

It is clear that $\preceq$ is a well-quasi ordering. We now show that $\preceq$ is compatible with the transition relation.

Let $s = \langle pc_1, x_1\rangle$, $t = \langle pc_2, x_2\rangle$, $u = \langle pc_3, x_3\rangle$, $s \xrightarrow{c} t$ and $s \preceq u$.

We have to consider the following cases:

- $c = c_1$
  Then $pc_1 = pc_3 = l_1$, $pc_2 = l_2$, $x_3 \geq x_1$ and $x_2 = 0$.
  Let $v = \langle l_2, 0\rangle$. Then $u \xrightarrow{c_1} v$ and $t \preceq v$.

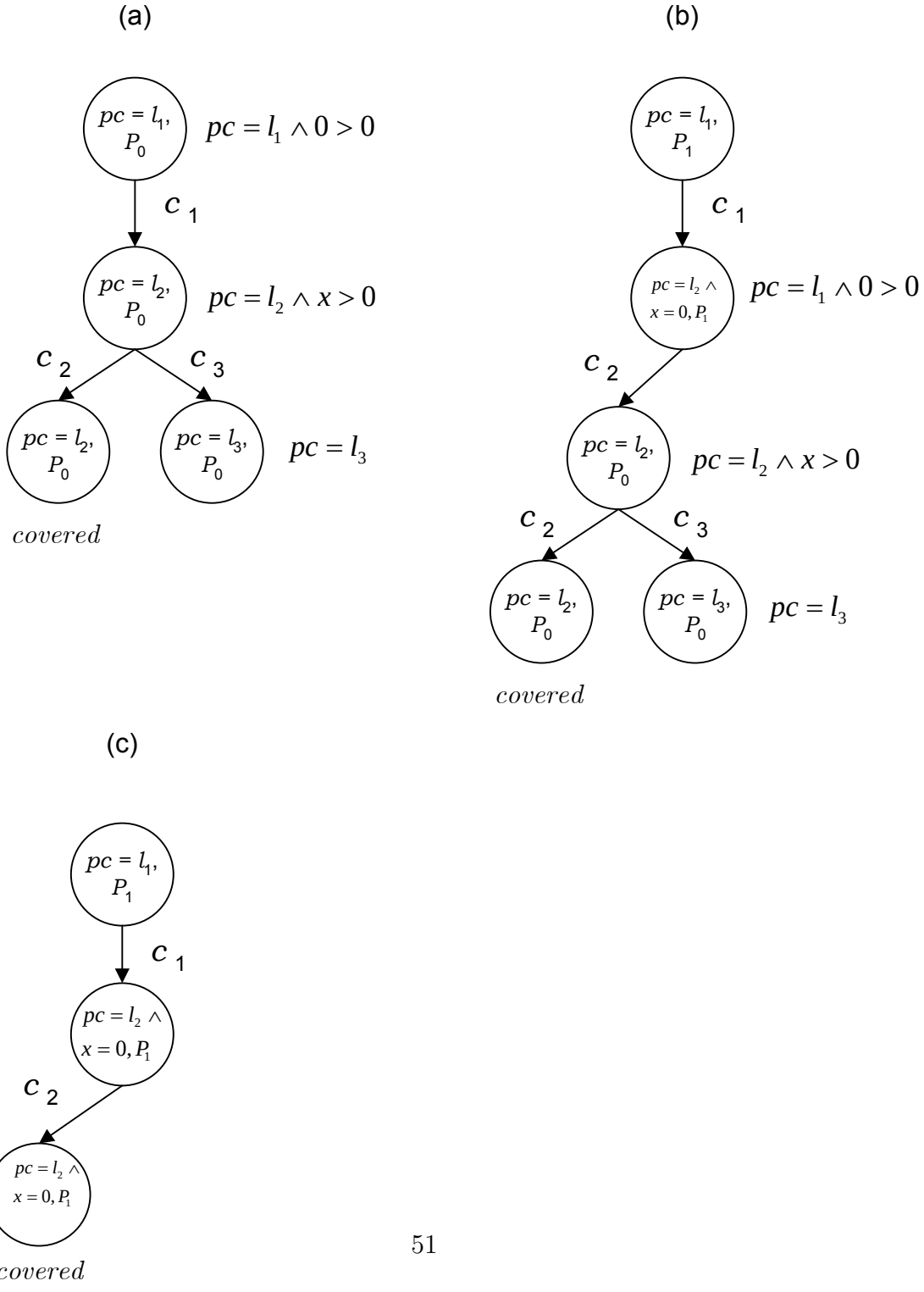
- $c = c_2$
  Then $pc_1 = pc_2 = pc_3 = l_2$, $x_1 = x_2 = 0$ and $x_3 \geq x_1$.
  Let $v = u$. Then $u \rightarrow^0 v$ and $t \preceq v$.

- $c = c_3$
  Then $pc_1 = pc_3 = l_2$, $pc_2 = l_3$, $x_3 \geq x_1 > 0$, $x_2 = x_1$.
  Let $v = \langle l_3, x_3\rangle$. Then $u \xrightarrow{c_3} v$ and $t \preceq v$.

Let $\mathsf{unsafe} = (pc = l_3)$. Obviously the set $[\![\mathsf{unsafe}]\!]$ is upward-closed.

Usually all the predicates expressing the control location are included in the initial set of predicates. Assume that the initial set of support predicates for the root is $P_0 = \{pc = l_1, pc = l_2, pc = l_3\}$. The resulting tree after the first 4 iterations (after processing the first error node) is shown on Figure 6.1 (a). On the right of the nodes we show their error regions in a simplified version dropping some disjuncts. The node labeled with $(pc = l_3, P_0)$ is an error node. The pivot node is the root. $P_1 \supseteq P_0 \cup \{x = 0\}$. Assume that only the regions of the nodes on the spurious error trace are recomputed and all other nodes in the subtree are kept unchanged. The node marked with covered should be unmarked since it is no longer covered by its parent. Then, the subtree starting from the child of the pivot node appears as the subtree with root the unmarked node. The tree after processing the next encountered error node is shown on Figure 6.1 (b). Thus, the abstract-check-refine loop never terminates. On the other hand, if the first time an error node is processed the entire subtree below the pivot node is deleted and constructed again with the new set of support predicates P_1, then the procedure terminates with the abstract reachability tree on Figure 6.1 (c).

Figure 6.1: Abstract reachability trees



From the above example we conclude that if we modify the algorithm from Chapter 4 in a way that only the regions of the nodes on the error trace are recomputed then the procedure is no longer guaranteed to terminate.

6.2 Taking into Account the Spurious Error Trace

Recall that in the algorithm in Chapter 4 the refinement step is not actually guided by the counterexample. The abstract counterexample is used to determine the pivot node, but its error region and the refinement predicates actually depend only on the length of the path from the pivot node to the error node. On the other hand, in the procedure from Chapter 5 the error region and the predicates actually depend on the sequence of commands labeling the path from the pivot node to the error node. This refinement heuristic results in the selection of more relevant predicates which is obviously a more practical approach. In this section we consider a modification of the algorithm in Chapter 4 where the spurious error trace is actually taken into account in computing the error regions and the refinement predicates. We are interested in showing the termination of this modified algorithm for the class of WSTS.

We take the following definition of error region:

Definition 6.1 (**Error region**). If $\mathbf{m}$ and $\mathbf{n}$ are nodes in some T_i with $\mathbf{m} \xrightarrow{\sigma} \mathbf{n}$ and $\mathbf{n}$ is an error node then $\mathsf{ucpre}_\sigma(\mathsf{unsafe})$ is called the *error region* of $\mathbf{m}$ for $\mathbf{n}$.

Let the set of predicates added to the set of support predicates of the pivot node consist of the negations of the atoms in the formula $\bigvee_{i=1}^{l+1} \mathsf{ucpre}_{\sigma[i,l+1)}(\mathsf{unsafe})$, where l is the length of σ. This might reduce the number of newly generated atoms, as some predicates that are not relevant are not added to the set of supports. Moreover, the pivot node determined in this way is guaranteed to be in the subtree with root the node found as a pivot node by the procedure in Chapter 4. Hence, the set of nodes deleted by the modified method is a subset of the set of nodes deleted by the method in Chapter 4, which means that the same or a bigger part of the tree from the current iteration is preserved for the next one. We have proven, that if the method proposed in Chapter 4 is optimized in such a way, no node in the

tree can be refined infinitely many times. The question whether the modified procedure is guaranteed to terminate on every WSTS and upward-closed set of error states is still open.

The following lemma and its corollary state that after refining the pivot node in the way described above, in the subsequent trees this node cannot be refined again w.r.t. the same trace. The proofs follow the lines of the ones for the similar statements from Chapter 5.

Lemma 6.2. *Let $\boldsymbol{n}{:}(\varphi, P)$ be a node and $\sigma \in \mathcal{C}^*$ such that $\mathsf{atoms}(\neg\chi_n) \subseteq P$ and $\varphi \leq \neg\mathsf{ucpre}_\sigma(\mathsf{unsafe})$ where*

$$\chi_n = \bigvee_{i=1}^{n+1} \mathsf{ucpre}_{\sigma[i,n+1)}(\mathsf{unsafe}).$$

Then

$$[\mathsf{post}^\#_\sigma((\varphi, P))] \leq \mathsf{safe}.$$

Corollary 6.3. *Let a node $\boldsymbol{n}{:}(\varphi, P)$ be refined in T_i w.r.t. a trace $\sigma \in \mathcal{C}^*$ with $\mathsf{atoms}(\neg\chi_n)$ where*

$$\chi_n = \bigvee_{i=1}^{n+1} \mathsf{ucpre}_{\sigma[i,n+1)}(\mathsf{unsafe}).$$

Let $k > i$ be such that $\boldsymbol{n}$ is not deleted in any of the trees T_j with $i < j < k$. Then for $\boldsymbol{m}{:}(\psi, Q)$ in T_k with $\boldsymbol{n} \xrightarrow{\sigma} \boldsymbol{m}$ it holds that $\psi \leq \mathsf{safe}$.

In a manner similar to the corresponding proofs from Chapter 5 we show that a node cannot be refined infinitely many times.

Lemma 6.4. *Let a node $\boldsymbol{n}$ be deleted only finitely many times, i.e. there is $k \geq 0$, such that $\boldsymbol{n}$ is not deleted from any of the trees $T_k, T_{k+1}, \ldots$. Then $\boldsymbol{n}$ is refined only a finite number of times.*

Corollary 6.5. *Every node is refined only finitely many times.*

Chapter 7

Two Families of WSTS

Several families of WSTS are used to model reactive systems in practice. Among them are timed automata, networks of timed automata, hybrid automata, FIFO channel systems, extended Petri nets, broadcast protocols and many others. In this chapter we discuss Petri nets and Lossy Channel Systems as particular examples of WSTS.

One of the main challenges in developing symbolic algorithms for a class of systems is to choose a symbolic representation of possibly infinite sets of system states. This symbolic representation should satisfy the requirements that allow for construction of a symbolic algorithm for the considered verification problem. It should be expressive enough and should allow for efficient performance of the basic operations. In our framework we assume the existence of such a symbolic representation satisfying some effectiveness requirements. The question is, whether for each family of WSTS there is an appropriate formalism. Here we address this problem for Petri nets and Lossy Channel Systems.

7.1 Petri Nets

7.1.1 Definition

Petri nets are introduced by Carl Adam Petri in 1962 in his PhD thesis. They are a well known model of concurrent systems. Here we give some important concepts from Petri net theory.

Definition 7.1 (**Net**). A *net* is a triple $\mathcal{N} = \langle P, T, F \rangle$, where

- P is a finite set of *places*,
- T is a finite set of *transitions*,
- $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ is the *flow relation*.

The *preset* of a place or transition x is ${}^\bullet x = \{y \in P \cup T | F(\langle y, x\rangle) > 0\}$. The *postset* of a place or transition x is $x^\bullet = \{y \in P \cup T | F(\langle x, y\rangle) > 0\}$. A *marking* is a mapping $M : P \to \mathbb{N}$. The configurations of a net $\mathcal{N}$ are markings.

Definition 7.2 (Petri net). A *Petri net* is a pair $N = \langle \mathcal{N}, M_0\rangle$, where $\mathcal{N}$ is a net, and M_0 is the initial marking.

Definition 7.3. Let $t \in T$.

- t is enabled at a marking M if $M(p) \geq F(\langle p, t\rangle)$ for every $p \in {}^\bullet t$.
- If t is enabled at M it can *fire* and its firing leads to a successor marking M' which is defined by

$$M'(p) = M(p) - F(\langle p, t\rangle) + F(\langle t, p\rangle).$$

  We denote this by the expression $M \xrightarrow{t} M'$.

Each Petri net defines a labeled transition system in the following way.

- The set of states is the set of markings.
- The set of initial states consists of the initial marking.
- The set of labels is the set of transitions.
- The transition relation consists exactly of the tuples $\langle M, t, M'\rangle$ such that $M \xrightarrow{t} M'$.

The simplest *ordering between markings* is the inclusion: $M \preceq M'$ when $M(p) \leq M'(p)$ for every place p. This is a well-quasi ordering according to Dickson's lemma. The transition system corresponding to a Petri net together with the ordering $\preceq$ is a WSLTS.

There are various extensions of Petri nets.

- *Petri nets with transfer arcs*: *transfer arcs* say whether the full contents of some place must be transferred to some other place.
- *Self-modifying Petri nets* the definition of which is given later.
- *Post self-modifying Petri nets*: the self-modifying extension is only allowed on post arcs (the arcs from transitions to places).

All of the above extensions fall into the class of WSTS and thus coverability is decidable for them. Most of the implemented algorithms use Karp-Miller's forward method for constructing the *coverability tree* [17] that cannot be generalized to all extensions as shown for example in [18].

7.1.2 Karp-Miller Algorithm

The Karp-Miller's tree is computed as follows.

1. Start with a tree with a single node labeled with the initial marking M_0.
2. Repeatedly pick an unprocessed leaf node labeled with M. For any transition t which is enabled at M and for which $M \xrightarrow{t} M'$ and there is no ancestor M'' of M' such that $M'' = M'$ do the following
   - for all p such that there is an ancestor M'' with $M''(p) < M'(p)$ replace $M'(p)$ by ω (where for every $n \in \mathbb{N}$, $n < \omega$),
   - create a child of the processed node labeled with M'.

The extrapolation with ω is sufficient to ensure the completeness of the method and the finiteness of the coverability tree. It means that arbitrarily large values can be reached for the particular place. The generation of ω places is precise for variants of Petri nets satisfying strict monotonicity, i.e. for $M_1 \xrightarrow{\sigma} M_2$ (where σ is a sequence of transitions) and M_3, such that $M_1 \prec M_3$ there exists M_4, such that $M_3 \xrightarrow{\sigma} M_4$ and $M_2 \prec M_4$.

But there are extensions of Petri nets such as Petri nets with reset arcs for which strict monotonicity does not hold (they satisfy only non-strict monotonicity). A *Petri net with reset arcs* is a Petri net extended with a set of *reset arcs* $R \subseteq P \times T$ from places to transitions. A reset arc from a place p to a transition t is such that the place p is reset to 0 whenever t fires. For such systems the tree constructed in this way is no longer precise. If $M'' \xrightarrow{\sigma} M'$

and $M'' \prec M'$, then there is no longer a guarantee that an arbitrarily large values can be reached for the places p for which $M''(p) < M'(p)$. Hence, the Karp-Miller tree cannot be used to decide coverability for such types of Petri nets.

7.1.3 Self-modifying Petri nets

The class of self-modifying Petri nets (SMPN) defined in [21] includes all monotonic extensions of Petri nets defined in the literature. Here we use the definition from [11].

Definition 7.4 (**SMPN**)**.** Let $P = \{p_1, \ldots, p_n\}$ and $T = \{t_1, \ldots, t_m\}$ be the sets of places and transitions respectively. The input and output effects of a transition t_i on a place p_j are given by two functions $D_{ij}^- : \mathbb{N}^n \to \mathbb{N}$ and $D_{ij}^+ : \mathbb{N}^n \to \mathbb{N}$. These functions are of the form $\alpha + \sum_{k=1}^{n} \beta_k \cdot M(p_k)$ where $\alpha, \beta_k \in \mathbb{N}$.

A transition t_i is firable from a marking M if $M(p_j) \geq D_{ij}^-(p_j)$ for every $j \in \{1, \ldots, n\}$. The successor marking M' of M under the transition t_i is computed as follows: first, we compute M'' such that for every $j \in \{1, \ldots, n\}$ $M''(p_j) = M(p_j) - {D_{ij}}^-(M)$ and then, $M'(p_j) = M''(p_j) + {D_{ij}}^+(M)$.

Definition 7.5 (**Strongly monotonic SMPN**)**.** A SMPN such that the corresponding labeled transition system equipped with the inclusion ordering $\preceq$ is a WSLTS is called *strongly monotonic SMPN*.

Almost all extensions of Petri nets defined in the literature are strongly monotonic, for example Petri nets with transfer arcs, Petri nets with reset arcs and post self-modifying Petri nets. There are exceptions such as Petri nets with non-blocking arcs [20] and lossy Petri nets [5], which we will not discuss here. However, we just mention that every Petri net with non-blocking arcs or lossy Petri net, can be transformed into a strongly monotonic SMPN equivalent to the original one w.r.t. coverability in polynomial time. The first generic complete procedure for the coverability problem for the whole class of strongly monotonic SMPN was proposed in [11].

7.1.4 Symbolic Representation

Petri nets can be represented as programs over $\mathbb{Z}$, i.e. $\mathbb{Z}$ is the domain of the data variables. The ordering $\preceq$ is a well-quasi ordering on the set of markings since markings are tuples of nonnegative integers. As a formalism for

representing sets of states we can use the theory of integer linear arithmetic. It satisfies the following conditions:

- entailment is decidable,
- the well-quasi ordering on states can be expressed as a formula,
- it admits quantifier elimination.

Hence, for Petri nets and their extensions there exists a symbolic representation that is adequate in the sense that it satisfies the effectiveness requirements discussed in the previous chapters. Since most of the extensions are strongly monotonic and hence belong to the class of WSLTS, the algorithm from Chapter 5 is a decision procedure for the coverability problem for them.

7.2 Lossy Channel Systems

One model for specifying and verifying data transfer protocols is that of *Communicating Finite State Machines*. It consists of finite-state processes that exchange messages via unbounded FIFO channels. Considering *lossy channel systems* where the channels may lose messages at any time, results in a model for which reachability is decidable. This restricted model covers a large class of communication protocols.

7.2.1 Definition

The system has a control part and a channel part. The control part is given by a location where the set of possible locations is the Cartesian product of the control states of the finite-state processes. The channel part consists of a finite set of FIFO channels, each of which contains a sequence of messages that are elements of a finite alphabet Σ. Here we follow the definition from [2].

Definition 7.6 (**Lossy channel system**)**.** A *lossy channel system* is a tuple $\mathcal{S} = \langle Q, q_i, F, \Sigma, T \rangle$ where

- Q is a finite set of locations,
- q_i is an initial location,

- F is a finite set of channels,
- Σ is a finite alphabet,
- T is a finite set of transitions, each of which is a triple of the form $\langle q_1, op, q_2 \rangle$ where q_1 and q_2 are control locations and op is a label of one of the forms:
  - $f!m$, where $f \in F$ and $m \in \Sigma$
  - $f?m$, where $f \in F$ and $m \in \Sigma$
  - nop.

With Σ^* we denote the set of finite sequences of elements of Σ. $\preceq^*$ is ordering between words, i.e. elements of Σ^*, such that $w_1 \preceq^* w_2$ iff w_1 is a (not necessarily contiguous) subword of w_2. It is well-known that $\preceq^*$ is a well quasi-ordering. The empty word is denoted by ε. With $\cdot$ we denote word concatenation.

7.2.2 Symbolic Representation

The problem with applying predicate abstraction to lossy channel systems is much harder than the one for Petri nets. The difficulty arises from the fact that we should consider programs where the domain of the variables is Σ^*. Here we propose a possible symbolic representation for lossy channel systems following the lines of [3]. We show that the set of formulas is closed under the operations pre, post and $\uparrow$.

Let $\mathcal{S} = \langle Q, q_i, F, \Sigma, T \rangle$ be a lossy channel system. With $\mathcal{S}$ we associate a program with set of variables $X = F \cup \{loc\}$, i.e. one variable corresponding to each channel in F, and one variable for the location. The domain of a variable in F is Σ^* and of loc is Q. We also assume that $Param$ is a countably infinite set of variables, which we call *parameters*, different from the program variables. We allow parameters to appear in the atomic formulas. With $x, y, \ldots$ we denote parameters and $f, g, \ldots$ stand for program variables. We put a bar to denote a tuple. The parameters appearing in a conjunction of atoms are implicitly existentially quantified, which means that $\bigwedge_{i=1}^{n} \varphi_i(\overline{f}, loc, \overline{x})$ is the same as $\exists \overline{x} \bigwedge_{i=1}^{n} \varphi_i(\overline{f}, loc, \overline{x})$, where φ_i is an atom with variables among $\overline{f}, loc$ and parameters among $\overline{x}$. We do not distinguish between two conjunctions in the case when one of them can be obtained

from the other by renaming of all parameters where different parameters are replaced by different ones.

The constraint system proposed in [3] consists of predicates (there they are called constraints) of the form φ_x where φ_x denotes the set $\{y | x \preceq^* y\}$. We consider atomic predicates of the same form where we allow x to be an *expression* that does not contain variables.

Definition 7.7. An *expression* is

- a channel variable, or
- a parameter, or
- an element of Σ^*, or
- a concatenation of expressions.

We consider *atomic predicates* of one of the following forms:

- $e_1 = e_2$ ($e_1 \neq e_2$) where e_1 and e_2 are expressions and at least one of them does not contain any program variables,
- $e_1 \preceq^* e_2$ ($e_1 \not\preceq^* e_2$) where e_1 and e_2 are expressions and e_1 does not contain any program variables,
- $loc = q$ ($loc \neq q$) where $q \in Q$.

In guarded commands we also allow predicates of the form $e_1 = e_2$ where e_1 is a primed program variable and e_2 is loc, an element of Q or an expression.

We define the set C of guarded commands for a lossy channel system $\mathcal{S} = \langle Q, q_i, F, \Sigma, T \rangle$ as the smallest set such that:

- If $\langle q_1, f!m, q_2 \rangle \in T$ then $c \in C$ where

$$c : loc = q_1 \wedge loc' = q_2 \wedge f' = f \cdot m \wedge \bigwedge_{g \in F,\, g \neq f} g' = g.$$

- If $\langle q_1, f?m, q_2 \rangle \in T$ then $c \in C$ where

$$c : loc = q_1 \wedge f = m \cdot x \wedge loc' = q_2 \wedge f' = x \wedge \bigwedge_{g \in F,\, g \neq f} g' = g.$$

- If $\langle q_1, \mathsf{nop}, q_2 \rangle \in T$ then $c \in C$ where

$$c : loc = q_1 \wedge loc' = q_2 \wedge \bigwedge_{f \in F} f' = f.$$

- If $m \in \Sigma$ then $c \in C$ where

$$c : f = x \cdot m \cdot y \wedge f' = x \cdot y \wedge \bigwedge_{g \in F,\, g \neq f} g' = g.$$

It is clear, that C is finite up to renaming of parameters.

The program $\langle X, \mathsf{init}, \delta \rangle$ representing the lossy channel system $\mathcal{S}$ is defined as follows.

- $X = F \cup \{loc\}$
- $\mathsf{init}(X) = loc = q_0 \wedge \bigwedge_{f \in F} f = \varepsilon$
- $\delta(X, X')$ is the formula describing the transition relation defined as the disjunction of all guarded commands in the set of guarded commands C as defined above.

A lossy channel system represented as a program defines a labeled transition system in the usual way. A *state* is a pair $\langle q, W \rangle$ where $q \in Q$ and $W : F \to \Sigma^*$. The *initial state* is the pair $\langle q_0, \varepsilon \rangle$, where ε is the function with $\varepsilon(f) = \varepsilon$ for every $f \in F$. The set of states of the lossy channel system we denote by S. The order $\preceq$ on states is defined as follows: for any $s = \langle q, W \rangle$ and $s' = \langle q', W' \rangle$, we have $s \preceq s'$ iff $q = q'$ and $W(f) \preceq^* W'(f)$ for all $f \in F$. For every transition system $\mathcal{S}$ that corresponds to a lossy channel system $(\mathcal{S}, \preceq)$ is a WSTS as stated in [9].

Now we show that the operators post, pre and $\uparrow$ are computable for the above symbolic representation. The only problematic issue is the quantifier elimination. Since we allowed parameters in the formulas and post, pre, $\uparrow$ distribute over disjunction, we can introduce new parameters in the process of eliminating existential quantifiers. For a formula of the form $\varphi = \bigvee_{i \in I} \bigwedge_{j \in J_i} \varphi_{ij}$ we have that $\mathsf{post}(\varphi) = \bigvee_{i \in I} \mathsf{post}(\bigwedge_{j \in J_i} \varphi_{ij})$, $\mathsf{pre}(\varphi) = \bigvee_{i \in I} \mathsf{pre}(\bigwedge_{j \in J_i} \varphi_{ij})$, $\varphi\uparrow = \bigvee_{i \in I} (\bigwedge_{j \in J_i} \varphi_{ij})\uparrow$.

Let φ be a conjunction of atoms. We define

post – if

$$c : loc = q_1 \wedge loc' = q_2 \wedge f' = f \cdot m \wedge \bigwedge_{g \in F,\, g \neq f} g' = g,$$

then $\mathsf{post}_c(\varphi) = \varphi[x/f, q_1/loc] \wedge loc = q_2 \wedge f = x \cdot m$.

– if

$$c : loc = q_1 \wedge f = m \cdot x \wedge loc' = q_2 \wedge f' = x \wedge \bigwedge_{g \in F,\, g \neq f} g' = g,$$

then $\mathsf{post}_c(\varphi) = \varphi[m \cdot x/f, q_1/loc] \wedge loc = q_2 \wedge f = x$.

– if

$$c : loc = q_1 \wedge loc' = q_2 \wedge \bigwedge_{f \in F} f' = f,$$

then $\mathsf{post}_c(\varphi) = \varphi[q_1/loc] \wedge loc = q_2$.

– if

$$c : f = x \cdot m \cdot y \wedge f' = x \cdot y \wedge \bigwedge_{g \in F,\, g \neq f} g' = g,$$

then $\mathsf{post}_c(\varphi) = \varphi[x \cdot m \cdot y/f] \wedge f = x \cdot y$.

pre – if

$$c : loc = q_1 \wedge loc' = q_2 \wedge f' = f \cdot m \wedge \bigwedge_{g \in F,\, g \neq f} g' = g,$$

then $\mathsf{pre}_c(\varphi) = loc = q_1 \wedge \varphi[q_2/loc, f \cdot m/f]$.

– if

$$c : loc = q_1 \wedge f = m \cdot x \wedge loc' = q_2 \wedge f' = x \wedge \bigwedge_{g \in F,\, g \neq f} g' = g,$$

then $\mathsf{pre}_c(\varphi) = loc = q_1 \wedge f = m \cdot x \wedge \varphi[q_2/loc, x/f]$.

– if

$$c : loc = q_1 \wedge loc' = q_2 \wedge \bigwedge_{f \in F} f' = f,$$

then $\mathsf{pre}_c(\varphi) = loc = q_1 \wedge \varphi[q_2/loc]$.

– if

$$c : f = x \cdot m \cdot y \wedge f' = x \cdot y \wedge \bigwedge_{g \in F,\ g \neq f} g' = g,$$

then $\mathsf{pre}_c(\varphi) = f = x \cdot m \cdot y \wedge \varphi[x \cdot y/f]$.

$\uparrow$ – $\varphi\uparrow = \varphi[x_1/f_1, \ldots, x_n/f_n] \wedge \bigwedge_{i=1}^{n} x_i \preceq^* f_i$.

In this definition the newly introduced parameters are chosen to be different and should not appear in φ. Furthermore, if the resulting conjunction contains a conjunct of the form $x = e$ where e does not contain program variables, x is eliminated by removing this conjunct and substituting x with e in all other conjuncts.

So, the chosen symbolic representation is adequate, in the sense that the images of the operators pre, post and $\uparrow$ are computable and are expressible in this formalism. We do not know whether entailment between formulas over this set of atoms is decidable. This question is out of the scope of this thesis.

Chapter 8

Related Work

In this chapter we review the definitions of several general classes of transition systems for which completeness results hold. We also look at the known algorithms for deciding reachability for systems in these classes and at the assumptions needed for their completeness. We start with the definitions of systems with finite bisimulation quotient and systems with finite simulation quotient and give the completeness results that are known for both of these classes. Then, we briefly describe the LazyAbstraction procedure which is guaranteed to terminate for systems with finite trace-equivalence under certain assumptions. Finally, we discuss the completeness results for the backward and forward approaches to checking coverability of WSTS, concentrating on the differences between the forward procedure presented in [10] and our approach.

8.1 Systems with Finite (Bi)simulation Quotient

Let us fix a transition system $\langle S, I, \mathcal{C}, \delta \rangle$ and a set of atoms P. For a state s with $L(s)$ we denote the set of atomic predicates which are satisfied by s, formally $L(s) = \{p \in P | s \models p\}$.

Definition 8.1 (**Simulation relation**). A relation $R \subseteq S \times S$ is a *simulation relation* on S iff for any $(s, t) \in R$ it holds that $L(s) = L(t)$ and for any u such that $s \rightarrow u$, there exists v such that $t \rightarrow v$ and $(u, v) \in R$.

Definition 8.2 (**Bisimulation relation**)**.** A relation R is a *bisimulation relation* on S iff R is symmetric and a simulation relation on S.

A state t *simulates* a state s iff (s,t) is in the greatest simulation relation on S, which exists since simulations are closed under arbitrary union. States s and t are *simulation equivalent* (written $s \sim t$) if they simulate each other. States s and t are bisimilar in $\mathcal{S}$ (written as $s \approx t$) if (s,t) is contained in the greatest bisimulation relation on S. It is clear that if s and t are bisimilar then they simulate each other. Thus, if the index of $\approx$ is finite then the index of $\sim$ is also finite.

Definition 8.3. We say that:

- $\mathcal{S}$ *has finite simulation quotient* if the index of $\sim$ is finite.
- $\mathcal{S}$ *has finite bisimulation quotient* if the index of $\approx$ is finite.

So, every transition system that has finite bisimulation quotient w.r.t. a set of atoms P has finite simulation quotient w.r.t. P. The class of systems with finite simulation quotient is a proper extension of the class of systems with finite bisimulation quotient [12].

As $\sim$ ($\approx$) is an equivalence relation, it induces a *quotient transition system* whose states are the equivalence classes of S w.r.t. $\sim$ ($\approx$). The set of initial states consists of the equivalence classes of the initial states of $\mathcal{S}$ and there is a transition (A, c, B) iff $\exists s, t: \ s \in A \wedge t \in B \wedge s \xrightarrow{c} t$.

The algorithm presented in [19] iteratively constructs a finite-state abstract program from a given possibly infinite-state concrete program by means of syntactic program transformation. The finite state abstract program has only boolean variables. They correspond to atomic predicates. This finite set of atoms is constructed iteratively, starting with the initial finite set of predicates P that consists of the atoms that appear in the specification formula and the description of the program and applying at each step the *weakest liberal precondition* operator wlp ($\mathsf{wlp}(\varphi) = \neg\mathsf{pre}(\neg\varphi)$) to the predicates in the current set. This process is repeated until no new predicates are generated or until a given iteration bound K is exceeded. In the first case the result is a bisimilar abstraction of the program and in the second it is a conservative approximation (i.e. able to match every computation of the concrete one).

Theorem 8.4. *(**Simulation Theorem**) The finite state abstract program simulates the concrete program w.r.t. the initial set of predicates P.*

Theorem 8.5. *(**Bisimulation Theorem**) If the procedure terminates because no new predicates are generated, then the abstract program is bisimulation-equivalent to the concrete one w.r.t. the initial set of predicates P.*

Moreover, if the system has finite bisimulation (simulation) quotient, then this algorithm is complete, i.e. there is a value m for the iteration bound such that if the loop is iterated m times, the algorithm is guaranteed to produce an abstract program which is bisimulation (simulation)-equivalent to the concrete one with respect to the initial set of predicates. These results are formally stated in the following two theorems.

Theorem 8.6. *(**Bisimulation Completeness**) If the concrete program has a finite reachable bisimulation quotient, then there is an appropriate choice for the iteration bound K such that the abstract program produced by the algorithm is bisimulation-equivalent to the concrete one w.r.t. the initial set of predicates P.*

Theorem 8.7. *(**Simulation Completeness**) If the concrete program has a finite simulation quotient, then there is an appropriate choice for the iteration bound K such that the abstract program produced by the algorithm is simulation-equivalent to the concrete one w.r.t. the initial set of predicates P.*

These classes of systems allow for the so called reductionist methods since they are "essentially finite-state". If we have computed a finite-state abstract system $\mathcal{A}$, which is bisimulation (simulation)-equivalent to the concrete transition system $\mathcal{S}$ with respect to the finite set of atoms P, which contains the atoms of safe, then we have that $\mathcal{A}$ is safe if and only if $\mathcal{S}$ is safe. So, in particular, the abstract program can be used to check reachability. Thus the method described in [19] can be used as a procedure for checking reachability. It employs predicate abstraction and the iterative construction of the set of predicates is actually iterative backward abstraction refinement.

Systems whose state spaces can be finitely partitioned are for example timed automata, rational relational automata and various classes of hybrid automata. Petri nets, lossy channel systems and integral relational automata do not allow for finite state partitioning.

8.2 Systems with Finite Trace-equivalence

In this section we discuss a class of systems called *systems with finite trace-equivalence* defined in [13].

8.2.1 Finite Trace-equivalence and Lazy Abstraction

Let $\langle S, I, \mathcal{C}, \delta \rangle$ be a labeled transition system.

Definition 8.8 (**Trace-equivalence**)**.** For a state $s \in S$ and $\sigma \in \mathcal{C}^*$ we write $s \xrightarrow{\sigma}$ if there is a state s' such that $s \xrightarrow{\sigma} s'$. Two states s and t are *trace-equivalent* if for every $\sigma \in \mathcal{C}^*$ we have $s \xrightarrow{\sigma}$ iff $t \xrightarrow{\sigma}$.

Definition 8.9. The labeled transition system $\mathcal{S}$ has *finite trace-equivalence* if the trace-equivalence relation on S has a finite index.

The *ascending chain condition* is that there does not exist an infinite strictly increasing sequence $\varphi_0 < \varphi_1 < \ldots$ of formulas.

LazyAbstraction is an algorithm, proposed in [13], for verifying reachability by means of predicate abstraction. It constructs the abstract reachability tree in a forward manner. Whenever an abstract counterexample is produced, it is analyzed backwardly by iterating pre. When the set of support predicates of the pivot node is refined, only enough atoms are added as to exclude the particular counterexample. A heuristic function is used to determine whether the subtree below the pivot node will be deleted or only the error trace will be refined. The LazyAbstraction procedure is guaranteed to terminate under certain conditions, provided that the heuristic always determines that the whole subtree below the pivot node should be deleted. These are:

- the system has finite trace-equivalence, and
- the ascending chain condition is satisfied, and
- error states have no outgoing transitions.

8.2.2 Finite Trace-equivalence and WSLTS

In this section we study the relationship between the two classes of systems-the class of systems that have finite trace-equivalence as defined in [13] and

the class of WSLTS. First we show that every labeled transition system $\mathcal{S}$ with finite trace-equivalence can be equipped with a well-quasi ordering $\preceq$ such that $(\mathcal{S}, \preceq)$ is a WSLTS under the assumption that for each $c \in \mathcal{C}$ the relation $\xrightarrow{c}$ is deterministic.

Proposition 8.10. *Let $\mathcal{S} = \langle S, I, \mathcal{C}, \delta \rangle$ be a labeled transition system that has finite trace-equivalence quotient. Assume that for each $c \in \mathcal{C}$ the relation $\xrightarrow{c}$ is deterministic. Then there is a well-quasi ordering $\preceq$ on S such that $(\mathcal{S}, \preceq)$ is a WSLTS.*

Proof. With $s_1 \simeq s_2$ we denote the fact that s_1 and s_2 are trace-equivalent. We define $\preceq$ as follows:

$$s_1 \preceq s_2 \text{ iff for every } \sigma \in \mathcal{C}^*,\ s_1 \xrightarrow{\sigma} \text{ implies } s_2 \xrightarrow{\sigma}.$$

Since $\simeq$ is an equivalence relation with finite index, it is clear that $\preceq$ is a well-quasi ordering.

We now check strong compatibility w.r.t. the labels of the transitions. Let $s_1, s_2, t_1 \in S$, $c \in \mathcal{C}$, $s_1 \preceq s_2$ and $s_1 \xrightarrow{c} t_1$. Since $s_1 \preceq s_2$, there is a state t_2, such that $s_2 \xrightarrow{c} t_2$. Let $t_1 \xrightarrow{\sigma}$. Then, since each command in $\mathcal{C}$ is deterministic, there is a unique state u_1 such that $t_1 \xrightarrow{\sigma} u_1$. Then, $s_1 \xrightarrow{c\sigma} u_1$. Since $s_1 \preceq s_2$, $s_2 \xrightarrow{c\sigma}$. Then, there exist states t' and u_2, such that $s_2 \xrightarrow{c} t' \xrightarrow{\sigma} u_2$. But since $\xrightarrow{c}$ is deterministic it holds that $t_2 = t'$. Hence, $t_2 \xrightarrow{\sigma}$ and thus, $t_1 \preceq t_2$. □

We have shown that every transition system that has finite trace- equivalence can be turned into a WSLTS by equipping it with a proper well-quasi ordering. In the general case this well-quasi ordering is not expressible with a formula and depends on the particular system. Observe that for the algorithm in Chapter 5 we do not require the existence of a formula $\varphi_{\preceq}$ corresponding to the ordering $\preceq$. This is because the existence of a well-quasi ordering is used as a termination argument for the procedure and this ordering is not part of the procedure itself. This means that the algorithm in Chapter 5 is guaranteed to terminate on every transition system with finite trace equivalence, which is not surprising.

Now we show that the class of WSLTS is strictly more general than the class of systems with finite trace-equivalence by giving an example of a WSLTS $\mathcal{S}$ which does not have finite trace-equivalence.

Example 8.1 (System that does not have finite trace-equivalence). Consider the transition system represented by the program with set of guarded

commands given in Table 8.1. The domain of x is $\mathbb{Z}$, i.e. the set of states S is $\{l_1, l_2\} \times \mathbb{Z}$. The set of initial states is denoted by the formula $\mathsf{init} = (x = -1 \wedge pc = l_1)$ and the set of error states is denoted by the formula $\mathsf{unsafe} = (pc = l_2)$.

	Guard	**Updates**
c_1	$pc = l_1 \wedge x \neq 0$	$pc' = pc, x' = x - 1$
c_2	$pc = l_1 \wedge x \geq 0$	$pc' = l_2, x' = x$

Table 8.1: Program that does not have finite trace-equivalence quotient

It is easy to see that the transition system does not have finite trace-equivalence. Consider the states $s_i = \langle pc = l_1, x = i \rangle$ where $i \geq 1$ and the traces $\sigma_i = {c_1}^i c_2$ for $i = 1, 2, \ldots$. For $i < j$ we have that $s_j \xrightarrow{\sigma_j}$, but $s_i \not\xrightarrow{\sigma_j}$ which implies that s_i and s_j are not trace-equivalent.

Now we define an ordering $\preceq$ on S, such that $\mathcal{S}$ together with $\preceq$ is a WSLTS. First we define an ordering on $\mathbb{Z}$ in the following way. Let $m, n \in \mathbb{Z}$, $m \sqsubseteq n$ iff

- $m \geq 0$, $n \geq 0$, and $m \leq n$ or
- $m < 0$, $n < 0$ and $n \leq m$.

Let $\preceq$ be defined as follows:

$$\langle pc_1, x_1 \rangle \preceq \langle pc_2, x_2 \rangle \text{ iff } pc_1 = pc_2 \text{ and } x_1 \sqsubseteq x_2.$$

It is clear that $\preceq$ is a well-quasi ordering.

We now show that $(\mathcal{S}, \preceq)$ satisfies strong compatibility w.r.t. the labels of the transitions. Let $s = \langle pc_1, x_1 \rangle$, $t = \langle pc_2, x_2 \rangle$, $u = \langle pc_3, x_3 \rangle$, $s \xrightarrow{c} t$ and $s \preceq u$. We distinguish between the following two cases.

- $c = c_1$
  Then $pc_1 = pc_2 = pc_3 = l_1$, $x_1 \sqsubseteq x_3$, $x_1 \neq 0$. Let $v = \langle l_1, x_3 - 1 \rangle$.
  If $x_1 \geq 0$ then $x_3 \geq x_1 > 0$ and thus $x_1 - 1 \sqsubseteq x_3 - 1$. If $x_1 < 0$ then $x_3 \leq x_1$ and thus $x_1 - 1 \sqsubseteq x_3 - 1$. Then $u \xrightarrow{c_1} v$ and $t \preceq v$.
- $c = c_2$
  Then $pc_1 = pc_3 = l_1$, $pc_2 = l_2$, $x_1 \geq 0$, $x_3 \geq x_1 \geq 0$ and $x_2 = x_1$. Let $v = \langle l_2, x_3 \rangle$. Then $u \xrightarrow{c_2} v$ and $t \preceq v$.

So, this transition system is a WSLTS, but it does not have finite trace-equivalence.

8.2.3 Backward Counterexample Analysis for WSTS

Here we give an example of a WSTS such that counterexample analysis based on iteration of pre does not terminate, but counterexample analysis based on ucpre terminates with success.

Example 8.2. The guarded commands of the program are given in Table 8.2.

	Guard	**Updates**
c_1	$pc = l_1$	$pc' = pc, x' = x + 1$
c_2	$pc = l_1 \wedge x \neq 1$	$pc' = pc, x' = x - 1$
c_3	$pc = l_1 \wedge x = 0$	$pc' = l_2, x' = x$

Table 8.2: Guarded commands of the program for which refinement with pre does not terminate

The domain of x is $\mathbb{Z}$. The set of initial states is given by the formula $\mathsf{init} = (pc = l_1 \wedge x = 1)$ and the set of error states by $\mathsf{unsafe} = (pc = l_2)$. The transition system denoted by the program above can be easily equipped with an ordering on the set of states that turns it into a WSTS.

First we define a well-quasi ordering on $\mathbb{Z}$ in the following way. Let $m, n \in \mathbb{Z}$, then $m \sqsubseteq n$ iff

- $m > 0$ and $n > 0$ and $m \leq n$ or
- $m \leq 0$ and $n \leq 0$ and $n \leq m$.

Then we define the ordering $\preceq$ between states as

$$\langle pc_1, x_1 \rangle \preceq \langle pc_2, x_2 \rangle \text{ iff } pc_1 = pc_2 \text{ and } x_1 \sqsubseteq x_2.$$

It is clear that $\preceq$ is a well-quasi ordering that is expressible with a formula and the set of error states is upward-closed w.r.t $\preceq$. Now we show compatibility. Let $s = \langle pc_1, x_1 \rangle$, $t = \langle pc_2, x_2 \rangle$, $u = \langle pc_3, x_3 \rangle$, $s \xrightarrow{c} t$ and $s \preceq u$.

We have to distinguish between the following cases:

- $c = c_1$
  Then $pc_1 = pc_2 = pc_3 = l_1$ and $x_2 = x_1 + 1$.

If $x_1 > 0$, then $x_3 \geq x_1$. Let $v = \langle l_1, x_3 + 1\rangle$. Then $u \xrightarrow{c_1} v$ and $t \preceq v$.
If $x_1 \leq 0$, then $x_3 \leq x_1$. If $x_1 = 0$, then $x_2 = 1$, so let $v = \langle l_1, 1\rangle$. Then $u \xrightarrow{c_1}{}^{1-x_3} v$ and $t \preceq v$. If $x_1 < 0$, then $x_3 \leq x_1$ and $x_2 \leq 0$, so let $v = \langle l_1, x_3 + 1\rangle$. Then $u \xrightarrow{c_1} v$ and $t \preceq v$.

- $c = c_2$
  Then, $pc_1 = pc_2 = pc_3 = l_1$, $x_1 \neq 1$ and $x_2 = x_1 - 1$.
  If $x_1 > 0$ then $x_3 \geq x_1$ and $x_3 \neq 1$. If $x_1 \leq 0$ then $x_3 \leq x_1$ and $x_3 \neq 1$.
  Let $v = \langle l_1, x_3 - 1\rangle$. Then, $u \xrightarrow{c_2} v$ and $t \preceq v$.

- $c = c_3$
  Then $pc_1 = pc_3 = l_1$, $pc_2 = l_2$, $x_1 = x_2 = 0$, $x_3 \leq x_1$. Let $v = \langle l_2, 0\rangle$. Then $u \xrightarrow{c_1}{}^{-x_3} \xrightarrow{c_3} v$ and $t \preceq v$.

Here we would like to point out that $(\mathcal{S}, \preceq)$ is not a WSLTS. This can be seen by considering the states $s = \langle l_1, 0\rangle$, $t = \langle l_2, 0\rangle$ and $u = \langle l_1, -3\rangle$. We have that $s \xrightarrow{c_3} t$ and $s \preceq u$, but there does not exist a state v such that $t \preceq v$ and $u \xrightarrow{c_3} v$.

During the construction of the abstract reachability tree in the case when the pre operator is used for counterexample analysis the procedure generates infinitely many spurious counterexamples that correspond to the unfolding of a loop. To exclude those abstract counterexamples the refinement based on pre generates infinitely many atomic predicates.

Let $P_0 = \{pc = l_1, pc = l_2, x = 1\}$.
In the following table we give the reachable regions of the nodes along the first encountered spurious counterexample together with their error regions. From the error regions we have dropped some disjuncts, but their intersection with the corresponding reachable region is empty.

Reachable region		Error region	
(init, P_0)	$pc = l_1 \wedge x = 1, P_0$	$\mathsf{pre}_{c_1}(pc = l_1 \wedge x = 0)$	$pc = l_1 \wedge x = -1$
$\mathsf{post}^{\#}_{c_1}((\mathsf{init}, P_0))$	$pc = l_1, P_0$	$\mathsf{pre}_{c_3}(\mathsf{unsafe})$	$pc = l_1 \wedge x = 0$
$\mathsf{post}^{\#}_{c_3}((pc = l_1, P_0))$	$pc = l_2, P_0$	unsafe	$pc = l_2$

The pivot node is the root and it is refined with the predicates in $P_1 = P_0 \cup \{x \neq 0, x \neq -1\}$ (for simplicity we drop $pc \neq l_1, pc \neq l_2$). Then the following spurious counterexample is produced.

Reachable region		Error region	
(init, P_1)	$pc = l_1 \wedge x = 1, P_1$	$\mathsf{pre}_{c_1}(pc = l_1 \wedge x = -2)$	$pc = l_1 \wedge x = -3$
$\mathsf{post}^{\#}_{c_1}((\mathsf{init}, P_1))$	$pc = l_1 \wedge x \neq 0 \wedge x \neq -1, P_1$	$\mathsf{pre}_{c_1}(pc = l_1 \wedge x = -1)$	$pc = l_1 \wedge x = -2$
$\mathsf{post}^{\#}_{c_1}((pc = l_1 \wedge x \neq 0 \wedge x \neq -1, P_1))$	$pc = l_1 \wedge x \neq 0, P_1$	$\mathsf{pre}_{c_1}(pc = l_1 \wedge x = 0)$	$pc = l_1 \wedge x = -1$
$\mathsf{post}^{\#}_{c_1}((pc = l_1 \wedge x \neq 0, P_1))$	$pc = l_1, P_1$	$\mathsf{pre}_{c_3}(\mathsf{unsafe})$	$pc = l_1 \wedge x = 0$
$\mathsf{post}^{\#}_{c_3}((pc = l_1, P_1))$	$pc = l_2, P_1$	unsafe	$pc = l_2$

Again the root node is refined with $P_2 = P_1 \cup \{x \neq -2, x \neq -3\}$. In the following two tables the reachable and the error regions of the nodes along the next counterexample are shown.

Reachable region	
(init, P_2)	$pc = l_1 \wedge x = 1, P_2$
$\mathsf{post}^{\#}_{c_1}((\mathsf{init}, P_2))$	$pc = l_1 \wedge x \neq 0 \wedge x \neq -1 \wedge x \neq -2 \wedge x \neq -3, P_2$
$\mathsf{post}^{\#}_{c_1}((pc = l_1 \wedge x \neq 0 \wedge x \neq -1 \wedge x \neq -2 \wedge x \neq -3, P_2))$	$pc = l_1 \wedge x \neq 0 \wedge x \neq -1 \wedge x \neq -2, P_2$
$\mathsf{post}^{\#}_{c_1}((pc = l_1 \wedge x \neq 0 \wedge x \neq -1 \wedge x \neq -2, P_2))$	$pc = l_1 \wedge x \neq 0 \wedge x \neq -1, P_2$
$\mathsf{post}^{\#}_{c_1}((pc = l_1 \wedge x \neq 0 \wedge x \neq -1, P_2))$	$pc = l_1 \wedge x \neq 0, P_2$
$\mathsf{post}^{\#}_{c_1}((pc = l_1 \wedge x \neq 0, P_2))$	$pc = l_1, P_2$
$\mathsf{post}^{\#}_{c_3}((pc = l_1, P_2))$	$pc = l_2, P_2$

Error region	
$\mathsf{pre}_{c_1}(pc = l_1 \wedge x = -2)$	$pc = l_1 \wedge x = -5$
$\mathsf{pre}_{c_1}(pc = l_1 \wedge x = -1)$	$pc = l_1 \wedge x = -4$
$\mathsf{pre}_{c_1}(pc = l_1 \wedge x = 0)$	$pc = l_1 \wedge x = -3$
$\mathsf{pre}_{c_1}(pc = l_1 \wedge x = 0)$	$pc = l_1 \wedge x = -2$
$\mathsf{pre}_{c_1}(pc = l_1 \wedge x = 0)$	$pc = l_1 \wedge x = -1$
$\mathsf{pre}_{c_3}(\mathsf{unsafe})$	$pc = l_1 \wedge x = 0$
unsafe	$pc = l_2$

$P_3 = P_2 \cup \{x \neq -4, x \neq -5\} \ldots$.

We observe that each time new atomic predicates are needed to exclude the current spurious counterexample that corresponds to an unfolding of the loop. So, the process diverges, although the program under consideration is correct.

If instead of pre the operator ucpre is used, during the analysis of the first encountered counterexample the procedure computes:

Reachable region		Error region	
(init, P_0)	$pc = l_1 \wedge x = 1, P_0$	$\mathsf{pre}(pc = l_1 \wedge x \leq 0)\uparrow$	$(pc = l_1 \wedge x \leq -1) \vee (pc = l_1 \wedge x \leq 0)$
$\mathsf{post}^{\#}_{c_1}((\mathsf{init}, P_0))$	$pc = l_1, P_0$	$\mathsf{pre}(pc = l_2)\uparrow$	$pc = l_1 \wedge x \leq 0$
$\mathsf{post}^{\#}_{c_3}((pc = l_1, P_0))$	$pc = l_2, P_0$	unsafe	$pc = l_2$

The root node is refined with $P_1 = P_0 \cup \{x > 0, x > -1\}$

Then for the abstract iteration after this refinement we have:

$$\mathsf{post}^{\#}_{c_1}((pc = l_1 \wedge x = 1, P_1)) = (pc = l_1 \wedge x > 0 \wedge x > -1, P_1)$$
$$\mathsf{post}^{\#}_{c_1}((pc = l_1 \wedge x > 0 \wedge x > -1, P_1)) = (pc = l_1 \wedge x > 0 \wedge x > -1, P_1)$$
$$\mathsf{post}^{\#}_{c_2}((pc = l_1 \wedge x > 0 \wedge x > -1, P_1)) = (pc = l_1 \wedge x > 0 \wedge x > -1, P_1)$$
$$\mathsf{post}^{\#}_{c_3}((pc = l_1 \wedge x > 0 \wedge x > -1, P_1)) = \mathsf{false}$$

Thus, Algorithm 3 terminates with success.

8.3 Well-structured Transition Systems

In this section we discuss the existing decision procedures for the coverability problem for the class of WSTS, which as we showed in the previous section is a proper extension of the class of transition systems with finite trace equivalence.

8.3.1 The Coverability Problem

The *coverability problem* for WSTS as defined in [10] is the following: *given an upward-closed set* bad*, does it hold that* $\mathit{lfp}(\mathsf{post}, I) \cap \mathsf{bad} = \emptyset$. For WSTS this problem is known to be decidable, i.e. reachability of an upward-closed set is decidable. There are generally to approaches to checking coverability - backward and forward.

Backward Iteration

There exists a generic backward algorithm for solving the coverability problem for all families of WSTS. The algorithm computes over finite representations of upward-closed sets of states. Every upward-closed set can be

represented by a finite set of minimal elements. This finite representation is adequate since union and intersection are effective. As an effectiveness assumption it is required that given a finite set M that represents an upward-closed set U it is possible to compute a finite set of minimal elements M' representing the upward-closure of $\mathsf{pre}(U)$. If this holds then the procedure iterating $\mathsf{min} \circ \uparrow \circ \mathsf{pre} \circ \uparrow$ (where $\mathsf{min}(A)$ returns a minor set of A) starting from a minor set of the upward-closed set bad is guaranteed to terminate. The result is a finite set of elements, the upward-closure of which is the set all states that can reach a state in bad.

Forward Iteration

In [11] a forward algorithm for the coverability problem is proposed. It constructs two sequences of abstractions of the set of reachable states of the system, one from below and one from above. The sequence of abstractions from below allows to detect cases when a state in bad is reachable. They are bounded iterations of post starting from the initial state. The abstractions from above are iterations of overapproximations of post that become more and more precise. This sequence allows to decide instances of the problem when bad is not reachable. The proposed algorithm is actually a general schema, and to be applied to a particular class of WSTS an *adequate domain of limits* has to be provided. This is, in fact, a set of abstract values that allows to represent any downward-closed set. In the forward algorithm downward-closed sets are approximation of the set of reachable states.

In [10] two of these authors proposed a generic and effective representation of downward-closed sets. The representation is formalized as a generic abstract domain. The abstract domain is automatically refined until a sufficiently precise overapproximation of the set of reachable states, which allows to decide the coverability problem, is obtained.

8.3.2 The Forward Algorithm

Here we briefly review the abstract interpretation framework and the completeness result from [10]. The forward approach to coverability is based on the notion of *covering set* - $Cover(S)$ which is the downward-closure of the set of reachable states in $\mathcal{S}$. Provided that bad is upward-closed it enjoys the following property: $\mathsf{lfp}(\mathsf{post}, I) \cap \mathsf{bad} = \emptyset$ if and only if $Cover(S) \cap \mathsf{bad} = \emptyset$.

Hence, in order to check the emptiness condition $\mathsf{lfp}(\mathsf{post}, I) \cap \mathsf{bad} = \emptyset$ it suffices to check that $Cover(S) \cap \mathsf{bad} = \emptyset$.

For an upward-closed set $A \subseteq S$ min is an operator such that $\mathsf{min}(A)$ is a minor set of A and $\mathsf{minpre}(A) = \mathsf{min}(\mathsf{pre}(A\uparrow)\uparrow)$.

The abstract domain

Overapproximations of the set of reachable states are downward-closed sets. Since they might be infinite, abstraction is needed. The abstract domain is parameterized by a finite set of states $D \subseteq S$. The abstract lattice is $\mathcal{L}(DCS(D))$ (where $DCS(D)$ is the set of all downward-closed subsets in D). $\subseteq$ is the partial order and $\cup_D$ and $\cap_D$ are the least upper bound and the greatest lower bound operators respectively. Since D is finite, $\cup_D, \cap_D$ are effective and $\subseteq_D$ is decidable. The abstraction of a set A consists of those elements of its downward-closure which are elements of D. The abstraction and concretization functions are defined as follows:

$$\begin{array}{ll} \text{For } A \subseteq S & \alpha_D(A) = A\downarrow \cap D \\ \text{For } B \in DCS(D) & \gamma_D(B) = \{s \in S | s\downarrow \cap D \subseteq B\} \end{array}$$

The pair of functions α_D and γ_D form a Galois insertion.

This representation is adequate since every downward-closed set can be exactly represented in some properly chosen abstract domain (a set A is represented exactly if $\gamma_D(\alpha_D(A)) = A$). For each $A \in DCS(S)$ if we define D to be $\mathsf{min}(S \backslash A)$, then A is represented exactly in $\mathcal{L}(DCS(D))$. Thus, the abstract framework described above provides an effective way to represent downward-closed sets. The more elements the set D has, the more downward-closed sets have exact representation.

Abstract interpretation

The abstract operator $\mathsf{post}^\#_D$ is defined as:

$$\mathsf{post}^\#_D = \alpha_D \circ \mathsf{post} \circ \gamma_D.$$

It satisfies the property $\mathsf{lfp}(\mathsf{post}, I) \subseteq \gamma_D(\mathsf{lfp}(\mathsf{post}^\#_D, \alpha_D(I)))$, i.e. the concretization of the set of abstract reachable states is an overapproximation of the set of concrete reachable states. If the WSTS is effective ($\preceq$ and the transition relation are decidable and minpre is computable) then the set of abstract reachable states $\mathsf{lfp}(\mathsf{post}^\#_D, \alpha_D(I))$ can be effectively computed.

Refinement

In order to obtain a complete method for solving the coverability problem, the abstract domain should be iteratively refined until an abstraction that is precise enough is obtained. If the abstract domain provides exact representation of $Cover(S)$, then the concretization of the set of abstract reachable states is exactly $Cover(S)$. The covering set is downward-closed and thus it can be represented exactly using a well-chosen finite D. To obtain an abstraction refinement procedure that is guaranteed to terminate, it suffices to ensure that at some point it considers an abstract domain $\mathcal{L}(DSC(D))$ in which $Cover(S)$ is exactly represented. If the sequence of finite sets of states produced by the iterative refinement is $D_0, D_1, \ldots$, for each D_j the algorithm checks whether there is a trace that consists only of elements of D_j starting with an initial state and reaching an error state. This guarantees termination of the algorithm in the case when $\mathsf{lfp}(\mathsf{post}, I) \cap \mathsf{bad} \neq \emptyset$.

In [10] two methods for abstraction refinement are proposed. Here we briefly discuss both of them.

1. **Enumeration of the finite subsets of S**
   If S is enumerable then we can enumerate the finite subsets of S: $D_0, D_1, \ldots$. There exists an index i, such that D_i provides exact representation of $Cover(S)$. So in the case when $\mathsf{lfp}(\mathsf{post}, I) \cap \mathsf{bad} = \emptyset$ the algorithm is guaranteed to terminate.

2. **Eliminating overapproximations leading to bad**
   If the finite set D contains the elements of $\mathsf{min}(\mathsf{lfp}(\mathsf{pre}, \mathsf{bad}))$ then the overapproximation is "good enough", i.e. if the concrete system is safe, so is the abstract one. Thus, termination of the procedure in the case when the transition system is safe is guaranteed when the iterative refinement eventually constructs such a finite set D. To this end at each refinement step the procedure adds more states to the current D_{i-1} - enough to exclude all spurious counterexamples of length i. The set of elements that are added at the i-th iteration is a minor set of the upward-closure of the set that consists of all abstract reachable states that can reach a state in bad in at most i steps. This guarantees that at some point the elements of $\mathsf{min}(\mathsf{lfp}(\mathsf{pre}, \mathsf{bad}))$ are added to D, since $\mathsf{lfp}(\mathsf{pre}, \mathsf{bad}) = (\mathsf{lfp}(\mathsf{minpre}, \mathsf{bad}))\uparrow$.

The key difference between the second approach described above and our approach is the representation of the sets of states. In our case a possibly

infinite set of states is finitely represented by a formula. Thus not only the problem with the effective representation of infinite state sets is solved, but we can also make use of predicate abstraction and various heuristics for predicate refinement. In the procedures from the previous chapters this heuristic is examination of abstract counterexamples by computing weakest preconditions.

In our approach we have different precision of the abstractions for different parts of the state space. Refinement predicates are added lazily on demand only to the part of the reachability tree where they are actually needed. Thus we not only do not construct the whole abstract state space from scratch every time, but also parts of the reachability tree that can be verified in the coarser abstraction are not refined. Thus, predicate abstraction probably allows for more efficient algorithms for checking coverability.

Chapter 9

Conclusion

9.1 Results

In this thesis we study the coverability problem for WSTS, i.e. reachability of a set of states which is upward-closed according to a well-quasi ordering on the set of states of the system. This problem is known to be decidable for this class of systems. We investigate the conditions under which abstract forward iteration based on predicate abstraction and refinement results into a decision procedure for coverability in WSTS. We show that in the standard schema the refinement step can be performed in a way that guarantees convergence of the abstract-check-refine loop under these conditions. For the two classes of systems WSTS and WSLTS we provide two instantiations and prove the completeness of the resulting algorithms.

In order to apply predicate abstraction as a complete abstract interpretation framework for coverability of WSTS we make the following assumptions:

- we have a symbolic representation of the transition system (it is given as a program),
- the well-quasi ordering is expressible (a formula $\varphi_{\preceq}$ exists),
- the set of initial states and the set of error states can be expressed as formulas,
- the underlying logical theory used for symbolic representation admits quantifier elimination.

We also study the relationship between the two classes of well-structured systems and the class of systems with finite-trace equivalence as defined in [13]. We show that every transition system with finite trace-equivalence can be turned into a WSLTS and give an example of a WSLTS that does not have finite trace-equivalence. Hence, the procedure in Chapter 5 is guaranteed to terminate on all instances of the class of systems with finite trace-equivalence.

9.2 Open Problems

There are some open questions that are out of the scope of this thesis.

- One of the assumptions that we made is that we have a symbolic representation that satisfies several requirements that permit the construction of an effective procedure. The question whether for each particular family of WSTS a logical formalism that satisfies these requirements exists is still to be studied. For the family of monotonic extensions of Petri nets this is the case.
- There are other heuristics for predicate selection for example based on interpolation that also suffer from the problem of divergence in the general case. The question is whether there are reasonable conditions under which we can obtain termination guarantee for procedures for checking coverability for WSTS that employ such heuristics.

Bibliography

[1] P. A. Abdulla, K. Cerans, B. Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *LICS '96: Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, page 313, Washington, DC, USA, 1996. IEEE Computer Society.

[2] Parosh Abdulla and Bengt Jonsson. Verifying Programs with Unreliable Channels. *Inf. Comput.*, 127(2):91–101, 1996.

[3] Parosh Abdulla and Bengt Jonsson. Ensuring completeness of symbolic verification methods for infinite-state systems. *Theor. Comput. Sci.*, 256(1-2):145–167, 2001.

[4] Thomas Ball, Andreas Podelski, and Sriram K. Rajamani. Relative Completeness of Abstraction Refinement for Software Model Checking. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 158–172, 2002.

[5] Ahmed Bouajjani and Richard Mayr. Model Checking Lossy Vector Addition Systems. *Lecture Notes in Computer Science*, 1563:323–333, 1999.

[6] Patrick Cousot and Radhia Cousot. Systematic Design of Program Analysis Frameworks. In *POPL*, pages 269–282, 1979.

[7] L.E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n prime factors. *Amer. J. Math.*, 35:413–422, 1913.

[8] A. Finkel. A Generalization of the Procedure of Karp and Miller to Well-Structured Transition Systems. In *ICALP*, pages 499–508, 1987.

[9] A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.

[10] Pierre Ganty, Jean-François Raskin, and Laurent Van Begin. A complete abstract interpretation framework for coverability properties of WSTS. In *Proc. of Verification, Model Checking and Abstract Interpretation (VMCAI)*, volume 3855 of *Lecture Notes in Computer Science*, pages 49–64. Springer-Verlag, 2006.

[11] Gilles Geeraerts, Jean-François Raskin, and Laurent Van Begin. Expand, Enlarge and Check: new algorithms for the coverability problem of WSTS. *Journal of Computer and System Sciences, volume 72(1), pp 180–203*, 2005.

[12] R.J. van Glabbeek. *Comparative Concurrency Semantics and Refinement of Actions*. PhD thesis, Free University, Amsterdam, 1990. Second edition available as *CWI tract* 109, CWI, Amsterdam 1996.

[13] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Gregoire Sutre. Lazy Abstraction. In *Symposium on Principles of Programming Languages*, pages 58–70, 2002.

[14] Thomas A. Henzinger, Rupak Majumdar, and Jean-François Raskin. A classification of symbolic transition systems. *ACM Trans. Comput. Logic*, 6(1):1–32, 2005.

[15] Graham Higman. Ordering by divisibility in abstract algebras. *Proc. London. Math.*, 2:326–336, 1952.

[16] Ranjit Jhala and Kenneth L. McMillan. A Practical and Complete Approach to Predicate Refinement. In *TACAS*, pages 459–473, 2006.

[17] Richard M. Karp and Raymond E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969.

[18] Michael Leuschel and Helko Lehmann. Coverability of Reset Petri Nets and Other Well-Structured Transition Systems by Partial Deduction. *Lecture Notes in Computer Science*, 1861:101+, 2000.

[19] Kedar S. Namjoshi and Robert P. Kurshan. Syntactic Program Transformations for Automatic Abstraction. In *CAV '00: Proceedings of the 12th International Conference on Computer Aided Verification*, pages 435–449, London, UK, 2000. Springer-Verlag.

[20] Jean-François Raskin and Laurent Van Begin. Petri Nets with Non-blocking Arcs are Difficult to Analyze.

[21] R. Valk. On the Computational Power of Extended Petri Nets. *Mathematical Foundations of Computer Science 1978*, pages 526–535, 1978.