Solving conjunctive and disjunctive parameterized Boolean equation systems using SMT solvers

Ruud Koolen

August 28, 2014
Abstract

In this paper, we consider methods for solving model checking problems expressed as parameterized Boolean equation systems symbolically by making use of SMT solvers. By unrolling the PBES and expressing relevant properties of that unrolling as an SMT proposition, the solution to the model checking problem expressed by a PBES can be computed by an SMT solver. Based on this technique, we present two algorithms for symbolically solving a class of model checking problems called conjunctive and disjunctive PBESs. By avoiding the need to instantiate the state space, these algorithms can efficiently solve problems with enormous or infinite state spaces.
# Contents

1 Introduction  

2 Background and related work  
   2.1 Fixpoint equation systems  
   2.2 Parameterized Boolean equation systems  
   2.3 Satisfiability modulo theories  
   2.4 Related work  

3 Conjunctive and disjunctive PBESs  
   3.1 Disjunctive PBESs  
   3.2 Witnesses  
   3.3 Acyclic unrollings  
   3.4 Conjunctive PBESs  

4 Unrolling in SMT  
   4.1 Encoding unrollings in SMT  
   4.2 Finding witnesses  
   4.3 The absence of witnesses  
   4.4 Deciding $[E](X)(d)$  
   4.5 Structured domains  
   4.6 Results  

5 Partial instantiation  
   5.1 Symbolic unrollings  
   5.2 Algorithm sketch  
   5.3 Exploring symbolic unrollings  
   5.4 Algorithm details  
   5.5 Results  

6 Conclusion and future work  

A Example problem details
Chapter 1

Introduction

*Model checking* consists of computing for a model of a given system, such as a piece of software or a networking protocol, whether it adheres to a certain behavioral property. For example, for a messaging protocol, one might want to determine whether each message sent is eventually received by the intended receiver. Often, the property one wants to check has the form of checking that the system never ends up in some type of “bad state”, or checking that the system always ends up in a “good state” after something unusual happens.

Traditionally, problems like these are solved by computing a list of states in the system, followed by computing how avoidable or unavoidable particular types of states are. The problem with this approach is that the number of states in the system may well be extremely large, which usually makes computing all states in the system infeasible. In those cases, the only practical solution is to symbolically analyze the system, reasoning out the truth or falsity of the property one is interested in.

SMT solvers are one kind of general tool for this kind of symbolic reasoning, capable of computing whether an object exists matching some kind of property. While SMT solvers are not specialized for model checking applications, they form a potent tool for answering questions such as whether a given system contains a reachable “bad state”.

In this paper, we present two techniques for solving model checking problems in a symbolic way using SMT solvers as the primary symbolic reasoning system, focussing on model checking problems encoded as parameterized Boolean equation systems (PBESs). We define a fragment of such problems called conjunctive and disjunctive PBESs, and we present two distinct algorithms based on SMT solvers for solving model checking problems in this fragment. As symbolic methods, these algorithms are able to solve a range of problems that are out of reach for existing methods.

In Chapter 2, we introduce the background theory behind the results in this paper, and introduce relevant notation. In Chapter 3, we define the concept of conjunctive and disjunctive PBESs, and prove basic theory that underlies the two algorithms for solving them. In Chapters 4 and 5 we respectively introduce the *smt-unrolling* and *partial-instantiation* algorithms for solving conjunctive and disjunctive PBESs symbolically using SMT solvers. Finally, in Chapter 6 we reflect on these results, and discuss possibilities for future research.
Chapter 2

Background and related work

Techniques for model checking can be classified into two main approaches: the so-called explicit state and symbolic model checking techniques. In explicit state model checking, the investigated system is represented by a finite directed graph known as a state space, representing the different states the system can be in together with the ways the system can change from one state to another. In symbolic model checking, the system is represented by a limited collection of algebraic equations, the solutions to which implicitly define such a state space (which need not be finite).

In [Mad97], Mader introduced the formalism of Boolean equation systems for encoding explicit-state model checking problems. When used in such a way, for a given behavioral property and a state space representing a system model, a Boolean equation system encodes the given property as applied to the state space; the model checking problem can then be solved by solving the resulting Boolean equation system.

Parameterized Boolean equation systems, defined by Groote and Willemse in [GW04] based on [GM99], are a symbolic extension of Boolean equation systems. Parameterized Boolean equation systems encode symbolic model checking problems analogously to how Boolean equation systems encode explicit-state model checking problems, consisting of equations whose solutions implicitly define a Boolean equation system.

The satisfiability modulo theories or SMT problem is the problem of computing for a particular first-order logical theory whether that theory has a model. That is, for a given set of function symbols and a given set of first-order axioms over those function symbols, the problem consists of determining whether an interpretation of those function symbols exists such that the axioms hold. Roughly speaking, the satisfiability modulo theories problem is a first-order generalization of the Boolean satisfiability problem, which is the special case where all function symbols are booleans (that is, functions from the one-point domain to the booleans) and all axioms are propositional.

The SMT problem is undecidable in general, as problems such as the halting problem can easily be expressed in it. However, algorithms exist that can
efficiently solve particular fragments of it, and various tools (known as SMT solvers) exist that are quite effective at solving practical SMT problems.

This paper describes two techniques for solving symbolic model checking problems encoded as parameterized Boolean equation systems by utilizing the symbolic reasoning capabilities of SMT solvers. The remainder of this chapter defines Boolean and parameterized Boolean equation systems in detail and establishes basic properties. It also describes the SMT problem, as well as the capabilities and limitations of existing SMT solvers.

2.1 Fixpoint equation systems

Let $A$ be a complete lattice, and let $X$ be a set of $A$-valued variable symbols. Then a fixpoint equation system over $A$, as defined in [Mad97], is a finite sequence of equations of the form $(\sigma X = \varphi)$, where $\sigma$ is a fixpoint symbol $\mu$ or $\nu$, $X \in X$ is a fixpoint variable, and $\varphi : A^n \rightarrow A$ for some $n \in \mathbb{N}$ is a monotone function of a set of fixpoint variables.

A fixpoint equation system is valid if each fixpoint variable occurs at the left hand side of at most one equation; it is closed if each fixpoint variable that occurs in the equation system also occurs at the left hand side of at least one equation. In this paper, only valid closed fixpoint equation systems are considered.

**Example 1.** The system $(\nu X = Y \lor Z)(\mu Y = Y \land Z)(\nu Z = X)$ is a valid and closed fixpoint equation system over the lattice of the Booleans.

The empty fixpoint equation system is denoted as $\epsilon$. For a fixpoint equation system including the equation $(\varsigma X = \varphi)$, $\varphi$ is called the definition of $X$; the fixpoint symbol $\varsigma$ is denoted as $\sigma(X)$.

In the context of a given fixpoint equation system $\mathcal{E}$, we write $X \triangleleft Y$ iff the equation in $\mathcal{E}$ with $X$ as the left hand side precedes the equation with $Y$ as the left hand side; that is, iff $\mathcal{E}$ is of the form $[\mathcal{E}_1(\sigma_1 X = \varphi)][\mathcal{E}_2(\sigma_2 Y = \varphi)][\mathcal{E}_3]$. We write $X \sqsubseteq Y$ iff $X \triangleleft Y$ or $X = Y$.

An environment $\theta : X \rightarrow A$ is a valuation function for fixpoint variables. For a given environment $\theta$ and a function $\varphi, \varphi(\theta)$ is the value of the function $\varphi$ after substituting each fixpoint variable $X$ occurring in $\varphi$ by $\theta(X)$.

The solution $[\mathcal{E}]\theta$ for fixpoint equation system $\mathcal{E}$ relative to environment $\theta$ is the environment defined as follows:

\[
\begin{align*}
[\epsilon] \theta & \equiv \theta \\
[(\mu X = \varphi)\mathcal{E}] \theta & \equiv [\mathcal{E}] \theta[X := \mu X.\varphi([\mathcal{E}] \theta)] \\
[(\nu X = \varphi)\mathcal{E}] \theta & \equiv [\mathcal{E}] \theta[X := \nu X.\varphi([\mathcal{E}] \theta)] \\
\mu X.\varphi([\mathcal{E}] \theta) & \equiv \{a \in A \mid a \geq \varphi([\mathcal{E}] \theta[X := a])\} \\
\nu X.\varphi([\mathcal{E}] \theta) & \equiv \{a \in A \mid a \leq \varphi([\mathcal{E}] \theta[X := a])\}
\end{align*}
\]

For a closed fixpoint equation system $\mathcal{E}$, the two environments $\theta_1$ and $\theta_2$ are equivalent for $\mathcal{E}$ if $\theta_1(X) = \theta_2(X)$ for all fixpoint variables $X$ that occur in $\mathcal{E}$. Since for any closed fixpoint equation system $\mathcal{E}$ and environments $\theta_1$ and $\theta_2$ it holds that $[\mathcal{E}] \theta_1$ and $[\mathcal{E}] \theta_2$ are equivalent for $\mathcal{E}$, we speak of the solution $[\mathcal{E}]$ of $\mathcal{E}$ independent of any specific environment $\theta$. 

3
The most common type of fixpoint equation systems are the *Boolean equation systems*, which are fixpoint equation systems over the usual lattice of the Booleans.

**Example 2.** The system \((\nu X = Y \lor Z)(\mu Y = Y \land Z)(\nu Z = X)\) is a Boolean equation system, with solution \(\{X \rightarrow \text{true}, Y \rightarrow \text{false}, Z \rightarrow \text{true}\}\).

Boolean equation systems can be used to express explicit-state model checking problems. In this scheme, a model checking problem is encoded as a Boolean equation system \(E\) together with a designated variable \(X_0\) of \(E\), such that the solution of the model checking problem is equal to the value of \(X_0\) in the solution of \(E\), denoted as \([E](X_0)\). The model checking problem can then be solved by solving the corresponding Boolean equation system.

Infinite Boolean equation systems, defined in \[Mad97\], form a straightforward extension to Boolean equation systems in which the number of equations, and the number of parameters of a given right hand side \(\phi\), may be infinite. They are isomorphic to the fixpoint equation systems over the lattice of infinite products of the Booleans. In this paper, we will use the concepts of Boolean equation systems and infinite Boolean equation systems interchangeably, and will use the term *BES* to refer to either.

### 2.2 Parameterized Boolean equation systems

A *parameterized Boolean equation system* or PBES over some nonempty domain \(D\) is a fixpoint equation system over the lattice of first-order predicates over \(D\). As a visual reminder that fixpoint variables in parameterized Boolean equation systems represent predicates over \(D\), PBES equations are denoted as \((\sigma X(d:D) = \phi)\).

**Example 3.** The system \((\nu X(n:\mathbb{N}) = X(n+1) \lor Y(n))(\mu Y(n:\mathbb{N}) = Y(n+1) \lor 5 < n < 10)\) is a PBES with domain \(\mathbb{N}\), with solution \(\{X(n) \rightarrow \text{true}, Y(n) \rightarrow n < 10\}\).

Just like Boolean equation systems are used to encode explicit-state model checking problems, PBESs can be used to encode symbolic model checking problems. In this encoding, the model checking problem is represented by a PBES \(\mathcal{E}\) over domain \(D\), a fixpoint variable \(X_0\) of \(\mathcal{E}\), and domain value \(d_0 \in D\), such that the solution of the model checking problem is equal to \([\mathcal{E}](X_0)(d_0)\).

Let \(\mathcal{E}\) be a PBES over domain \(D\), and let \(\mathcal{E}^*\) be a BES such that the following hold:

- the equations of \(\mathcal{E}^*\) are the equations of the form \((\sigma X^d = \phi^*)\), where \((\sigma X(d:D) = \phi)\) is an equation in \(\mathcal{E}\), \(d \in D\), and \(\phi^*\) is \(\phi\) with each fixpoint variable expression \(Y(f(d))\) replaced by \(Y^f(d)\);

- the equation ordering is such that \(X^d \sqsubset Y^e\) in \(\mathcal{E}^*\) for \(X \neq Y\) if and only if \(X \sqsubset Y\) in \(\mathcal{E}\).

Then \(\mathcal{E}^*\) is called an *instantiation* of \(\mathcal{E}\), as defined in \[DPW08\].

By construction, if \(\mathcal{E}_1^*\) and \(\mathcal{E}_2^*\) are both instantiations of \(\mathcal{E}\), then \(\mathcal{E}_1^*\) and \(\mathcal{E}_2^*\) can differ only in the relative order of different equations based on the same...
equation in $\mathcal{E}$. Because by Lemma 3.21 in [Mad97] the relative order of such equations does not affect the solution of $\mathcal{E}^*$, we speak of the instantiation $\mathcal{E}^*$ of $\mathcal{E}$.

The following property holds for the instantiation of a PBES:

**Theorem 1.** For PBES $\mathcal{E}$ over domain $D$, fixpoint variable $X(d:D)$ of $\mathcal{E}$, and first-order value $d \in D$, $[\mathcal{E}](X)(d) = [\mathcal{E}^*](X^d)$.

**Proof.** By Theorem 3 of [DPW08].

For a PBES $\mathcal{E}$, the fixpoint variables of $\mathcal{E}^*$ are called the states of $\mathcal{E}$. A state $X^d$ occurs in state $Y^e$ if the variable $X^d$ occurs in the definition of $Y^e$, after application of applicable Boolean simplifications. A sequence of states each occurring in the previous one is called an unrolling; in particular, a finite sequence $[X_0^{d_0}, \ldots, X_n^{d_n}]$ of $\mathcal{E}$ such that $X_i^{d_i+1}$ occurs in $X_i^{d_i}$ for each $i < n$ is called an $n$-step unrolling of $X_0^{d_0}$. If such an unrolling exists, $X_n^{d_n}$ is called reachable from $X_0^{d_0}$.

**Example 4.** In the PBES described in Example 3, $X^3$ and $Y^7$ are states. The states occurring in $X^3$ are $X^4$ and $Y^3$. No states occur in $Y^7$, because its definition $Y^8 \lor 5 < 7 < 10$ simplifies to true.

### 2.3 Satisfiability modulo theories

The **Boolean satisfiability problem** or **SAT problem** is the problem of deciding, for a propositional formula over Boolean variables $B_1, \ldots, B_n$, whether there is an assignment of Boolean values to those variables such that the formula is true. The Boolean satisfiability problem is NP-complete, but algorithms known as SAT solvers exist that can efficiently solve many practical SAT problems.

The **satisfiability modulo theories problem** or **SMT problem** is a first-order generalization of the SAT problem: it is the problem of determining the satisfiability of a first-order formula over a set of first-order variables. Since this problem is expressive enough to express problems such as the halting problem, the SMT problem is undecidable in general.

Like the SAT problem, the SMT problem can be solved efficiently for many practical instances by algorithms known as SMT solvers. Many fragments of the SMT problem – that is, versions of the problem using a restricted vocabulary for the formula to satisfy, and therefore having limited expressive power – are satisfiable, and SMT solvers tend to be quite effective at recognizing problem instances that are in some decidable fragment. Key restrictions typically required are that formulae may not contain quantifiers, variables may only be of particular domains, and the available operations are limited.

In this paper, we model SMT solvers as a facility that can determine whether a given closed Boolean formula is true or false, as long as that formula contains only existential quantifiers which must occur positively. Exact restrictions of domains and function symbols varies considerably between different SMT solvers, and are therefore not considered in this paper.
2.4 Related work

*Bounded model checking* is the technique of solving model checking problems by successively unrolling the transition relation underlying a given model checking problem, and using a SAT solver to investigate the proposition that this unrolling leads to a state providing a counterexample to the model checking problem under investigation. Surveyed by Clarke, Biere, Raimi, and Zhu in [CBRZ01], the technique of bounded model checking is strongly related to the approaches used in this paper.

Yet, there are some important differences between bounded model checking as described in the paper above, and the techniques discussed in this paper. Bounded model checking is commonly based on SAT solvers as the tool of choice for deciding the reachability of a state matching a certain predicate; a consequence of this is that these techniques can only work with model checking problems for which all domains are finite. Problems in which for example real numbers play a major role are outside the scope of this form of bounded model checking, and bounded model checking using SMT solvers as its vehicle is not commonly practiced.

Established model checking tools exist whose working is based on bounded model checking using satisfiability solving; a prominent example is the NuSMV symbolic model checker [CCG+02]. However, these tools are limited to fairly restricted logics, such as CTL and LTL, which limits there applicability.

The contributions presented in this paper try to extend these techniques, both to a broader set of expressible properties, and to settings involving more powerful data domains than those usable when using existing techniques. With any luck, this will considerably increase the set of problems that can be analyzed using this technique.
Chapter 3

Conjunctive and disjunctive PBESs

In Chapters 4 and 5, two techniques are presented for using SMT solvers to solve a restricted class of PBESs that we call conjunctive and disjunctive PBESs. In this chapter we develop these notions, and show basic properties underlying the algorithms in later chapters.

3.1 Disjunctive PBESs

Intuitively, a disjunctive PBES $E$ is a PBES such that its instantiation $E^*$ contains only disjunctions of fixpoint variables in the right hand sides. Transforming this to a syntactic requirement on PBESs, we say a PBES is disjunctive if each right hand side consists of a disjunction of existential quantifications over guarded fixpoint expressions. Specifically:

**Definition 1.** For $d$ in some domain $D$, a disjunctive clause over $d$ is an expression of the form

$$\left( \exists e \in E P(d, e) \land X(f(d, e)) \right)$$

where $E$ is some domain, $P$ is a predicate over $D \times E$, $X$ is a fixpoint variable, and $f$ is a function from $D \times E$ to $D$.

Clearly, for a given $d \in D$ and a disjunctive clause, rewriting the clause’s existential quantification over $E$ to a (possibly infinite) disjunction and expanding $P(d, e)$ for each such term yields a disjunction over fixpoint expressions. This allows us to formally define a disjunctive PBES as follows:

**Definition 2.** A disjunctive PBES is a PBES whose right hand sides consist of disjunctions of disjunctive clauses. That is, it is a PBES of the form

$$E \equiv \left( \sigma, X_i(d: D) = \bigvee_{(E, P, X, f) \in \psi(X_i)} \left( \exists e \in E P(d, e) \land X(f(d, e)) \right) \right)_i$$

where $\psi(X_i)$ represents the set of disjunctive clauses in the definition of $X_i$, encoded as a set of $(E, P, X, f)$-tuples each making up a disjunctive clause.
In this definition, \( E \) may be the one-point set, which means the existential quantification can be trivial. Since the constant true can be encoded as a fixpoint variable \( X_{\text{true}} \) defined in the equation \( (\nu X_{\text{true}} = X_{\text{true}}) \), the disjunctive clauses of \( E \) may include plain predicates of \( d \).

**Example 5.** The PBES

\[
(\nu X(n;N) = (\exists m \in N : m < 10 \land X(m)) \lor n = 20)
\]

is not technically disjunctive, but the straightforward reformulation

\[
(\nu X_1(n;N) = (\exists m \in N : m < 10 \land X_1(m)) \lor (\exists z \in \{\star\} : n = 20 \land X_2))
\]

is disjunctive.

**Remark.** The mapping of plain predicates onto guards of the invocation of \( X_{\text{true}} \) simplifies the analysis of disjunctive PBESs, but may cause needless inefficiencies in any practical implementation of the techniques presented here. The possible optimization of encoding plain predicates in a more direct manner is straightforward, and therefore not detailed in this paper.

### 3.2 Witnesses

If a given PBES is disjunctive, then its unrolling can efficiently be solved using the method of Gaussian elimination, described in Section 6.4 of [Mad97]. For such a PBES \( E \), the fact that \( [E](X)(d) = \text{true} \) for some state \( X \) can be shown by exhibiting a sequence of substitutions and eliminations to perform, represented by a single unrolling of \( X \).

**Lemma 2.** Let \( E \) be a disjunctive PBES, let \( X \) be a state of \( E \) with \( \sigma(X) = \nu \), and let \( [X_0^{d_0}, \ldots, X_n^{d_n}] \) be an unrolling of at least one step such that \( X_0^{d_0} = X_n^{d_n} = X \) and \( X_i \leq X_i \) for all \( i \leq n \). Then \([E](X)(d) = \text{true}\).

**Proof.** Because the definition of each \( X_i^{d_i} \) with \( i < n \) is of the form \( X_i^{d_{i+1}} \lor \varphi \), substitution of the definitions of \( X_i^{d_i} \) up to \( X_1^{d_1} \) in the definition of \( X_0^{d_0} \) yields a formula of the form \( X_i^{d_i} \lor \varphi \). In particular, if \( i = n - 1 \), because \( X_0^{d_0} = X_n^{d_n} = X \), this yields a definition for \( X_i^{d_i} \) of the form \( X \lor \varphi \).

Let the BES \( \mathcal{F} \) be \( \mathcal{E}^* \) with the definition of \( X \) replaced with the formula \( X \lor \varphi \), as produced by the substitution process above. Because \( X \leq X_i \) for all \( i < n \), by Lemma 6.3 of [Mad97], \( \mathcal{F}[\mathcal{E}^*] = [\mathcal{E}^*] \). But because \( \sigma(X) = \nu \), by Lemma 6.2 of [Mad97], \( [\mathcal{E}^*](X^d) = \text{true} \). Therefore, by Theorem 1, \( [\mathcal{E}](X)(d) = [\mathcal{E}^*](X) = [\mathcal{E}^*](X^d) = \text{true} \). \( \Box \)

**Theorem 3.** Let \( E \) be a disjunctive PBES, let \( X \) and \( Y \) be states of \( E \) such that \( \sigma(Y) = \nu \), let \( [X_0^{d_0}, \ldots, X_n^{d_n}] \) be an unrolling such that \( X_0^{d_0} = X^{d_0} \) and \( X_n^{d_n} = Y^e \), and let \( [Y_0^{e_0}, \ldots, Y_m^{e_m}] \) be an unrolling of at least one step such that \( Y_0^{e_0} = Y_n^{e_n} = Y^e \) and \( Y \leq Y_i \) for all \( i \leq m \). Then \( [\mathcal{E}](X)(d) = \text{true} \).

**Proof.** By Lemma 2, \( [\mathcal{E}](Y)(e) = \text{true} \). Because \( X_i^{d_i} \) for \( i < n \) are all disjunctions, backwards substitution gives \( X^d = X_0^{d_0} = \text{true} \), and thus (by Theorem 1) \( [\mathcal{E}](X)(d) = \text{true} \). \( \Box \)
The two unrollings required by Theorem 3 have the property that the last state of the first unrolling coincides with the first state of the second unrolling. It follows that the two unrollings can exist if and only if a single unrolling exists containing both unrollings as subsequences, which we call a witness. Specifically:

**Definition 3.** Let $E$ be a disjunctive PBES, and let $X^d$ be a state of $E$. Then a **witness** for $X^d$ is an unrolling of the form $[X^d]S_1[Y^e]S_2[Y^e]S_3$, such that $\sigma(Y) = \nu$ and $Y \subseteq Z$ for all $Z^l \in S_2$.

Clearly, a witness for state $X^d$ of disjunctive PBES $E$ exists if and only if Theorem 3 is satisfied for $E$ and $X^d$.

A witness is **minimal** if it does not contain any superfluous cycles that can be removed while maintaining the witness; that is, a witness is not minimal if it has the form $[X^d]S_1S_2S_3$ for nonempty $S_2$ such that $[X^d]S_1S_3$ is also a witness, and minimal otherwise. Because any state that has a witness also has a minimal witness, the absence of witnesses for a given state can be shown by proving the absence of any minimal witnesses for that state.

3.3 Acyclic unrollings

Theorem 3 states that if a witness exists for state $X^d$ of disjunctive PBES $E$, then $[E](X)(d) = true$. The converse of this theorem does not hold in general; if $[E](X)(d) = true$ for some disjunctive PBES $E$ and state $X^d$, no witness for $X^d$ need exist. For example, in the disjunctive PBES $(\nu X(n:N) = X(n + 1))$, no witness exists for any state, yet the solution of that PBES for $X^0$ is true. For this to happen, however, the PBES needs to have an infinite unrolling of $X^d$ consisting of distinct states.

Let an **acyclic unrolling** of state $X^d$ of PBES $E$ be an unrolling such that each state occurs in it no more than once. Then if the length of acyclic unrollings of $X^d$ is bounded, the converse of Theorem 3 does hold:

**Theorem 4.** Let $E$ be a disjunctive PBES, let $X^d$ be a state of $E$, and let $n$ be a natural number such that each unrolling of $X^d$ of more than $n$ states contains at least one state multiple times. Furthermore, let $[E](X)(d) = true$. Then a witness for $X^d$ exists.

**Proof.** The states reachable from $X^d$ form a BES $B$. Therefore, by Corollary 3.37 and Theorem 9.4 of [Mad97], there is a BES $B'$ with $[B'] = [B]$ such that $B'$ is $B$ with each equation $(\sigma X = \phi)$ replaced by $(\sigma X = Z)$, where $Z$ is one of the variables occurring in $\phi$. Then there is a single infinite sequence $[Z_0^f, Z_1^f, \ldots]$ such that $Z_0^f = X^d$ and $Z_i^{f+1}$ occurs in $Z_i^f$ for all $i$. Because the subsequence $[Z_0^f, \ldots, Z_n^{f+1}]$ is also an unrolling of $X^d$ in $E$, it contains at least one duplicate state; therefore, the complete sequence consists of a finite prelude $[Z_0^f, \ldots, Z_k^f]$ followed by repeated copies of the finite cycle $[Z_{k+1}^f, \ldots, Z_l^f]$.

Let $p$ be the number in $(k, l]$ such that $Z_p^f \leq Z_i^f$ for $k < i \leq l$. Then $[Z_0^f, \ldots, Z_p^{f+1}]$ is an unrolling of $X^d$ such that $Z_p^f = Z_{p+l-k}^f$ and $Z_p^f \leq Z_i^f$ for $p \leq i \leq p + l - k$. Thus, all that remains to be proven is to show that this unrolling is a witness is that $\sigma(Z_p) = \nu$. 


3.4 Conjunctive PBESs

Disjunctive PBESs have a dual notion which we call conjunctive PBESs. Just like a disjunctive PBES is a PBES whose instantiation contains only disjunctions of fixpoint variables in the right hand sides, a PBES is conjunctive if the right hand sides of the instantiation consist only of conjunctions of fixpoint variables.

Definition 4. For $d$ in some domain $D$, a conjunctive clause over $d$ is an expression of the form

$$\left( \forall_{e \in E} P(d, e) \Rightarrow X(f(d, e)) \right)$$

where $E$ is some domain, $P$ is a predicate over $D \times E$, $X$ is a fixpoint variable, and $f$ is a function from $D \times E$ to $D$.

Definition 5. A conjunctive PBES is a PBES whose right hand sides consist of conjunctions of conjunctive clauses. That is, it is a PBES of the form

$$\mathcal{E} \equiv \left( \sigma_i, X_i(d; D) = \bigwedge_{(E, P, X, f) \in \psi(X_i)} \left( \forall_{e \in E} P(d, e) \Rightarrow X(f(d, e)) \right) \right)_i$$

where $\psi(X_i)$ represents the set of conjunctive clauses in the definition of $X_i$, encoded as a set of $(E, P, X, f)$-tuples each making up a conjunctive clause.

Dual versions of Theorems 3 and 4 hold for conjunctive PBESs, whose proofs are completely analogous to the originals and therefore omitted.

Theorem 5. Let $\mathcal{E}$ be a conjunctive PBES, let $X^d$ and $Y^e$ be states of $\mathcal{E}$ such that $\sigma(Y) = \mu$, let $[X^d_0, \ldots, X^d_n]$ be an unrolling such that $X^d_0 = X^d$ and $X^d_n = Y^e$, and let $[Y^e_0, \ldots, Y^e_m]$ be an unrolling of at least one step such that $Y^e_0 = Y^e_m = Y^e$ and $Y \leq Y_i$ for all $i \leq m$. Then $[\mathcal{E}](X)(d) = \text{false}$.

Theorem 6. Let $\mathcal{E}$ be a conjunctive PBES, let $X^d$ be a state of $\mathcal{E}$, and let $n$ be a natural number such that each unrolling of $X^d$ of more than $n$ states contains at least one state multiple times. Furthermore, let $[\mathcal{E}](X)(d) = \text{false}$. Then a witness for $X^d$ exists.

The algorithms and analyses described in the following chapters generally deal with disjunctive PBESs only. Due to duality, all of them have obvious analogous versions dealing with conjunctive PBESs, but for the sake of brevity and simplicity these analyses are omitted from the remaining parts of this paper.
Chapter 4

Unrolling in SMT

The model checking problem for PBESs is the problem of determining, for a PBES $E$ over domain $D$, variable $X$ of $E$, and value $d \in D$, whether $[E](X)(d) = true$. For disjunctive PBESs, the notion of witnesses as defined in Section 3.2 suggests an obvious approach for solving this problem using SMT solvers: the question whether a witness for $X^d$ exists in $E$ can simply be expressed as an SMT problem, and solved as such.

Unfortunately, the proposition that some witness for a given PBES state exists is problematic to express as an SMT problem; something that can instead be expressed is the proposition that a witness for a state exists of a certain length. Solving this problem for increasing lengths will then eventually find any existing witness. Showing that a witness does not exist is a different matter which cannot be demonstrated using this technique; after all, a witness could always exist of a length larger than any length tested so far. Therefore, an independent method for finding out when to stop searching for ever-larger witnesses is necessary to complete the above technique.

In this chapter, we develop an algorithm for solving model checking problems consisting of disjunctive PBESs using the approach described above. In Section 4.1, we describe how to express the existence of unrollings of $X^d$ of a given length in terms of an SMT problem; Section 4.2 extends this to the existence of a witness for $X^d$ of a given length, and Section 4.3 describes how to show that no witness for $X^d$ exists for any length larger than some $n$. Based on these last two results, Section 4.4 crystalizes this into an algorithm for deciding whether $[E](X)(d) = true$, and discusses the strengths and limitations of this technique. In Section 4.5, we discuss an optimization of the above algorithm based on efficiently encoding particular data domains in SMT problems; finally, Section 4.6 presents and analyzes the results of applying this algorithm to a collection of practical model checking problems.

4.1 Encoding unrollings in SMT

An unrolling for a PBES $E$ consists of a sequence of states of $E$, for which each pair of consecutive states matches some predicate expressing that the latter state occurs in the definition of the former. Thus, to express the concept of
unrollings for a PBES in terms of an SMT problem, one needs some encoding of PBES states in terms of SMT variables, as well as some way to express the occurs-in predicate for a pair of states as an SMT proposition.

For a PBES $\mathcal{E}$ over domain $D$, a state $X^d$ of $\mathcal{E}$ is characterized by the combination of its fixpoint variable $X$ of $\mathcal{E}$, and the first-order value $d : D$. Since each fixpoint variable $X$ of $\mathcal{E}$ occurs as the left hand side of exactly one equation of $\mathcal{E}$, we can identify the fixpoint variable $X$ with the equation number $T$ of its defining equation; in that case, we denote $X$ as $\chi(T)$. In this way, a state $\chi(T)^d$ of $\mathcal{E}$ can be encoded as a pair $(T, d)$; we will use these two notations interchangeably. An unrolling $[X_0^d, \ldots, X_n^d]$ of $n$ steps can then be encoded as a set of variables $T_i, d_i$ for $0 \leq i \leq n$, with $(T_i, d_i)$ encoding the $i$th state in the unrolling.

Using this encoding, for a disjunctive PBES $\mathcal{E}$, the proposition that state $(T_b, d_b)$ occurs in state $(T_a, d_a)$ can be encoded as a disjunction over all fixpoint variables that $T_a$ could identify and all fixpoint variable invocations that occur in the accompanying definition:

**Definition 6.** For a disjunctive PBES $\mathcal{E}$ as defined in Definition 2 and states $(T_a, d_a)$ and $(T_b, d_b)$ of $\mathcal{E}$, we define the SMT proposition $\text{occurs}(T_a, d_a, T_b, d_b)$ as

$$\text{occurs}(T_a, d_a, T_b, d_b) \equiv \bigvee_{i < |\mathcal{E}|} \bigvee_{(E, P, \chi(T), f) \in \psi(\chi(i))} \left( (T_a = i) \land (T_b = T) \land \exists e \in E [P(d_a, e) \land (d_b = f(d_a, e))] \right)$$

**Lemma 7.** For a disjunctive PBES $\mathcal{E}$ and states $(T_a, d_a)$ and $(T_b, d_b)$ of $\mathcal{E}$, the state $(T_b, d_b)$ occurs in $(T_a, d_a)$ if and only if $\text{occurs}(T_a, d_a, T_b, d_b) = \text{true}$.

**Proof.** Assume $(T_b, d_b)$ occurs in $(T_a, d_a)$. Then by the definition of $\mathcal{E}$, $(\exists e \in E P(d_a, e) \land d_b = f(d_a, e))$ holds for some $(E, P, \chi(T_b), f) \in \psi(\chi(T_a))$. Thus, $\text{occurs}(T_a, d_a, T_b, d_b)$ is true for $i = T_a$, $T = T_b$, and the given values of $E$, $P$, and $f$.

Conversely, assume that $\text{occurs}(T_a, d_a, T_b, d_b) = \text{true}$. Because $i = T_a$ and $T = T_b$, $(\exists e \in E P(d_b, e) \land d_b = f(d_b, e))$ holds for some $(E, P, \chi(T_b), f) \in \psi(\chi(T_a))$. Therefore, $(T_b, d_b)$ occurs in $(T_a, d_a)$. \qed

With occurs defined as above, an SMT proposition expressing that the sequence $[(T_0, d_0), \ldots, (T_n, d_n)]$ forms an unrolling follows naturally:

**Definition 7.** For a disjunctive PBES $\mathcal{E}$ and states $(T_0, d_0), \ldots, (T_n, d_n)$ of $\mathcal{E}$, we define the SMT proposition $\text{unrolling}_n([(T_0, d_0), \ldots, (T_n, d_n)])$ as

$$\text{unrolling}_n([(T_0, d_0), \ldots, (T_n, d_n)]) \equiv \bigwedge_{i < n} \text{occurs}(T_i, d_i, T_{i+1}, d_{i+1})$$

Clearly, $\text{unrolling}_n([(T_0, d_0), \ldots, (T_n, d_n)]) = \text{true}$ if and only if the sequence $[(T_0, d_0), \ldots, (T_n, d_n)]$ forms an $n$-step unrolling of $(T_0, d_0)$. 

12
4.2 Finding witnesses

For a disjunctive PBES $\mathcal{E}$ and state $X^d$ of $\mathcal{E}$, a witness for $X^d$ is an unrolling of the shape $[X^d[S_1][Y^e][S_2][Y^e][S_3]$ for some state $Y^e$ with $\sigma(Y) = \nu$, such that $Y \triangleq Z$ for all $Z^j \in S_2$. For a given unrolling of $X^d$, the property of having such a shape is easy to recognize: an unrolling $[X^d_0, \ldots, X^d_n]$ is a witness for $X^d_0$ if and only if there exist $k$, $m$ with $k < m \leq n$ such that $X^d_k = X^d_m$, $\sigma(X_k) = \nu$, and $X_k \triangleq X_i$ for $k < i < m$.

With that in mind, the property for a given sequence of states of being a witness for its first state can be expressed as an SMT property, using an existential quantification over $k$ and $m$ as well as additional variables $(T^*, d^*)$ representing both $X^d_k$ and $X^d_m$:

**Definition 8.** For a disjunctive PBES $\mathcal{E}$ and states $(T_0, d_0), \ldots, (T_n, d_n)$ of $\mathcal{E}$, we define the SMT proposition $\text{witness}_n((T_0, d_0), \ldots, (T_n, d_n))$ expressing the claim that $[(T_0, d_0), \ldots, (T_n, d_n)]$ is a witness for $(T_0, d_0)$ as

$$\text{witness}_n((T_0, d_0), \ldots, (T_n, d_n)) \equiv \text{unrolling}_n((T_0, d_0), \ldots, (T_n, d_n)) \land$$

$$\exists k, m, T^*, d^* \left( k < m \land \bigvee_{i \leq n} (k = i \land T^* = T_i \land d^* = d_i) \land \bigvee_{i \leq n} (m = i \land T^* = T_i \land d^* = d_i) \land T^* \in \{T \mid \sigma(\chi(T)) = \nu \} \land \bigwedge_{i \leq n} k \leq i \leq m \Rightarrow T_i \geq T^* \right)$$

With this definition in hand, it is trivial to express the proposition that an $n$-step witness exists for a given state $(T, d)$ as an existential quantification over all potential $n$-step witnesses:

**Definition 9.** For a PBES $\mathcal{E}$, state $(T, d)$ of $\mathcal{E}$, and natural number $n$, we define the SMT proposition $\text{witness-exists}_n(T, d)$ as

$$\text{witness-exists}_n(T, d) \equiv \exists_{T_0, d_0, \ldots, T_n, d_n} [(T_0, d_0) = (T, d) \land \text{witness}_n((T_0, d_0), \ldots, (T_n, d_n))]$$

**Theorem 8.** Let $\mathcal{E}$ be a disjunctive PBES and let $(T, d)$ be a state of $\mathcal{E}$. Then $\text{witness-exists}(T, d) = \text{true}$ if and only if an $n$-step witness for $(T, d)$ exists.
Proof. If \( \text{witness-exists}(T, d) = \text{true} \), then there are \((T_0, d_0), \ldots, (T_n, d_n)\) as well as \(k, m\), and \((T^*, d^*)\) such that \((T_0, d_0) = (T, d)\), \((T_k, d_k) = (T^*, d^*)\), \(\sigma(\chi(T_k)) = \nu\), and \(T_k \leq T_i\) and therefore \(\chi(T_k) \leq \chi(T_i)\) for \(k \leq i \leq m\). Thus, \((T_0, d_0), \ldots, (T_n, d_n)\) is a witness for \((T_0, d_0)\), which means it is a witness for \((T, d)\).

Conversely, let an \(n\)-step witness for \((T, d)\) exist, which is an unrolling of the form \([(T, d)]S_1[(T^*, d^*)]S_2[(T^*, d^*)]S_3\) with \(\sigma(\chi(T^*)) = \nu\) and \(\chi(T^*) \leq \chi(T_i)\) for \((T_i, d_i) \in S_2\). But then that value of \((T^*, d^*)\) satisfies the \text{witness} predicate, which means \(\text{witness-exists}(T, d) = \text{true}\). \(\square\)

Because the value of the proposition \(\text{witness-exists}_{n}(T, d)\) can be computed by an SMT solver for given values of \(T, d, n\), the above forms one component out of two of an algorithm for solving a given model checking problem in the form of a disjunctive PBES. The following section will cover the other half of the problem: that of showing that no witnesses of a given state exist for any length larger than some number \(n\).

4.3 The absence of witnesses

Section 4.2 demonstrated a method for showing that \([E](X)(d) = \text{true}\) by showing that an \(n\)-step witness for \(X^d\) exists for some \(n\). Showing that \([E](X)(d) = \text{false}\), on the other hand, requires that no \(n\)-step witness for \(X^d\) exists for any \(n\). To show this, determining the absence of such a witness for any finite set of witness lengths is not sufficient, so a different approach is necessary.

Theorem 4 shows that bounds on the length of acyclic unrollings imply that \([E](X)(d) = \text{true}\) if and only if a witness for \(X^d\) exists. Further analysis can demonstrate bounds on the length of witnesses based on bounds of the length of acyclic unrollings, which provides the missing part for an algorithm for computing the value of \([E](X)(d)\).

**Theorem 9.** Let \(E\) be a disjunctive PBES, let \(X^d\) be a state of \(E\), and let \(n\) be a number such that no \(n\)-step acyclic unrolling starting in \(X^d\) exists. Furthermore, assume that no witness for \(X^d\) exists of any length \(m \leq 2n\). Then \([E](X)(d) = \text{false}\).

**Proof.** Assume that \([E](X)(d) = \text{true}\). Because the length of acyclic unrollings of \(X^d\) is bounded, by Theorem 4, a witness for \(X^d\) exists; let \([X_0^d, \ldots, X_m^d]\) be the shortest possible witness for \(X^d\). By assumption, no witness for \(X^d\) of less than or equal to \(2n\) exists; therefore, \(m > 2n\).

Since the sequence \([X_0^d, \ldots, X_m^d]\) is a witness for \(X^d\), it has the form \([X^d]S_1[Y^e]\)\(S_2[Y^e]\)\(S_3\). Because this witness is minimal, \(S_3\) must be empty; for the same reason, both \([X^d]S_1[Y^e]\) and \(S_2[Y^e]\) must be acyclic, as otherwise the witness could be shortened by removing the redundant cycle. Thus, \(Y^e\) occurs exactly twice in this witness, of which one occurrence is at the very end of the witness; let \(Y^e = X_k^d = X_m^d\) with \(k < m\). Because \([X^d]S_1[Y^e] = [X_0^d, \ldots, X_k^d]\) is acyclic, \(k < n\); because \(m > 2n\), \(m - k > 2n - k \geq n\).

Let \(\ell\) be such that \(X_\ell^d\) is the first state in the witness that occurs more than once; let \(j > i\) such that \(X_i^d = X_j^d\). Because \(S_2[Y^e] = [X_j^d, \ldots, X_m^d]\) is acyclic, \(i \leq k\); similarly, because \([X^d]S_1[Y^e] = [X_0^d, \ldots, X_k^d]\) is acyclic,
j > k. But then the sequence \([X_i, X_{i+1}, \ldots, X_m, X_{k+1}, \ldots, X_{j-1}]\) is an acyclic unrolling of \(m - k - 1 \geq n\) steps. Because by assumption \(X_i\) is the first state in the witness that occurs more than once, this unrolling does not contain any states from \([X_0, \ldots, X_{i-1}]\); from this, it follows that the sequence \([X_0, \ldots, X_i, X_{i+1}, \ldots, X_m, X_{k+1}, \ldots, X_{j-1}]\) is an acyclic unrolling starting in \(X = X_0\) of at least \(n\) steps. But by assumption, no \(n\)-step acyclic unrolling starting in \(X\) exists, which is a contradiction.

Theorem 9 shows that to prove that \([\mathcal{E}](X)(d) = \text{false}\), it is sufficient to show that no \(n\)-step acyclic unrolling exists starting in \(X\), and no \(2n\)-step witness exists for \(X\). Section 4.2 provides a method for showing the latter; the remaining problem is showing the former.

Fortunately, the existence of an \(n\)-step acyclic unrolling for state \((T, d)\) of disjunctive PBES \(E\) is easy to express as an SMT proposition:

**Definition 10.** For a PBES \(E\), state \((T, d)\) of \(E\), and natural number \(n\), we define the SMT proposition \(\text{acyclic-unrolling-exists}_n(T, d)\) as

\[
\text{acyclic-unrolling-exists}_n(T, d) \equiv \\
\exists T_0, d_0, \ldots, T_n, d_n \left( \begin{array}{c} \\
\text{unrolling}_n((T_0, d_0), \ldots, (T_n, d_n)) \land \\
(T_0, d_0) = (T, d) \land \\
\bigwedge_{i<j} (T_i, d_i) \neq (T_j, d_j) \\
\end{array} \right)
\]

Clearly, \(\text{acyclic-unrolling-exists}_n(T, d) = \text{true}\) if and only if an \(n\)-step unrolling of \((T, d)\) exists.

### 4.4 Deciding \([\mathcal{E}](X)(d)\)

Sections 4.2 and 4.3 describe techniques for both showing that an \(n\)-step witness for a given state exists, and showing that no such witness can exist of size larger than \(n\). Together, those techniques suggest an algorithm for deciding \([\mathcal{E}](X)(d)\) by attempting to prove either for increasingly large values of \(n\): once a witness of size \(n\) is found, it is known that \([\mathcal{E}](X)(d) = \text{true}\), and when a witness of size \(n\) is not found but proof that no witness of size larger than \(n\) can exist is found, it is known that \([\mathcal{E}](X)(d) = \text{false}\).

Theorem 9 gives a way for showing that \([\mathcal{E}](X)(d) = \text{false}\) by showing that no witness exists for any length less than or equal to some natural number \(2n\), and Theorem 8 provides a way of showing the absence of a witness of a given specific length. Fortunately, there is an easier way of showing that witnesses for some \(X^n\) exist of large lengths, and therefore by contraposition that no witnesses exist of small lengths:

**Lemma 10.** For a disjunctive PBES \(E\) and state \(X^d\) of \(E\), let a witness exist of length \(n\). Then witnesses exist for \(X^d\) of any length \(m \geq n\).
Proof. The witness for \( X^d \) of length \( n \) has the shape \([X^d]S_1[Y^e]S_2[Y^e]S_3\). But then \([X^d]S_1[Y^e]S_2[Y^e]\) is also a witness for \( X^d \), and so is any extension constructed by appending any number of copies of the sequence \( S_2[Y^e] \) to \([X^d]S_1[Y^e]S_2[Y^e]\). Truncating such an extended at the desired length gives a witness for any length larger than the length of \([X^d]S_1[Y^e]S_2[Y^e]\), which is less than or equal to \( n \).

Given that both witnesses and acyclic unrollings can be searched for for arbitrarily increasing lengths, an obvious algorithm using this scheme is the following, where \( \alpha > 1 \) is a configurable multiplication factor:

```
Algorithm smt-unrolling\(_\alpha\)(\(E\), \(X^d\))
1   \( n := 1 \)
2   while true:
3       For \(E\), compute whether witness-exists\(_{2n}\)(\(X^d\)) is true
4           if it is true:
5               return true
6       For \(E\), compute whether acyclic-unrolling-exists\(_n\)(\(X^d\)) is false
7           if it is false:
8               return false
9       \( n := n \times \alpha \)
```

For this algorithm, the following properties hold:

**Theorem 11.** For disjunctive PBES \( E \), state \( X^d \) of \( E \), and multiplication factor \( \alpha > 1 \), if \( \text{smt-unrolling}_\alpha(E, X^d) \) returns \( b \), then \([E](X)(d) = b\). In other words, \( \text{smt-unrolling}_\alpha \) is sound.

Proof. Let \( \text{smt-unrolling}_\alpha(E, X^d) \) return \( b \). Then \( \text{witness-exists}_{2n}(X^d) \) is true for some \( n \). Therefore, by Theorems 8 and 3, \([E](X)(d) = b\).

Alternatively, let \( \text{smt-unrolling}_\alpha(E, X^d) \) return \( f \). Then for some \( n \), both \( \text{witness-exists}_{2n}(X^d) \) and \( \text{acyclic-unrolling-exists}_n(X^d) \) are false; thus, no \( n \)-step acyclic unrolling of \( X^d \) exists, and by Theorem 8 and Lemma 10, no \( m \)-step witness for \( X^d \) exists for any \( m \leq 2n \). Therefore, by Theorem 9, \([E](X)(d) = f\).

**Theorem 12.** For disjunctive PBES \( E \), state \( X^d \) of \( E \), and multiplication factor \( \alpha > 1 \), if the length of acyclic unrollings of \( X^d \) is bounded, computation of \( \text{smt-unrolling}_\alpha(E, X^d) \) will terminate, under the assumption that the individual SMT computations terminate.

Proof. If the length of acyclic unrollings of \( X^d \) is bounded, then there is an \( m \) such that \( \text{acyclic-unrolling-exists}_m(X^d) \) is false; \( \text{smt-unrolling} \) will therefore eventually return \( f \) after computing that \( \text{acyclic-unrolling-exists}_n(X^d) \) is unsatisfiable for some \( n \geq m \), unless the algorithm terminates before that by returning \( b \). In either case, it terminates.

Together, these two properties show that \( \text{smt-unrolling} \) is a correct algorithm for computing the value of \([E](X)(d)\); if furthermore \( X^d \) has bounded acyclic unrollings – which in particular is the case for any PBESs for which the reachable state space is finite – the algorithm is complete, up to incompleteness of the SMT algorithm used.
Showing that \( X^d \) has bounded acyclic unrollings is potentially as hard as the model-checking problem itself, and undecidable in general. Fortunately, this is not required, as the smt-unrolling algorithm can be applied when one does not know whether \( X^d \) has bounded acyclic unrollings. When applying the smt-unrolling algorithm to a state for which acyclic unrollings exist for any length, the algorithm may run indefinitely, without ever terminating; but if it does terminate, the solution produced is correct. This characteristic is shared with many other model-checking algorithms; for instance, any algorithm that relies on generating the explicit state space represented by an algebraic system description will fail to terminate if that state space turns out to be infinitely large.

### 4.5 Structured domains

For a PBES \( E \) over domain \( D \), \( D \) typically consists of the disjoint union (denoted \( \cup \)) of domains \( D_X \) for each equation \( (\sigma X(d_X:D_X) = \varphi) \), such that the value of \( X(d) \) is only well-defined for \( d \in D_X \). Moreover, each \( D_X \) typically consists of the Cartesian product of distinct variables \( d_{X,1}, d_{X,2}, \ldots, d_{X,K} \).

**Example 6.** The PBES

\[
\begin{align*}
(\nu X(n:N, z:Z) &= x > 3 \lor Y(z + 1)) \\
(\mu Y(z:Z) &= X(|z|, z))
\end{align*}
\]

has the formal domain \( D = (N \times Z) \cup Z \).

In this section, we consider PBESs with domains such that each PBES variable \( X_j \) is well-defined over the domain \( D_{j,1} \times \ldots \times D_{j,K} \), i.e., PBESs of the form

\[
\begin{align*}
E &= (\sigma X_1(d_{1,1}:D_{1,1}, \ldots, d_{1,K_1}:D_{1,K_1}) = \varphi_1) \\
&\quad \cdots \\
&\quad (\sigma X_N(d_{N,1}:D_{N,1}, \ldots, d_{N,K_N}:D_{N,K_N}) = \varphi_N)
\end{align*}
\]

where \( N \) is the number of equations in \( E \).

When computing whether \( \text{witness-exists}_n(T,d) = \text{true} \) for some \( E \) over domain \( D \) and state \((T,d)\) using an SMT solver, it may improve the efficiency of this computation to express the structure of the domain \( D \) explicitly, by replacing the quantified variables \( d_i:D \) of \( \text{witness-exists}_n(T,d) \) by sets of variables \( \{d_{i,j,k}:D_{j,k} \mid j < J, k < K_j\} \), encoding the original \( d_i \) as \( (d_{i,1}, \ldots, d_{i,T_i,K_{T_i}}) \).

Replacing the variables \( d_i \) like this yields the following expanded version of occurs:
occurs'( \( T_a, (d_{a,1,1}, \ldots, d_{a,1,K_1}, \ldots, d_{a,N,1}, \ldots, d_{a,N,K_1}) \), \\
\( T_b, (d_{b,1,1}, \ldots, d_{b,1,K_1}, \ldots, d_{b,N,1}, \ldots, d_{b,N,K_1}) \)) \equiv

\bigvee_{i < |E|} \bigvee_{(E,P,T,f_1,\ldots,f_K) \in \psi(\chi(i))} \left( (T_a = i) \land (T_b = T) \land \\
\bigvee_{e \in E} \bigg[ P(d_{a,i,1},\ldots,d_{a,i,K_1},e) \land \\
\bigwedge_{k \leq K_T} (d_{b,T,k} = f_k(d_{a,i,1},\ldots,d_{a,i,K_1},e)) \bigg] \right)

Expanding the other occurrences of first-order variables in the witness-exists
and acyclic-unrolling-exists SMT predicates is straightforward.

4.6 Results

To test the practical effectiveness of the techniques presented in this chapter,
we have implemented the algorithm described in Section 4.4 using the CVC4
[BCD+11] and Yices [DDM06] SMT solvers. Both implementations were then
used to solve, or attempt to solve, a broad collection of example problems taken
from the mCRL2 example distribution [mcr14].

For each investigated problem, we measured the running time of both implementa-
tions of the smt-unrolling\textsubscript{2} algorithm; for comparison, we also measured the
running time of the traditional algorithm that simply enumerates all reachable
states in the PBES, as implemented by the pbes2bool program in the mCRL2
distribution. We also counted the number of PBES states in each problem, in-
dicated as “\#states”, as well as the size of the shortest witness or the shortest
acyclic unrolling characterizing the solution of each problem (“\#unrolling”).

For computations that would not terminate, such as the pbes2bool algorithm
on most infinite state space PBESs, we denote a running time of $\infty$. A running
time of “timeout” denotes a case where computation was aborted after an ex-
ceedingly large amount of time, but the computation is expected to terminate
eventually.

The sections below contain a rough description of the problems tested. The
full problem details are described in Appendix A, along with the exact test
procedure.

4.6.1 Food distribution

The food distribution problem [Zan13] describes a collection of isolated, non-self-
sufficient villages that periodically need to be supplied with food, produced by
a supply center in the area which is assumed to have an infinite supply of food.
The villages are supplied using a single truck with bounded cargo capacity, which
drives around between the supply center and the different villages, loading or
unloading food at each location. It takes time for the truck to travel between two
places, and each village consumes a certain amount of food from its stockpile
per time unit; furthermore, the amount of food that each village can store
is bounded. The model-checking problem to be solved for this system is the following: is there a driving and unloading schedule for the truck to follow such that none of the villages ever run out of food?

This problem is specified in the mCRL2 example distribution, using a configuration with three villages that can store up to 120, 120, and 200 units of food, a truck that can carry up to 300 units of food at any time, and food amounts expressed as integers. This gives it a naive upper bound on the size of the reachable state space of about $120 \times 120 \times 200 \times 300 \approx 10^9$ states; the actually reachable part of this state space turns out to contain approximately $10^8$ states.

Because \textit{pbes2bool} works by enumerating all reachable states in the PBES, this tool requires an exceedingly long time to solve the food distribution problem; the computation was aborted after running for several days.

For the purposes of this test, we produced several variants of the food distribution problem. For the problem configuration specified above, the solution is \textit{false}; that is, it is not possible to keep the villages supplied with food indefinitely. Changing the truck capacity to 320 food units changes this, producing a version whose solution is \textit{true}.

To test the relation between the size of the state space and the running time of the \textit{smt-unrolling$_2$} algorithm, we produced a version where the sizes of the village food stores and the truck capacity have all been multiplied by 1000; this has the result of increasing the size of the state space by a factor of $1000^4$ to an intimidating $10^{20}$ reachable states in the PBES. Finally, a version was created in which all food units are expressed as real numbers instead of integers, resulting in a state space that is infinitely and even uncountably large.

The test results for the above problems are summarized in Table 4.1. For the first two problems, the \textit{pbes2bool} tests were never completed, but are estimated to take somewhere between one and two weeks to run; the reason we did not attempt \textit{pbes2bool} on the other two cases should need no explanation.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Witness</th>
<th>#states</th>
<th>unrolling</th>
<th>pbes2bool</th>
<th>CVC4</th>
<th>Yices</th>
</tr>
</thead>
<tbody>
<tr>
<td>food-300</td>
<td>no</td>
<td>$\approx 10^9$</td>
<td>21</td>
<td>timeout</td>
<td>6.65s</td>
<td>8.46s</td>
</tr>
<tr>
<td>food-320</td>
<td>yes</td>
<td>$\approx 10^8$</td>
<td>12</td>
<td>timeout</td>
<td>1.52s</td>
<td>3.20s</td>
</tr>
<tr>
<td>food-32000</td>
<td>yes</td>
<td>$\approx 10^{20}$</td>
<td>12</td>
<td>timeout</td>
<td>1.51s</td>
<td>5.00s</td>
</tr>
<tr>
<td>food-real</td>
<td>yes</td>
<td>$\infty$</td>
<td>12</td>
<td>$\infty$</td>
<td>1.50s</td>
<td>3.23s</td>
</tr>
</tbody>
</table>

Table 4.1: Food distribution test results.

4.6.2 Retransmission protocols

The \textit{sliding window protocol} is used by network protocols such as TCP to coordinate packet retransmissions, ensuring that transmitted data is received in the correct order and without omissions or duplications even in the context of unreliable network connections. The sliding window protocol, and many variations of it, have been extensively studied in the formal methods literature, and the mCRL2 example distribution contains formal descriptions of several such systems. In this paper, we concern ourselves with three different simplified variations of the sliding window protocol:
• abp: The alternating bit protocol [BSW69], specified in the academic/abp example in the mCRL2 example collection;

• cabp: The related concurrent alternating bit protocol [KM92], specified in mCRL2’s academic/cabp example; and

• onebit: The sliding window protocol with a one-item window size [BG94], specified in mCRL2’s academic/onebit example.

Each of these protocols consists of messages containing information regarding the packet retransmission system, along with payload data containing the message body; all these protocols handle this payload as an opaque unit of data, which at some point is delivered to the user without further interpretation. For the purposes of model checking problems regarding those retransmission protocols, however, the payload data is not opaque: after all, different values for payload data can result in different corresponding PBES states. As a consequence, the amount of possible pieces of payload data – in other words, the size of the payload domain – can affect the number of states in the PBES, which is of crucial importance to the efficiency of algorithms such as pbes2bool.

Similarly to Section 4.6.1, we created two versions of each of the three retransmission protocols: one for which the payload domain consists of the Booleans, and one for which the payload domain consists of the integers. For the latter versions, this means that the number of reachable states in the resulting PBESs is infinite, which implies that they cannot be solved by pbes2bool or any other algorithms that work by enumerating the set of all reachable states.

For each of the resulting six systems, five different properties were investigated:

• payload-fairness: Each payload that can be sent infinitely often is sent infinitely often (false for all protocols);

• infinite-loss: A message can be lost infinitely often (true for cabp, false otherwise);

• infinite-sending: A given payload can be sent infinitely often (true for all protocols);

• send-receive: Any given payload is received eventually (false for all protocols);

• fair-send-receive: Any given payload is received eventually, as long as some messages eventually arrive correctly (true for abp, false otherwise).

Applied to six different systems, the above properties yield thirty model checking problems. The results of testing the pbes2bool, CVC4, and Yices algorithms on those problems are summarized in Table 4.2.

4.6.3 Other example problems

Besides the structured classes of problems regarding food distribution and retransmission protocols above, we also tested the smt-unrolling2 algorithm against several miscellaneous model checking problems that are part of the mCRL2 example collection:
<table>
<thead>
<tr>
<th>Problem</th>
<th>Witness</th>
<th>#states</th>
<th></th>
<th>unrolling</th>
<th>pbes2bool</th>
<th>CVC4</th>
<th>Yices</th>
</tr>
</thead>
<tbody>
<tr>
<td>abp-bool/payload-fairness</td>
<td>yes</td>
<td>594</td>
<td>22</td>
<td>0.04s</td>
<td>7.23s</td>
<td>3.35m</td>
<td></td>
</tr>
<tr>
<td>abp-bool/infinite-loss</td>
<td>yes</td>
<td>119</td>
<td>13</td>
<td>0.03s</td>
<td>1.14s</td>
<td>1.19s</td>
<td></td>
</tr>
<tr>
<td>abp-bool/infinite-sending</td>
<td>yes</td>
<td>78</td>
<td>20</td>
<td>0.03s</td>
<td>1.59s</td>
<td>6.80s</td>
<td></td>
</tr>
<tr>
<td>abp-bool/send-receive</td>
<td>yes</td>
<td>231</td>
<td>9</td>
<td>0.04s</td>
<td>1.22s</td>
<td>1.95s</td>
<td></td>
</tr>
<tr>
<td>abp-bool/fair-send-receive</td>
<td>no</td>
<td>131</td>
<td>unknown</td>
<td>0.07s</td>
<td>timeout</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>abp-int/payload-fairness</td>
<td>yes</td>
<td>∞</td>
<td>22</td>
<td>∞</td>
<td>5.11s</td>
<td>13:45m</td>
<td></td>
</tr>
<tr>
<td>abp-int/infinite-loss</td>
<td>yes</td>
<td>∞</td>
<td>13</td>
<td>∞</td>
<td>1.11s</td>
<td>0.91s</td>
<td></td>
</tr>
<tr>
<td>abp-int/infinite-sending</td>
<td>yes</td>
<td>∞</td>
<td>20</td>
<td>∞</td>
<td>1.54s</td>
<td>6.22s</td>
<td></td>
</tr>
<tr>
<td>abp-int/send-receive</td>
<td>yes</td>
<td>∞</td>
<td>9</td>
<td>∞</td>
<td>1.18s</td>
<td>1.60s</td>
<td></td>
</tr>
<tr>
<td>abp-int/fair-send-receive</td>
<td>no</td>
<td>∞</td>
<td>unknown</td>
<td>∞</td>
<td>timeout</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>cabp-bool/payload-fairness</td>
<td>yes</td>
<td>3714</td>
<td>7</td>
<td>1:49m</td>
<td>2.35s</td>
<td>4.35s</td>
<td></td>
</tr>
<tr>
<td>cabp-bool/infinite-loss</td>
<td>no</td>
<td>753</td>
<td>unknown</td>
<td>0.08s</td>
<td>timeout</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>cabp-bool/infinite-sending</td>
<td>yes</td>
<td>514</td>
<td>24</td>
<td>5.95s</td>
<td>47.0s</td>
<td>4:24m</td>
<td></td>
</tr>
<tr>
<td>cabp-bool/send-receive</td>
<td>yes</td>
<td>1553</td>
<td>5</td>
<td>0.15s</td>
<td>1.19s</td>
<td>1.98s</td>
<td></td>
</tr>
<tr>
<td>cabp-bool/fair-send-receive</td>
<td>yes</td>
<td>753</td>
<td>5</td>
<td>0.08s</td>
<td>1.32s</td>
<td>1.72s</td>
<td></td>
</tr>
<tr>
<td>cabp-int/payload-fairness</td>
<td>yes</td>
<td>∞</td>
<td>7</td>
<td>∞</td>
<td>1.38s</td>
<td>2.00s</td>
<td></td>
</tr>
<tr>
<td>cabp-int/infinite-loss</td>
<td>no</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>cabp-int/infinite-sending</td>
<td>yes</td>
<td>∞</td>
<td>24</td>
<td>∞</td>
<td>47.6s</td>
<td>2:25m</td>
<td></td>
</tr>
<tr>
<td>cabp-int/send-receive</td>
<td>yes</td>
<td>∞</td>
<td>5</td>
<td>∞</td>
<td>0.83s</td>
<td>1.43s</td>
<td></td>
</tr>
<tr>
<td>cabp-int/fair-send-receive</td>
<td>yes</td>
<td>∞</td>
<td>5</td>
<td>∞</td>
<td>0.96s</td>
<td>1.28s</td>
<td></td>
</tr>
<tr>
<td>onebit-bool/payload-fairness</td>
<td>yes</td>
<td>655352</td>
<td>7</td>
<td>timeout</td>
<td>2.30s</td>
<td>4.18s</td>
<td></td>
</tr>
<tr>
<td>onebit-bool/infinite-loss</td>
<td>yes</td>
<td>199937</td>
<td>6</td>
<td>timeout</td>
<td>1.34s</td>
<td>1.28s</td>
<td></td>
</tr>
<tr>
<td>onebit-bool/infinite-sending</td>
<td>yes</td>
<td>88834</td>
<td>24</td>
<td>timeout</td>
<td>1.39m</td>
<td>4:34m</td>
<td></td>
</tr>
<tr>
<td>onebit-bool/send-receive</td>
<td>yes</td>
<td>297089</td>
<td>4</td>
<td>31.8s</td>
<td>0.40s</td>
<td>0.32s</td>
<td></td>
</tr>
<tr>
<td>onebit-bool/fair-send-receive</td>
<td>yes</td>
<td>199937</td>
<td>4</td>
<td>23.3s</td>
<td>0.40s</td>
<td>0.25s</td>
<td></td>
</tr>
<tr>
<td>onebit-int/payload-fairness</td>
<td>yes</td>
<td>∞</td>
<td>7</td>
<td>∞</td>
<td>1.37s</td>
<td>0.97s</td>
<td></td>
</tr>
<tr>
<td>onebit-int/infinite-loss</td>
<td>yes</td>
<td>∞</td>
<td>6</td>
<td>∞</td>
<td>1.00s</td>
<td>0.63s</td>
<td></td>
</tr>
<tr>
<td>onebit-int/infinite-sending</td>
<td>yes</td>
<td>∞</td>
<td>24</td>
<td>∞</td>
<td>1.38m</td>
<td>2:41m</td>
<td></td>
</tr>
<tr>
<td>onebit-int/send-receive</td>
<td>yes</td>
<td>∞</td>
<td>4</td>
<td>∞</td>
<td>0.38s</td>
<td>0.21s</td>
<td></td>
</tr>
<tr>
<td>onebit-int/fair-send-receive</td>
<td>yes</td>
<td>∞</td>
<td>4</td>
<td>∞</td>
<td>0.37s</td>
<td>0.20s</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Retransmission protocols test results.
### Table 4.3: Miscellaneous test results.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Witness</th>
<th>#states</th>
<th>unrolling</th>
<th>$pbes2bool$</th>
<th>CVC4</th>
<th>Yices</th>
</tr>
</thead>
<tbody>
<tr>
<td>bakery</td>
<td>yes</td>
<td>$\infty$</td>
<td>6</td>
<td>$\infty$</td>
<td>0.65s</td>
<td>0.50s</td>
</tr>
<tr>
<td>dining/nodeadlock</td>
<td>yes</td>
<td>14159</td>
<td>11</td>
<td>2.51s</td>
<td>3.19s</td>
<td>1:16m</td>
</tr>
<tr>
<td>dining/nostarvation</td>
<td>yes</td>
<td>127423</td>
<td>7</td>
<td>22.6s</td>
<td>3.05s</td>
<td>9.32s</td>
</tr>
<tr>
<td>parallel</td>
<td>no</td>
<td>1001</td>
<td>1001</td>
<td>7.25s</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td>scheduler</td>
<td>no</td>
<td>14</td>
<td>12</td>
<td>0.03s</td>
<td>0.99s</td>
<td>0.69s</td>
</tr>
<tr>
<td>trains/fairness</td>
<td>no</td>
<td>258</td>
<td>unknown</td>
<td>0.03s</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td>trains/nodeadlock</td>
<td>no</td>
<td>33</td>
<td>27</td>
<td>0.03s</td>
<td>7.42s</td>
<td>17.5s</td>
</tr>
</tbody>
</table>

• **bakery**: In Lamport’s bakery algorithm [Lam74], must each requesting process eventually enter the critical section? (No.)

• **dining/nodeadlock**: In the Dining Philosophers problem [Hoa78] with 8 philosophers, can a deadlock occur? (Yes.)

• **dining/nostarvation**: In the Dining Philosophers problem [Hoa78] with 8 philosophers, can a situation occur where one philosopher will never eat again? (Yes.)

• **parallel**: In a system consisting of three parallel deadlock-free ten-state automata, can a deadlock occur? (No.)

• **scheduler**: In Milner’s scheduler [Mil82], can a deadlock occur? (No.)

• **trains/fairness**: Two trains require access to a critical section of railway, protected by Peterson’s algorithm [Pet81]. Is the access control fair – that is, does every train that wants access to the critical section get it eventually? (Yes.)

• **trains/nodeadlock**: Two trains require access to a critical section of railway, protected by Peterson’s algorithm [Pet81]. Can a deadlock occur? (No.)

The results of testing the $pbes2bool$, CVC4, and Yices algorithms on those problems are summarized in Table 4.3.

### 4.6.4 Conclusions

The most immediate conclusion suggested by the results in this section is the observation that the running time of the $smt-unrolling_2$ algorithm depends critically on the length of the shortest unrolling required to prove the solution to the model checking problem. Indeed, for all test problems with a shortest proving unrolling smaller than 15 states, the measured running time is quite modest; whereas for most problems with shortest proving unrollings larger than 20 states, the running time increases sharply. Moreover, we have not been able to solve any problems with shortest proving unrollings larger than 30 states in any reasonable amount of time.

This behavior matches the theory of SMT solvers. For a given PBES, the $smt-unrolling_2$ algorithm produces SMT problem instances with a number of
variables linear in the length of the shortest proving unrolling for that PBES; since SMT solvers have expected running times exponential in the number of free variables occurring in the problem, the behavior of the test results in this section are unsurprising.

A more subtle observation is that the running time of the smt-unrolling\textsubscript{2} algorithm does not seem to be strongly affected by the number of reachable states in a given PBES. This is clearest in Table 4.1, where the running times for the bottom three problems are very similar despite dramatic differences in the size of the state spaces. For Table 4.2, the results are a bit murkier, but they support the same picture: for 21 out of 26 problems, the running times for the Boolean-payload and integer-payload versions of the problem are within a factor 3 of each other. Surprisingly, for all but three of the problems, the version of the problem with the infinite payload domain takes significantly less time to solve than the version with the small finite payload domain.

The insensitivity of the smt-unrolling\textsubscript{2} algorithm to the size of a PBES’s state space is easy to explain by analyzing the structure of this state space, which is most clear in the retransmission protocol problems. For these problems, the state space of the PBES approximately consists of the Cartesian product of the state of the retransmission protocol, and the value of the payload data being transported by the protocol; that is, states are approximately pairs \((s, d)\), where \(s \in S\) is the state of the protocol state and \(d \in D\) is an element of the payload domain \(D\), which in the examples of this section is \(\mathbb{B}\) or \(\mathbb{N}\). In solving the model checking problem, the SMT solver will have to navigate the protocol state space \(S\) in order to find states or state cycles matching the desired property; but the payload data \(d\) is essentially a free variable, with very few – if any – constraints. The result is that the SMT solver can search a path in \(S\) with very limited regards for \(D\), leading to a running time that is very similar for \(D = \mathbb{B}\) versus \(D = \mathbb{N}\).

The reason why larger payload domains tend to result in shorter computation times is less clear. One hypothesis is that for small finite domains, the SMT solver can be tempted to fruitlessly analyze the problem by performing case distinction on the value of the payload; whereas for infinite domains, it recognizes this approach as hopeless and does not waste time on this line of reasoning that turns out to not yield any useful insights. But to produce any real insight into this matter, in-depth study of the analysis performed by SMT solvers on these problems is required, which is outside the scope of this article.

One case where the size of the state space is relevant is when trying to solve problems for which no witness exists. For those problems, the solution to the model checking problem is proved by showing the absence of an acyclic unrolling of a certain length \(n\). For PBESs with a finite state space, such an \(n\) always exists, bounded by the size of the state space. Unfortunately, it frequently happens that this bound is quite sharp, and that an acyclic unrolling is only absent for lengths quite close to the number of states in the PBES; examples of this situation are the parallel, scheduler, and trains/nodeadlock examples.

Because the running time of the smt-unrolling\textsubscript{2} algorithm is often exponential in the length of the shortest absent acyclic unrolling, this means that problems such as the above are only solveable in reasonable time when the accompanying PBESs have very modest amounts of reachable states; in the parallel example, a 1000-state problem is completely infeasible for solving using
Because the \textit{pbes2bool} algorithm’s running time depends on the size of the reachable state space, and the running time of the \textit{smt-unrolling} algorithm depends on the size of the shortest proving unrolling, the efficiency of the two algorithms is incomparable. Indeed, there are cases where the \textit{smt-unrolling} algorithm is exponentially (or even infinitely) faster than \textit{pbes2bool}, such as the food distribution problem; and vice versa, such as the parallel problem.

For problems with large data domains governed by simple predicates, such as the food distribution and retransmission protocol examples, SMT solvers can solve that part of the model checking problem in a purely symbolic way which is independent of the size of the domain, and through that technique provide exponential speedups over the more naive state space enumeration approach. When no such structure is apparent, however, all the transformation of the model checking problem to SMT accomplishes is that an inefficient encoding of the state space enumeration problem is performed by the SMT solver instead of tools optimized for this purpose.
Chapter 5

Partial instantiation

The results from Section 4.6 suggest that using an SMT solver to search for unrollings of a disjunctive PBES tends to work well to the degree that the PBES contains clauses with large quantification domains, and poorly to the degree that the PBES contains many different clauses.

To understand why this is the case, we can decompose an unrolling into a sequence of PBES clauses through which each state in the unrolling occurs in the previous state, and a sequence of values in the quantification domain of each such PBES clause. For such a clause sequence, finding values in the accompanying quantification domains that form a valid unrolling is a problem well-suited to the strengths of an SMT solver; on the other hand, enumerating valid clause sequences is little more than a state space enumeration problem, for which an SMT solver isn’t nearly as efficient as more specialized algorithms.

In this chapter, we develop an approach based on formally separating the two tasks described above, and using the right tool for each job. The basic technique here is to enumerate all the infinitely many clause sequences, and using an SMT solver to investigate each such sequence in search of proof determining the value of the model checking problem, one way or the other.

In Section 5.1, we formalize the concept of sequences of PBES clauses, and the assorted decomposition of unrollings. Section 5.2 develops an outline of an algorithm using this approach, after which Section 5.3 describes the details of how to encode the parts of this algorithm best solved using an SMT solver. In Section 5.4, we finalize the details of this algorithm, and prove properties regarding its correctness. Finally, Section 5.5 presents the experimental results of comparing this algorithm with the results from Section 4.6.

5.1 Symbolic unrollings

A disjunctive PBES, as defined in Definition 2, is a PBES for which the right hand sides all consist of disjunctions of disjunctive clauses of the form $(\exists e \in E : P(d, e) \land X(f(d, e)))$ for some $(E, P, X, f)$. An unrolling of a PBES is a sequence of states of that PBES such that each state occurs in the preceding state in the sequence; for a disjunctive PBES, that means that for each pair of consecutive states $(X^d_a, X^d_b)$, there is a disjunctive clause $(E, P, X_b, f)$ in the definition of $X_a$ such that $P(d_a, e)$ and $d_b = f(d_a, e)$ for some $e \in E$. 25
For an unrolling $[X_0^d, \ldots, X_n^d]$ of a disjunctive PBES, we can consider the sequence of clauses $(E_{i+1}, P_{i+1}, X_{i+1}, f_{i+1}) \in \psi(X_i)$ matching the requirements above; call this sequence a symbolic unrolling. Using this notion, an unrolling can be interpreted as a combination of a symbolic unrolling and the first-order values $[d_0, \ldots, d_n]$ making up the unrolling states; moreover, for a given symbolic unrolling and sequence of first-order values, the proposition that this combination forms an unrolling can be expressed in a straightforward way as a quantification over the quantification domains $E_i$ making up the symbolic unrolling.

Detaching the notion of a symbolic unrolling from any specific unrollings that belong to it, we can formally define a symbolic unrolling as an independent concept:

**Definition 11.** Let $\mathcal{E}$ be a disjunctive PBES, and let $X_0$ be a fixpoint symbol of $\mathcal{E}$. Then a symbolic unrolling of $X_0$ is a sequence of disjunctive clauses $[(E_1, P_1, X_1, f_1), \ldots, (E_n, P_n, X_n, f_n)]$ such that $(E_{i+1}, P_{i+1}, X_{i+1}, f_{i+1})$ is a disjunctive clause of $X_i$, denoted $(E_{i+1}, P_{i+1}, X_{i+1}, f_{i+1}) \in \psi(X_i)$, for all $i < n$.

A disjunctive PBES can be interpreted as a finite directed multigraph, with each fixpoint variable representing a vertex and each disjunctive clause $(E, P, X, f) \in \psi(Y)$ representing an edge $(Y, X)$; in this visualization, the symbolic unrollings of the PBES correspond with the finite paths through the graph. Clearly then, unless this graph is acyclic, the set of symbolic unrollings for a PBES is infinite and denumerable; equally obviously, the symbolic unrollings of a given fixpoint symbol $X_0$ can be efficiently enumerated using standard graph exploration algorithms.

A symbolic unrolling $L = [(E_1, P_1, X_1, f_1), \ldots, (E_n, P_n, X_n, f_n)]$ of $X_0$ generates a set of unrollings, called the unrollings in $L$, which are those unrollings $[X_0^{d_0}, \ldots, X_n^{d_n}]$ such that $X_i^{d_i+1}$ occurs in $X_i^{d_i}$ through the disjunctive clause $(E_{i+1}, P_{i+1}, X_{i+1}, f_{i+1})$.

**Definition 12.** Let $\mathcal{E}$ be a disjunctive PBES, let $X_0$ be a fixpoint variable of $\mathcal{E}$, let $L = [(E_1, P_1, X_1, f_1), \ldots, (E_n, P_n, X_n, f_n)]$ be a symbolic unrolling of $X_0$, and let $U = [X_0^{d_0}, \ldots, X_n^{d_n}]$ be a sequence of states of $\mathcal{E}$. Then $U$ is an unrolling in $L$, denoted $U \in L$ by abuse of notation, if and only if for all $i < n$, there is an $e \in E_{i+1}$ such that $P_{i+1}(d_i, e)$ and $d_{i+1} = f_{i+1}(d_i, e)$.

**Example 7.** In the disjunctive PBES

\[
(\nu X(n:N) = (\exists m \in \mathbb{N} : n > 10 \land Y(n + m)) \lor (\exists z \in \{*\} : n \leq 10 \land X(2n)))
\]

\[
(\mu Y(n:N) = (\exists z \in \mathbb{Z} : \text{true} \land X(n + z^2)) \lor (\exists z \in \{*\} : \text{true} \land X(2n)))
\]

the sequence of disjunctive clauses

\[
L = \left[ (\{*, (n, z \mapsto n \leq 10), X, (n, z \mapsto 2n)) \right]
\]

\[
\left[ (\mathbb{N}, (n, m \mapsto n > 10), Y, (n, m \mapsto n + m)) \right],
\]

\[
(\mathbb{Z}, \text{true}, X, (n, z \mapsto n + z^2))
\]

is a symbolic unrolling of $X$, and the sequence of states $[X^6, X^{12}, Y^{15}, X^{24}]$ is an unrolling of $X^6$ in $L$. 

26
The symbolic unrollings of a given disjunctive PBES $\mathcal{E}$ do not form a partition of the unrollings of $\mathcal{E}$, as a given unrolling may be generated by multiple different symbolic unrollings. For example, in Example 7, the unrolling $[Y^4, X^8]$ is generated both by the symbolic unrolling $[(Z, \text{true}, X, (n, z \mapsto n + z^2))]$ and by $[(\langle \ast \rangle, \text{true}, X, (n, z \mapsto 2n))]$. However, the unrollings in the symbolic unrollings of a disjunctive PBES do always cover the unrollings of that PBES.

**Lemma 13.** Let $\mathcal{E}$ be a disjunctive PBES, let $X^d$ be a state of $\mathcal{E}$, and let $U = [X_0^{d_0}, \ldots, X_n^{d_n}]$ be a sequence of states of $\mathcal{E}$. Then $U$ is an unrolling of $X^d$ if and only if $X_0^{d_0} = X^d$ and there is a symbolic unrolling $L$ of $X$ such that $U \in L$.

**Proof.** Let $U$ be an unrolling of $X^d$. Then $X_i^{d_i+1}$ occurs in $X_i^{d_i}$ for all $i < n$, and $X_0^{d_0} = X^d$. Then for each $i < n$, by the definition of a disjunctive PBES, there is a tuple $(E_{i+1}, P_{i+1}, X_{i+1}, f_{i+1}) \in \psi(X_i)$ and $e \in E_{i+1}$ such that $P_{i+1}(d_i, e) = \text{true}$ and $d_{i+1} = f_{i+1}(d_i, e)$. But then $L = [(E_1, P_1, X_1, f_1), \ldots, (E_n, P_n, X_n, f_n)]$ is a symbolic unrolling of $X_0 = X$, and $U \in L$.

Conversely, let $L = [(E_1, P_1, X_1, f_1), \ldots, (E_n, P_n, X_n, f_n)]$ be a symbolic unrolling of $X$ such that $U \in L$, and let $X_0^{d_0} = X^d$. Then for each $i < n$, $d_{i+1} = f_{i+1}(d_i, e)$ for some $e \in E_{i+1}$ such that $P_{i+1}(d_i, e) = \text{true}$. Because $L$ is a symbolic unrolling of $X = X_0$, $(E_{i+1}, P_{i+1}, X_{i+1}, f_{i+1}) \in \psi(X_i)$ for each $i < n$. But that means $X_{i+1}^{d_{i+1}}$ occurs in $X_i^{d_i}$ for $i < n$. Because furthermore $X_0^{d_0} = X^d$, that makes $U$ an unrolling of $X^d$.

Lemma 13 shows that the unrollings in the symbolic unrollings of a disjunctive PBES $\mathcal{E}$ are exactly the unrollings of $\mathcal{E}$. Because the symbolic unrollings of $\mathcal{E}$ are efficiently denumerable, the unrollings of $\mathcal{E}$ can be found by enumerating the symbolic unrollings of $\mathcal{E}$ and searching for unrollings in each symbolic unrolling; by extension, the same holds for witnesses and acyclic unrollings. The remainder of this chapter describes how to use an SMT solver to find such unrollings in a given symbolic unrolling, as well as developing an algorithm for deciding model checking problems using this approach.

### 5.2 Algorithm sketch

The results from Sections 3.2 and 3.3 together with Theorem 9 suggest that in order to determine the value of $[\mathcal{E}](X)(d)$ for some disjunctive PBES $\mathcal{E}$ and state $X^d$, all we need to do is enumerating all symbolic unrollings of $X$, investigate each for both witnesses for $X^d$ and acyclic unrollings of $X^d$, and either return true when a witness is found or return false when we find an $n$ such that no $n$-step acyclic unrolling of $X^d$ exists and no $2n$-step witness of $X^d$ exists. While this does work, more efficient schemes are possible.

For a disjunctive PBES $\mathcal{E}$ and symbolic unrolling $L$ of $\mathcal{E}$, an extension of $L$ is any unrolling that contains $L$ as a prefix (which includes the trivial extension $L$ of $L$); a one-step extension is an extension that is one clause longer than $L$. With this concept in hand, we can generalize the question of whether a witness for some state $X^d$ of $\mathcal{E}$ exists, to the question of whether a witness for $X^d$ exists in some extension of the symbolic unrolling $L$. Similarly, the question of whether there is a bound on the length of acyclic unrollings of $X^d$ can be generalized to the question of whether such a bound exists for acyclic unrollings.
of $X^d$ in extensions of $L$. In this generalization, the original question of whether a witness for $X^d$ exists anywhere is equivalent to the question of whether one exists in an extension of the empty symbolic unrolling.

The two generalized problems defined above have a simple recursive structure:

**Lemma 14.** Let $\mathcal{E}$ be a disjunctive PBES, let $X^d$ be a state of $\mathcal{E}$, and let $L$ be a symbolic unrolling of $X$. Then a witness for $X^d$ exists in an extension of $L$ if and only if one exists in $L$, or one exists in an extension of one of the one-step extensions of $L$.

*Proof.* The extensions of $L$ consist of the union of the 0-step extensions of $L$ and the extensions of $L$ of at least one step. The former is just $L$; the latter are exactly the extensions of the one-step extensions of $L$. \qed

**Lemma 15.** Let $\mathcal{E}$ be a disjunctive PBES, let $X^d$ be a state of $\mathcal{E}$, and let $L$ be a symbolic unrolling of $X$. Then the length of acyclic unrollings of $X^d$ in extensions of $L$ is bounded if and only if no acyclic unrolling of $X^d$ exists in $L$, or the length of acyclic unrollings of $X^d$ in extensions of $K$ is bounded for all one-step extensions $K$ of $L$.

*Proof.* If no acyclic unrollings of $X^d$ exist in $L$, any unrolling of $X^d$ in $L$ contains a cycle; since each unrolling in an extension of $L$ contains an unrolling in $L$ as a prefix, each unrolling in an extension of $L$ also contains a cycle. On the other hand, if the length of acyclic unrollings of $X^d$ in extensions of $K$ is bounded for all one-step extensions $K$ of $L$, then the maximum of those bounds provides a bound on the length of acyclic unrollings of $X^d$ in extensions of $L$.

Conversely, let the length of acyclic unrollings of $X^d$ in extensions of $L$ be bounded. Because the unrollings in extensions of the one-step extensions $K$ of $L$ are also unrollings in $L$, the length of acyclic unrollings of $X^d$ in extensions of $K$ are bounded by the same bound. \qed

The recursive structure demonstrated by Lemmata 14 and 15 allows for a search procedure that first investigates a given symbolic unrolling $L$ for witnesses or acyclic unrollings of $X^d$, followed by recursively searching all one-step extensions of $L$ if this investigation did not find a witness or a bound on the length of acyclic unrollings. But of course, similar to the problem faced in Section 4.3, this is only helpful if combined with a method for determining that a witness for $X^d$ does not exist in any extension of $L$.

Fortunately, similarly to how Theorem 9 shows that bounds on the length of acyclic unrollings translate to limits on the types of witnesses that can exist in a PBES, so does the absence of acyclic unrollings in a symbolic unrolling $L$ entail limits on the existence of witnesses in extensions of $L$:

**Lemma 16.** Let $\mathcal{E}$ be a disjunctive PBES, let $X^d$ be a state of $\mathcal{E}$, and let $L$ be a symbolic unrolling of $X$. Furthermore, let there be a $p$ such that in every unrolling $[X^d_0, \ldots, X^d_n]$ of $X^d$ in $L$, neither $[X^d_0, \ldots, X^d_p]$ nor $[X^d_p, \ldots, X^d_n]$ are acyclic. Then no minimal witness for $X^d$ exists in any extension of $L$.

*Proof.* Assume a minimal witness $[X^d_0, \ldots, X^d_m]$ exists in some extension of $L$. Because this witness is an unrolling that has an unrolling of $L$ as a prefix,
neither \([X^d_0, \ldots, X^d_p]\) nor \([X^d_p, \ldots, X^d_m]\) are acyclic. Let \(C_1\) denote the cycle in \([X^d_0, \ldots, X^d_p]\), and let \(C_2\) denote the cycle in \([X^d_p, \ldots, X^d_m]\).

Since the sequence \([X^d_0, \ldots, X^d_m]\) is a witness for \(X^d\), it has the form \([X^d], S_1[Y^e], S_2[Y^e], S_3\). Because this witness is minimal, \(S_3\) must be empty; for the same reason, both \([X^d], S_1[Y^e]\) and \(S_2[Y^e]\) must be acyclic, as otherwise the witness could be shortened by removing the redundant cycle. Because both these parts are acyclic, \(C_1\) must partially overlap both \([X^d], S_1[Y^e]\) and \(S_2[Y^e]\).

But then \(C_2\) is contained entirely in \(S_2[Y^e]\), which contradicts the assumption that \(S_2[Y^e]\) is acyclic.

Call a symbolic unrolling as required by Lemma 16 a symbolic unrolling with two disjoint cycle blocks for \(X^d\). Conveniently, the fact that a given symbolic unrolling \(L\) has two disjoint cycle blocks for \(X^d\) proves both that no minimal witnesses for \(X^d\) exist in any extension of \(L\), and that the length of acyclic unrollings of \(X^d\) in such extensions is bounded.

Similarly to Lemma 14, a minimal witness for \(X^d\) exists in an extension of \(L\) if and only if it exists either in \(L\) itself or in an extension of any of the one-step extensions of \(L\). By attempting to either find a witness for \(X^d\) in \(L\) or to show that \(L\) has two disjoint cycle blocks for \(X^d\), we can make one of the following judgments for any state \(X^d\) and symbolic unrolling \(L\) of \(X^d\):

- if \(L\) contains a witness for \(X^d\), the extensions of \(L\) contain a witness for \(X^d\);
- if \(L\) has two disjoint cycle blocks for \(X^d\), the extensions of \(L\) do not contain a minimal witness for \(X^d\) and the length of acyclic unrollings of \(X^d\) in extensions of \(L\) is bounded;
- if for any one-step extension \(K\) of \(L\) it holds that the extensions of \(K\) contain a witness for \(X^d\), so do the extensions of \(L\);
- if for all one-step extensions \(K\) of \(L\) it holds that the extensions of \(K\) do not contain a minimal witness for \(X^d\) and the length of acyclic unrollings of \(X^d\) in extensions of \(K\) is bounded, the same holds for the extensions of \(L\).

Together, these clauses specify a graph search problem: in the tree of extensions of \(L\), we want to either show that this tree contains a node which contains a witness for \(X^d\), or show that any simple path starting in the root must eventually hit a node with two disjoint cycle blocks for \(X^d\).

Implementing this graph search problem using a standard breadth first search yields the following algorithm:
The algorithm depicted above returns true for a state $X^d$ and symbolic unrolling $L$ of $X$ if and only if there is an extension $K$ of $L$ containing a witness for $X^d$; it returns false if and only if the length of acyclic unrollings of $X^d$ in extensions of $L$ is bounded and the extensions do not contain a minimal witness for $X^d$. Taking $L = []$, it follows that $\text{search-witnesses}_L(X^d)$ returns true if and only if a witness for $X^d$ exists, and false if the length of acyclic unrollings is bounded and no minimal witnesses for $X^d$ exist. Because a witness for a state exists if and only if a minimal witness exists for it, by Theorems 3 and 4, $\text{search-witnesses}_L$ is a sound algorithm for computing the value of $[E](X^d)$, and complete for those states $X^d$ whose acyclic unrollings are bounded.

For the special case $L = []$, certain optimizations are possible to the $\text{search-witnesses}_L$ algorithm. When computing whether $L'$ contains a witness for $X^d$, it is known that none of the proper prefixes of $L'$ contain a witness for $X^d$; taking that information into account may improve the performance of the contains-witness computation.

More substantially, if a symbolic unrolling $L'$ has two disjoint cycle blocks for $X^d$, then there is some proper prefix $K$ of $L'$ such that every unrolling of $X^d$ in $K$ is acyclic. Therefore, the computation of whether $L'$ has two disjoint cycle blocks for $X^d$ can be decomposed into two parts, reducing the amount of computation to be performed for each symbolic unrolling.

This decomposition can be performed by storing, for each symbolic unrolling $L'$ in the queue $Q$, the length of the smallest proper prefix of $L'$ for which every unrolling of $X^d$ contains at least one cycle, or $\bot$ if no such proper prefix exists. Then for any symbolic unrolling $L'$ for which no such prefix exists, we know that $L'$ does not have two disjoint cycle blocks for $X^d$; whereas for symbolic unrollings $L'$ for which such a prefix with length $m$ does exist, $L'$ has two disjoint cycle blocks for $X^d$ if and only if every unrolling of $X^d$ contains a cycle in the last $|L'| - m + 1$ states.

Storing this length $m$ together with the symbolic unrolling $L'$ as a pair $(L', m)$ in $Q$, implementing both these optimizations yields the following improved algorithm for $L = []$:

\begin{algorithm}
\begin{tabular}{ll}
1 & $Q := [L]$ \\
2 & \textbf{while} $|Q| > 0$: \\
3 & \quad $L' := Q_0$ \\
4 & \quad $Q := [Q_1, \ldots, Q_{|Q|}]$ \\
5 & \quad \textbf{if} $L'$ contains a witness for $X^d$: \\
6 & \quad \quad \textbf{return} true \\
7 & \quad \textbf{if} $L'$ does not have two disjoint cycle blocks for $X^d$: \\
8 & \quad \quad \textbf{for each} one-step extension $K$ of $L'$: \\
9 & \quad \quad \quad $Q := Q + [K]$ \\
10 & \quad \textbf{return} false
\end{tabular}
\end{algorithm}
Algorithm search-witnesses$D(X^d)$

1. \( Q := [[\emptyset, \bot]] \)
2. while \(|Q| > 0|\):
3. \((L', m) := Q_0\)
4. \( Q := [Q_1, \ldots, Q_{|Q|}] \)
5. if \( L' \) contains a witness for \( X^d \):
6. return true
7. if \( m = \bot \):
8. if every unrolling of \( X^d \) in \( L' \) contains a cycle:
9. \( m := |L'| \)
10. for each one-step extension \( K \) of \( L' \):
11. \( Q := Q + [(K, m)] \)
12. else:
13. if some unrolling of \( X^d \) in \( L' \) does not contain a cycle
14. in the last \(|L'| - m + 1 \) states:
15. for each one-step extension \( K \) of \( L' \):
16. \( Q := Q + [(K, m)] \)
17. return false

In the above algorithm outline, the computations of lines 5, 8 and 13 are intended to be solved using an SMT solver. The details of doing so are the subject of the next section.

5.3 Exploring symbolic unrollings

The algorithm described at the end of Section 5.2 contains two problems that are well-suited to the strength of an SMT solver. Those are the problem of determining whether a given symbolic unrolling contains a witness for a given state, and the problem of determining whether a symbolic unrolling contains an unrolling of a given state that is acyclic over a particular subsequence of the unrolling.

Both of these problems are of the form of determining whether a given symbolic unrolling \( L \) contains an unrolling of a state \( X^d \) matching a certain predicate. Similarly to the approach taken in Chapter 4, we will describe how to solve these problems using an SMT solver by first expressing the existence of an unrolling of \( X^d \) in \( L \) as an SMT problem, followed by extending this baseline into propositions expressing the existence of the particular desired unrollings.

5.3.1 Unrollings

For a disjunctive PBES \( \mathcal{E} \), state \( X^d_0 \) of \( \mathcal{E} \), and symbolic unrolling \( L = \[(E_1, P_1, X_1, f_1), \ldots, (E_n, P_n, X_n, f_n)\] \) of \( X_0 \), the unrollings of \( X^d_0 \) in \( L \) are those sequences of states \( [X^d_0, \ldots, X^d_n] \) such that for all \( i < n \), \( d_{i+1} = f_{i+1}(d_i, e) \) for some \( e \in E_{i+1} \) such that \( P_{i+1}(d_i, e) \). In particular, the fixpoint variables \( X_i \) making up the states \( X^d_i \) are fixed by the symbolic unrolling to which the unrolling belongs. This means that an unrolling in \( L \) can be represented as a sequence of first-order values \([d_0, \ldots, d_n]\).

Given such a sequence \([d_0, \ldots, d_n]\), the proposition that this sequence represents an unrolling in \( L \) is just a version of the unrolling predicate defined in Section 4.1, restricted to a particular set of disjunctive clauses:
Definition 13. For a disjunctive PBES $\mathcal{E}$, fixpoint variable $X$ of $\mathcal{E}$, disjunctive clause $C = (E, P, Y, f) \in \psi(X)$, and first-order values $d_a$ and $d_b$, we define the SMT proposition $\text{occurs}_{C}(d_a, d_b)$ expressing whether $Y^{d_0}$ occurs in $X^{d_a}$ through the disjunctive clause $C$ as

$$\text{occurs}_{C}(d_a, d_b) \equiv \exists e \in E (P(d_a, e) \land d_b = f(d_a, e))$$

Definition 14. For a disjunctive PBES $\mathcal{E}$, fixpoint variable $X_0$ of $\mathcal{E}$, symbolic unrolling $L = [(E_1, P_1, X_1, f_1), \ldots , (E_n, P_n, X_n, f_n)]$, and first-order values $d_0, \ldots , d_n$, we define the SMT proposition $\text{unrolling}_{X_0, L}(d_0, \ldots , d_n)$ expressing whether $[X_0^{d_0}, X_1^{d_1}, \ldots , X_n^{d_n}]$ is an unrolling in $L$ as

$$\text{unrolling}_{X_0, L}(d_0, \ldots , d_n) \equiv \bigwedge_{i<n} \text{occurs}_{(E_{i+1}, P_{i+1}, X_{i+1}, f_{i+1})}(d_i, d_{i+1})$$

Lemma 17. For a disjunctive PBES $\mathcal{E}$, fixpoint variable $X_0$ of $\mathcal{E}$, symbolic unrolling $L = [(E_1, P_1, X_1, f_1), \ldots , (E_n, P_n, X_n, f_n)]$, and first-order values $d_0, \ldots , d_n$, $\text{unrolling}_{X_0, L}(d_0, \ldots , d_n)$ is true if and only if $[X_0^{d_0}, X_1^{d_1}, \ldots , X_n^{d_n}]$ is an unrolling in $L$.

Proof. Let $\text{unrolling}_{X_0, L}(d_0, \ldots , d_n) = \text{true}$. Then for each $i < n$, there is an $e \in E_{i+1}$ such that $P_{i+1}(d_i, e) = \text{true}$ and $d_{i+1} = f_{i+1}(d_i, e)$. But that makes $[X_0^{d_0}, X_1^{d_1}, \ldots , X_n^{d_n}]$ an unrolling in $L$.

Conversely, let $[X_0^{d_0}, X_1^{d_1}, \ldots , X_n^{d_n}]$ be an unrolling in $L$. Then for all $i < n$, there is an $e \in E_{i+1}$ such that $P_{i+1}(d_i, e) = \text{true}$ and $d_{i+1} = f_{i+1}(d_i, e)$, which means $\text{occurs}_{(E_{i+1}, P_{i+1}, X_{i+1}, f_{i+1})}(d_i, d_{i+1}) = \text{true}$. But in that case $\text{unrolling}_{X_0, L}(d_0, \ldots , d_n)$ is also true.

Using the $\text{unrolling}_{X, L}$ predicate, it is easy to express the proposition that an unrolling of $X^d$ matching predicate $P$ exists in $L$ for some state $X^d$ and unrolling $L$ of $X$. Indeed, this is the case if and only if there exist $d_0, \ldots , d_n$ such that $d_0 = d$, $\text{unrolling}_{X, L}(d_0, \ldots , d_n) = \text{true}$, and $P(d_0, \ldots , d_n) = \text{true}$.

Remark. The above is rendered in the most straightforward way using an existential quantification over the variables $d_0, \ldots , d_n$, but this is not strictly necessary. In the definition of $\text{unrolling}_{X, L}(d_0, \ldots , d_n)$, there is a clause $d_{i+1} = f_{i+1}(d_i, e)$ for each $i < n$, which means there is exactly one value $d_{i+1}$ can take on for a given value of $d_i$ and $e$; thus, an existential quantification over $d_{i+1}$ is superfluous, as its value can simply be substituted wherever it occurs. The same holds for $d_0$, for which $d_0 = d$ holds.

Substituting definitions for all variables $d_i$ yields an SMT proposition in which the only quantified variables are the variables $e_i : E_i$, for the disjunctive clauses $(E_i, P_i, X_i, f_i) \in L$. This optimization may improve performance of the SMT solver, but it complicates the analysis in this section; therefore, it is left implicit and omitted from the remainder of this paper.

5.3.2 Witnesses

The problem invoked on line 5 of the search-witnesses algorithm is the problem of computing, for a given state $X^d$ of disjunctive PBES $\mathcal{E}$ and symbolic unrolling $L$ of $X$, whether there is a witness for $X^d$ in $L$. When computing this, we may
assume that all proper prefixes of $L$ do not contain witnesses of $X^d$, which means we can restrict ourselves to witnesses $[X^d]S_1[Y^e]S_2[Y^e]S_3$ for which $S_3$ is empty.

With a sequence of states of $E$ encoded as described in Section 5.3.1, the proposition that these states form a witness is simple to define:

**Definition 15.** For a disjunctive PBES $E$, fixpoint variable $X_0$ of $E$, symbolic unrolling $L = [(E_1, P_1, X_1, f_1), \ldots, (E_n, P_n, X_n, f_n)]$ of $X$, and first-order values $[d_0, \ldots, d_n]$, we define the SMT proposition $\text{witness}_L(d_0, \ldots, d_n)$ as

$$\text{witness}_L(d_0, \ldots, d_n) \equiv \text{unrolling}_L(d_0, \ldots, d_n) \land$$

$$(\sigma(X_n) = \nu) \land \bigwedge_{k<n} \left(X_k = X_n \land \bigwedge_{k<j<n} X_k \leq X_j \right) \land d_k = d_n$$

The proposition expressing that a witness for $X^d$ in $L$ exists is then trivial:

**Definition 16.** For a disjunctive PBES $E$, fixpoint variable $X_0$ of $E$, and symbolic unrolling $L = [(E_1, P_1, X_1, f_1), \ldots, (E_n, P_n, X_n, f_n)]$, we define the SMT proposition $\text{witness-exists}_L(d_0)$ as

$$\exists_{d_1, \ldots, d_n} \text{witness}_L(d_0, \ldots, d_n)$$

**Theorem 18.** Let $E$ be a disjunctive PBES, let $X^d$ be a state of $E$, and let $L = [(E_1, P_1, X_1, f_1), \ldots, (E_n, P_n, X_n, f_n)]$ be a symbolic unrolling of $X$. Furthermore, let no witness exist for $X^d$ in any proper prefix of $L$. Then a witness for $X^d$ exists in $L$ if and only if $\text{witness-exists}_L(d) = \text{true}$.

**Proof.** Let $[X_0^{d_0}, \ldots, X_n^{d_n}]$ be a witness for $X^d$ in $L$, with $k < m$ such that $X_k^{d_k} = X_n^{d_n}$, $\sigma(X_k) = \nu$, and $X_k \leq X_i$ for all $k < i < m$. Because no witness for $X^d$ exists in any proper prefix of $L$, $m = n$; because $[X_0^{d_0}, \ldots, X_n^{d_n}]$ is an unrolling in $L$, $\text{unrolling}_L(d_0, \ldots, d_n) = \text{true}$. But because $\sigma(X_k) = \sigma(X_m) = \sigma(X_0) = \nu$ and $X_k \leq X_i$ for $k < i < n$, that makes $\text{witness-exists}_L(d_0) = \text{true}$ for the given values of $[d_1, \ldots, d_n]$ and $k$.

Conversely, let $\text{witness-exists}_L(d_0) = \text{true}$. Then there exist $[d_0, \ldots, d_n]$ and $k$ such that $[X_0^{d_0}, \ldots, X_n^{d_n}]$ is an unrolling of $X^d$ in $L$, $X_k^{d_k} = X_n^{d_n}$, $\sigma(X_k) = \nu$, and $X_k \leq X_j$ for all $k < j < n$. But then $[X_0^{d_0}, \ldots, X_n^{d_n}]$ is a witness. Because $[X_0^{d_0}, \ldots, X_n^{d_n}]$ is an unrolling of $X^d$ in $L$, that makes $[X_0^{d_0}, \ldots, X_n^{d_n}]$ a witness for $X^d$ in $L$. \hfill $\Box$

### 5.3.3 Acyclic unrollings

The problem invoked on lines 8 and 13 of the search-witnesses algorithm is the problem of computing, for a given state $X^d$ of disjunctive PBES $E$ and symbolic unrolling $L$ of $X^d$, whether there is an unrolling $[X_0^{d_0}, \ldots, X_n^{d_n}]$ of $X^d$ in $L$ for which the subsequence $[X_k^{d_k}, \ldots, X_m^{d_m}]$ is acyclic for some $k \leq m$. Expressing this as an SMT proposition is trivial:
Definition 17. Let $E$ be a disjunctive PBES, let $X^d_0$ be a state of $E$, let $L = [(E_1, P_1, X_1, f_1), \ldots, (E_n, P_n, X_n, f_n)]$ be a symbolic unrolling of $X$, and let $k \leq m \leq n$. Then we define the SMT proposition $\text{acyclic-unrolling-exists}_{L,k,m}(d_0)$ as

$$\text{acyclic-unrolling-exists}_{L,k,m}(d_0) \equiv \exists d_1, \ldots, d_n \text{ unrolling}_{L}(d_0, \ldots, d_n) \land \bigwedge_{k \leq i < j \leq m} (X_i \neq X_j \lor d_i \neq d_j)$$

Clearly, $\text{acyclic-unrolling-exists}_{L,k,m}(d) = \text{true}$ if and only if an unrolling $[X^d_0, \ldots, X^d_n]$ of $X^d$ exists in $L$ for which $[X^d_k, \ldots, X^d_m]$ is acyclic.

5.4 Algorithm details

In Section 5.2 we defined the algorithm $\text{search-witnesses}$ for computing the value of $[E](X)(d)$ for some disjunctive PBES $E$ and state $X^d$ of $E$; in Section 5.3, we filled the gaps in this algorithm by implementing the remaining problems as invocations of an SMT solver. Together, they yield this finished algorithm, which we call $\text{partial-instantiation}$:

Algorithm $\text{partial-instantiation}(E, X^d)$
1. $Q := [[[\bot]]]$
2. while $|Q| > 0$:
   3. $(L, m) := Q_0$
   4. $Q := \{Q_1, \ldots, Q_{|Q|}\}$
   5. if witness-exists$_L(d) = \text{true}$:
      6. return true
   7. if $m = \bot$:
      8. if $\text{acyclic-unrollings-exist}_{L,0,|L|}(d) = \text{false}$:
         9. $m := |L|$
         10. for each one-step extension $K$ of $L$:
              11. $Q := Q + [(K, m)]$
      12. else:
         13. if $\text{acyclic-unrollings-exist}_{L,m,|L|}(d) = \text{true}$:
             14. for each one-step extension $K$ of $L$:
                 15. $Q := Q + [(K, m)]$
      16. return false

This algorithm has the same soundness and completeness properties as the $\text{smt-unrolling}_\alpha$ algorithm:

Theorem 19. For disjunctive PBES $E$ and state $X^d$ of $E$, if the invocation $\text{partial-instantiation}(E, X^d)$ returns $b$, then $[E](X)(d) = b$. In other words, $\text{partial-instantiation}$ is sound.

Proof. Let the invocation $\text{partial-instantiation}(E, X^d)$ return $\text{true}$. Then clearly, witness-exists$_L(d)$ is true for some symbolic unrolling $L$ of $X$. Because for all proper prefixes $K$ of $L$, witness-exists$_K(d)$ was computed to be $\text{false}$, it follows by Theorem 18 that a witness exists for $X^d$ in $L$. By Theorem 3, $[E](X)(d) = \text{true}$.

Alternatively, let $\text{partial-instantiation}(E, X^d)$ return $\text{false}$. Let $n$ be the length of the last symbolic unrolling considered in the while-loop. Then for
all $n$-clause symbolic unrollings of $X$, there is a prefix $L$ and number $m$ such that \textup{acyclic-unrollings-exist}_{L,m,|L|}(d)$ was computed to be false on line 13, which means that every unrolling of $X^d$ in $L$ contains a cycle in the last $|L| - m + 1$ states. Furthermore, the fact that \textup{acyclic-unrollings-exist}_{L,m,|L|}(d)$ was computed at all means that \textup{acyclic-unrollings-exist}_{K,0,m}(d) was computed to be false on line 8, where $K$ is the $m$-clause prefix of $L$; but that means that every unrolling of $X^d$ in $L$ contains a cycle in the first $m$ states. Together with the fact that each such unrolling also contains a cycle in the last $|L| - m + 1$ states, this means that $L$ has two disjoint cycle blocks for $X^d$; by Lemma 16, this means no minimal witness for $X^d$ exists in any extension of $L$. Since also all prefixes $P$ of $L$ have \textup{witness-exists}_P(d) = \text{false}$ (for otherwise \textup{partial-instantiation}(E, X^d) would have returned true), it follows that no witness exists for $X^d$, and the length of acyclic unrollings of $X^d$ is bounded. By Theorem 4, this means $[E](X)(d) = \text{false}$. 

**Theorem 20.** For disjunctive PBES $E$ and state $X^d$ of $E$, if the length of acyclic unrollings of $X^d$ is bounded, computation of \textup{partial-instantiation}(E, X^d) will terminate, under the assumption that the individual SMT computations terminate.

**Proof.** Let $n$ be the length of the longest acyclic unrolling of $X^d$. Then \textup{acyclic-unrollings-exist}_{L,0,|L|}(d) = \text{false} for all symbolic unrollings $L$ with $|L| > n$. Thus, for all $L$ with $|L| > 2n$ that get considered in the \texttt{while}-loop, \textup{acyclic-unrollings-exist}_{L,n+1,|L|}(d) gets computed. Any unrolling of length larger than $2n$ must contain a cycle in the last $n + 1$ states, for otherwise an acyclic unrolling of more than $n$ states can be constructed; hence, each symbolic unrolling $L$ with $|L| > 2n$ has two disjoint cycle blocks. Therefore, the proposition \textup{acyclic-unrollings-exist}_{L,n+1,|L|}(d) is false. Because then neither line 11 nor line 15 ever gets executed for an $L$ with $|L| > 2n$, \textup{partial-instantiation}(E, X^d) must terminate eventually.

Theorems 19 and 20 show that the \textup{partial-instantiation} algorithm is as powerful as the \textup{smt-unrolling} algorithm, while making more efficient use of the strengths of SMT solvers yet avoiding their weak points. The next section describes the degree to which this approach is successful.

### 5.5 Results

In order to compare the effectiveness of the \textup{partial-instantiation} algorithm with the \textup{smt-unrolling} algorithm developed in Chapter 4, we have implemented the \textup{partial-instantiation} algorithm using the CVC4 SMT solver. This implementation was then used to investigate the same sample problems used in Section 4.6, thus measuring the relative strengths of the two different algorithms.

As in Section 4.6, we measured the running time of both the \textup{partial-instantiation} (denoted “partial”) and \textup{smt-unrolling} algorithms as applied to each tested problem. For each problem, we also counted the number of distinct SMT problems solved during the execution of the \textup{partial-instantiation} algorithm, denoted “#problems”. The results of these measurements are summarized in Table 5.1.
<table>
<thead>
<tr>
<th>Problem</th>
<th>Witness</th>
<th>unrolling</th>
<th>#problems</th>
<th>smt-unrolling</th>
<th>partial</th>
</tr>
</thead>
<tbody>
<tr>
<td>food-300</td>
<td>no</td>
<td>21</td>
<td>5414</td>
<td>6.65s</td>
<td>1.53m</td>
</tr>
<tr>
<td>food-320</td>
<td>yes</td>
<td>12</td>
<td>1235</td>
<td>1.52s</td>
<td>16.9s</td>
</tr>
<tr>
<td>food-320000</td>
<td>yes</td>
<td>12</td>
<td>1235</td>
<td>1.51s</td>
<td>17.6s</td>
</tr>
<tr>
<td>food-real</td>
<td>yes</td>
<td>12</td>
<td>1235</td>
<td>1.50s</td>
<td>18.9s</td>
</tr>
<tr>
<td>abp-bool/payload-fairness</td>
<td>yes</td>
<td>22</td>
<td>5361</td>
<td>7.23s</td>
<td>1.40s</td>
</tr>
<tr>
<td>abp-bool/infinite-loss</td>
<td>yes</td>
<td>13</td>
<td>147</td>
<td>1.14s</td>
<td>0.06s</td>
</tr>
<tr>
<td>abp-bool/infinite-sending</td>
<td>yes</td>
<td>20</td>
<td>625</td>
<td>1.50s</td>
<td>0.26s</td>
</tr>
<tr>
<td>abp-bool/send-receive</td>
<td>yes</td>
<td>9</td>
<td>59</td>
<td>1.22s</td>
<td>0.05s</td>
</tr>
<tr>
<td>abp-bool/fair-send-receive</td>
<td>no</td>
<td>unknown</td>
<td>6100</td>
<td>timeout</td>
<td>3.27s</td>
</tr>
<tr>
<td>abp-int/payload-fairness</td>
<td>yes</td>
<td>22</td>
<td>5361</td>
<td>5.11s</td>
<td>1.33s</td>
</tr>
<tr>
<td>abp-int/infinite-loss</td>
<td>yes</td>
<td>13</td>
<td>147</td>
<td>1.11s</td>
<td>0.06s</td>
</tr>
<tr>
<td>abp-int/infinite-sending</td>
<td>yes</td>
<td>20</td>
<td>625</td>
<td>1.54s</td>
<td>0.22s</td>
</tr>
<tr>
<td>abp-int/send-receive</td>
<td>yes</td>
<td>9</td>
<td>59</td>
<td>1.18s</td>
<td>0.04s</td>
</tr>
<tr>
<td>abp-int/fair-send-receive</td>
<td>no</td>
<td>unknown</td>
<td>6100</td>
<td>timeout</td>
<td>3.35s</td>
</tr>
<tr>
<td>cabp-bool/payload-fairness</td>
<td>yes</td>
<td>7</td>
<td>247</td>
<td>2.35s</td>
<td>0.09s</td>
</tr>
<tr>
<td>cabp-bool/infinite-loss</td>
<td>no</td>
<td>unknown</td>
<td>unknown</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td>cabp-bool/infinite-sending</td>
<td>yes</td>
<td>24</td>
<td>unknown</td>
<td>47.0s</td>
<td>timeout</td>
</tr>
<tr>
<td>cabp-bool/send-receive</td>
<td>yes</td>
<td>5</td>
<td>71</td>
<td>1.19s</td>
<td>0.05s</td>
</tr>
<tr>
<td>cabp-bool/fair-send-receive</td>
<td>yes</td>
<td>5</td>
<td>181</td>
<td>1.32s</td>
<td>0.07s</td>
</tr>
<tr>
<td>cabp-int/payload-fairness</td>
<td>yes</td>
<td>7</td>
<td>247</td>
<td>1.38s</td>
<td>0.08s</td>
</tr>
<tr>
<td>cabp-int/infinite-loss</td>
<td>no</td>
<td>unknown</td>
<td>unknown</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td>cabp-int/infinite-sending</td>
<td>yes</td>
<td>24</td>
<td>unknown</td>
<td>47.6s</td>
<td>timeout</td>
</tr>
<tr>
<td>cabp-int/send-receive</td>
<td>yes</td>
<td>5</td>
<td>71</td>
<td>0.83s</td>
<td>0.05s</td>
</tr>
<tr>
<td>cabp-int/fair-send-receive</td>
<td>yes</td>
<td>5</td>
<td>181</td>
<td>0.96s</td>
<td>0.06s</td>
</tr>
<tr>
<td>onebit-bool/payload-fairness</td>
<td>yes</td>
<td>7</td>
<td>1925</td>
<td>2.30s</td>
<td>0.30s</td>
</tr>
<tr>
<td>onebit-bool/infinite-loss</td>
<td>yes</td>
<td>6</td>
<td>1253</td>
<td>1.34s</td>
<td>0.24s</td>
</tr>
<tr>
<td>onebit-bool/infinite-sending</td>
<td>yes</td>
<td>24</td>
<td>unknown</td>
<td>1.39m</td>
<td>timeout</td>
</tr>
<tr>
<td>onebit-bool/send-receive</td>
<td>yes</td>
<td>4</td>
<td>57</td>
<td>0.40s</td>
<td>0.04s</td>
</tr>
<tr>
<td>onebit-bool/fair-send-receive</td>
<td>yes</td>
<td>4</td>
<td>261</td>
<td>0.40s</td>
<td>0.10s</td>
</tr>
<tr>
<td>onebit-int/payload-fairness</td>
<td>yes</td>
<td>7</td>
<td>1925</td>
<td>1.37s</td>
<td>0.35s</td>
</tr>
<tr>
<td>onebit-int/infinite-loss</td>
<td>yes</td>
<td>6</td>
<td>1253</td>
<td>1.00s</td>
<td>0.24s</td>
</tr>
<tr>
<td>onebit-int/infinite-sending</td>
<td>yes</td>
<td>24</td>
<td>unknown</td>
<td>1.38m</td>
<td>timeout</td>
</tr>
<tr>
<td>onebit-int/send-receive</td>
<td>yes</td>
<td>4</td>
<td>57</td>
<td>0.38s</td>
<td>0.06s</td>
</tr>
<tr>
<td>onebit-int/fair-send-receive</td>
<td>yes</td>
<td>4</td>
<td>261</td>
<td>0.37s</td>
<td>0.09s</td>
</tr>
<tr>
<td>bakery</td>
<td>yes</td>
<td>6</td>
<td>203</td>
<td>0.65s</td>
<td>0.05s</td>
</tr>
<tr>
<td>dining/nodeadlock</td>
<td>yes</td>
<td>11</td>
<td>unknown</td>
<td>3.19s</td>
<td>timeout</td>
</tr>
<tr>
<td>dining/nostarvation</td>
<td>yes</td>
<td>7</td>
<td>95095</td>
<td>3.05s</td>
<td>9.94s</td>
</tr>
<tr>
<td>parallel</td>
<td>no</td>
<td>1001</td>
<td>unknown</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td>scheduler</td>
<td>no</td>
<td>12</td>
<td>900</td>
<td>0.99s</td>
<td>0.18s</td>
</tr>
<tr>
<td>trains/fairness</td>
<td>no</td>
<td>unknown</td>
<td>0.03s</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td>trains/nodeadlock</td>
<td>no</td>
<td>27</td>
<td>unknown</td>
<td>7.42s</td>
<td>timeout</td>
</tr>
</tbody>
</table>

Table 5.1: partial-instantiation algorithm test results.
5.5.1 Food distribution

The most immediate observation about the test results regarding the food distribution problem, as described in Section 4.6.1, is that the partial-instantiation algorithm is considerably slower for this class of problems than the naive smt-unrolling algorithm. Indeed, the partial-instantiation algorithm requires an order of magnitude more time to run on all four examples. While the reason for this isn’t clear, some speculation is possible.

In the food distribution system, each symbolic unrolling represents a sequence of villages the distribution truck visits in that order; the quantification variables represent the amount of food loaded or unloaded at each village. Because bad decisions about where to drive can easily lead to villages inevitably running out of food several steps later, a symbolic unrolling representing an unwise driving schedule can be doomed from very early on, even though it make take several further steps for this to become apparent.

When this is the case, in the smt-unrolling algorithm, the SMT solver may be able to conclude the hopelessness of this driving order early on, and as a far-reaching conclusion decide to drop that possibility from further consideration, thus realizing strong constraints about possible valid routes. On the other hand, in the partial-instantiation algorithm, no such foresight is possible, and significant computation time is wasted on such a dead end before its hopelessness becomes clear. This difference may well explain the difference in performance between the two algorithms.

5.5.2 Retransmission protocols

For the test results regarding the retransmission protocol problems, things look more encouraging. Here, for 24 out of 30 problems, the partial-instantiation algorithm performs considerably better than the smt-unrolling algorithm; for many problems, by more than one order of magnitude. For two problems, the partial-instantiation algorithm can find a solution in seconds whereas the smt-unrolling algorithm does not get anywhere in any reasonable amount of time; on the other hand, there are four problems that the smt-unrolling algorithm can solve but the partial-instantiation algorithm cannot.

Here, the reason behind the success of the partial-instantiation algorithm is straightforward. As described in Section 4.6.4, the state space of these PBESs very cleanly decomposes into a Cartesian product of a finite protocol state space and anywhere between zero and a handful of payload variables; moreover, the payload values are treated as purely opaque units of data and not manipulated in any way. In the partial-instantiation algorithm, the result of this is that the breadth first search loop explores the finite protocol state space, whereas the SMT problems describe the relations between payload variables. But because payload variables are opaque and never manipulated, those SMT problems are all completely trivial, and therefore very easily solved.

As a result, what effectively happens is that the partial-instantiation algorithm can explore the finite protocol state space of the retransmission protocol problems independently from the assorted payload values. In other words, the payload values are effectively abstracted out, and an explicit-state algorithm is used to investigate the remaining quotient space. This makes the complete algorithm the “best of both worlds” in a powerful sense. The fact that the
explicit-state algorithm used is a rather naive one explains why this technique nonetheless fails on some of the retransmission protocol problems.

5.5.3 Conclusions

Overall, the comparison between the partial-instantiation and smt-unrolling algorithms as summarized in Table 5.1 give a mixed impression. While the partial-instantiation algorithm has major benefits over the smt-unrolling algorithm in the majority of tested problems, there are also several cases where it causes a major slowdown instead, as well as some where the partial-instantiation algorithm breaks down altogether.

On the other hand, the availability of two distinct algorithms that work well on a different set of problems has its benefits. While we have no detailed formal analysis of the type of problems to which the two techniques are most suited, the analyses in the previous two sections give a clear indication that there is additional structure to be unraveled here. With more insight in the type of problem structures SMT solvers can effectively deduce properties about, it may well be possible to see exactly to what degree both approaches are applicable to a given problem. With luck, it may even be possible to combine the two techniques in an automated way, leaving exactly those parts of the problem to the SMT solver that it is likely to work well at.

The discussion in Section 5.5.2 regarding explicit-state algorithms applied to quotient spaces warrants some further independent attention. In the partial-instantiation algorithm, each symbolic unrolling represents a set of PBES states; thus, disjunctive clauses each form some sort of transition between sets of PBES states. The partial-instantiation algorithm can then be interpreted as exploring the implied graph of sets of states, in the hope of finding a cycle or some other interesting structure. When interpreted as an explicit-state model checking algorithm over this implied graph, it is clear that this explicit-state model checking algorithm is a fairly rudimentary one.

When interpreting a PBES as a graph of sets of states in this way, a natural question is whether this graph can be constructed explicitly, after which more sophisticated explicit-state model checking algorithms could be used to investigate the resulting graph. While this might indeed be possible, this construction is impeded by a fundamental problem. In order to create such a quotient state space, one needs to be able to solve problems such as “for a set of states \(X\) and a state transformer \(f\), does \(f(X)\) contain any states not contained in \(X\)?”

This question is equivalent to questions of the form \(\exists x \in X, \neg(f(x) \in X)\). Because this is the conjunction of an NP-complete and an coNP-complete problem, it is not in NP; from this we can conclude that problems such as this can not generally, or even usually, be solved using an SMT solver. Thus, while the construction of such a quotient graph might be possible, it would require a more powerful tool than an SMT solver, and is therefore outside the scope of this paper.
Chapter 6

Conclusion and future work

In this paper, we presented two algorithms for solving a class of parameterized Boolean equation systems that we call conjunctive and disjunctive PBESs, based on using the power of SMT solvers to find states in a large domain matching some property. Both algorithms work by investigating unrollings of the PBES, in search of particular unrollings showing the truth or falsity of the model checking problem in question.

Both algorithms have running times that depend strongly on the length of the unrolling needed to fix the value of the model checking problem. Moreover, both problems hardly depend on the size of the state space of the PBES, if at all. This has the exciting consequence that both algorithms have no problem at all dealing with problems with very large or even infinitely large state spaces. On the other hand, it also means that either algorithm may take a very long time solving problems that would be utterly trivial when using explicit-state techniques.

The two algorithms differ primarily in the amount of work they leave to the SMT solver. The first algorithm, smt-unrolling, simply encodes the entire model checking problem as an SMT problem. In contrast, the second algorithm – partial-instantiation – decomposes the problem into an explicit-state component and a symbolic component, using the SMT solver only to tackle the latter half while using explicit-state techniques on the former half. This works well if the model checking problem does in fact have such a structure that can be neatly decomposed; if such a structure is less clear, or absent altogether, the smt-unrolling algorithm works better.

Both algorithms solve only the conjunctive and disjunctive fragments of the set of all PBESs. While this is a serious restriction, conjunctive and disjunctive PBESs are quite common in practice: a survey of the mCRL2 example problem collection shows that roughly as many as half of all problems in this set are conjunctive or disjunctive. Under the assumption that the mCRL2 example collection is reasonably representative, this provides some indication of how applicable an algorithm for conjunctive or disjunctive PBESs really is, in the wider landscape of practical PBES problems in general.

Some open work remains on the topic of using SMT solvers for solving PBES model checking problems, and related techniques. One obvious topic is the extension of the techniques described in this paper to PBESs that are neither
conjunctive nor disjunctive. While this is problematic for pure-SMT approaches such as the *smt-unrolling* algorithm, the *partial-instantiation* algorithm is flexible enough that it could likely be extended to problems beyond the conjunctive and disjunctive. If possible, this would make the techniques described in this paper considerably more widely applicable.

Sections 5.5.1 and 5.5.2 contain some basic analysis of cases in which the *smt-unrolling* algorithm is effective, versus cases where the *partial-instantiation* algorithm works better. The structure discussed there suggests there is more going on than has been revealed; if the underlying structure were better understood, it might well be possible to design an algorithm combining the strengths of the *smt-unrolling* and *partial-instantiation* algorithms, reliably being more effective than either.

Finally, the concept of distilling an explicit-state model checking problem of which the states represent sets of states of a symbolic model checking problem is a potentially very powerful technique. If something along those lines is indeed possible, it could support combinations of explicit-state and symbolic model checking on a far larger scale than has been investigated in this paper. While such a technique would probably use tools beyond SMT solvers and therefore be outside the scope of this paper per se, the possibilities it would support are formidable.
Bibliography


Appendix A

Example problem details

In Section 4.6, we gave a rough description of the various model checking problems used to test the \textit{smt-unrolling} and \textit{partial-instantiation} algorithms. To ensure reproducibility, this appendix describes the exact problems investigated in these experiments, as well as the exact test procedure used.

All model checking problems used to test the algorithms described in this paper are part of the example problem set of the mCRL2 model checking toolset \cite{mcr14}; in this paper, we used the example problems that are part of the 201310.0 release of the mCRL2 distribution. Each investigated model checking problem was produced using the following general procedure:

1. The problem is based on a system description in the form of an mcrl2 file that is part of the example distribution. If necessary, is modified to suit our purposes.

2. The system description is converted to a linear process specification using the \texttt{mcrl2lps} tool.

3. To optimize the resulting linear process specification for analysis, summations over enum domains (which includes the Booleans) are flattened using the \texttt{lpssuminst} tool.

4. Using the \texttt{lps2pbes} tool, the optimized linear process specification is combined with a formula in the modal $\mu$-calculus expressing the property to be verified.

5. Attempts are made to solve the resulting PBES using the four algorithms measured in this paper: the \texttt{pbes2bool} algorithm that is part of the mCRL2 toolset, the CVC4 and Yices implementations of the \textit{smt-unrolling} algorithm, and the \textit{partial-instantiation} algorithm. For the \texttt{pbes2bool} algorithm, we tried solving each problem using both the \texttt{-rjitty} and \texttt{-rjittyC} options, and counted the faster result; no other nonstandard options were used.

In the remainder of this appendix, we specify the exact problems used for these measurements, describing both the system description used and the formula computed.
Food distribution

The food distribution problem is based on the academic/food_distribution/food_package.mcrl2 system. The version of this system that is part of the mCRL2 example collection uses a truck with a capacity of 320 units of food, yielding the “food-320” version of this problem; reducing this number to 300 yields the “food-300” version. The “food-320000” problem was produced by multiplying all food amounts in the food-320 version by 1000; finally, the “food-real” version was constructed by changing all variables indicating food amounts from being integer-valued to being real-valued.

The food distribution system description uses four action labels indicating truck movements to the four locations in the area, named moveA, moveB, moveC, and moveS, indicating movements towards the three villages and the supply center, respectively. Furthermore, it contains an action starve indicating that some village has used up its food supply.

The property verified for the four versions of the food distribution system is the proposition that a driving and distribution schedule is possible such that no village ever runs out of food. Using the action labels specified above, this is rendered as the formula $\nu X. (moveS \lor moveA \lor moveB \lor moveC) X \land \neg starve$, as specified in the sustained_delivery.mcf file.

Retransmission protocols

The different retransmission protocol problems are based on three different protocols, represented by three different system descriptions in the example collection: the academic/abp/abp.mcrl2 system describing the alternating bit protocol, the academic/cabp/cabp.mcrl2 system describing the concurrent alternating bit protocol, and the academic/onebit/onebit.mcrl2 system describing the sliding window protocol using a single window size bit. In the unmodified versions, each of these systems uses a two-element set as its payload domain; we modified each to produce a version using the integers as its payload domain instead. Together with the originals, this yields six different investigated systems: the “abp-bool”, “abp-int”, “cabp-bool”, “cabp-int”, “onebit-bool”, and “onebit-int” systems.

For each of those six systems, we verified five different properties, yielding a total of thirty model checking problems. The renderings of those properties as formulae in the modal $\mu$-calculus differs slightly between the three protocols, and to some degree between the Boolean and integer versions as well; here, we describe only the version for the “abp-bool” system, trusting that the other versions can be trivially reconstructed by the reader if necessary.

The exact actions used internally by the three different protocols differs considerably among them, but only four actions are referred to by the formulae below. All protocols have an action $r_1$ indicating the start of an attempt by the sending party to send a message payload; similarly, all have an action $s_4$ indicating that the receiving party successfully received such a payload. There is an action $c_3$ indicating the communication of a transmission error; finally, the $i$ action signifies any action internal to one of the communicating parties, which does not involve any communication.

The following five properties have been verified for each system:
• payload-fairness: Each payload that can be sent infinitely often is sent infinitely often; specified in \texttt{infinitely\_often\_enabled\_then\_infinitely\_often\_taken.mcf}. Encoded as
\[\forall d:D.([\text{true}]\nu X.\mu Y.\nu Z.([r1(d)]X \land ([r1(d)]false \lor \lnot r1(d))Y) \land \lnot r1(d)Z)).\]

• infinite-loss: A message can be lost infinitely often; specified in \texttt{infinitely\_often\_lost.mcf}. Encoded as
\[\langle \text{true}\rangle(\exists d:D.([r1(d)](\nu X.\mu Y.([c3]X \lor \lnot c3 \land \lnot s4(d))Y))).\]

• infinite-sending: For a specific payload, denoted \( p \), that payload can be sent infinitely often. Specified in \texttt{infinitely\_often\_receive\_d1.mcf}. Encoded as \( \nu X.\mu Y.([r1(p)]X \lor \lnot r1(p))Y)\).

• send-receive: Any given payload is received eventually; specified in \texttt{read\_then\_eventually\_send.mcf}. Encoded as \( [\text{true}](\forall d:D.([r1(d)](\nu X.\mu Y.([s4(d)]X \land \lnot s4(d))Y))).\)

• fair-send-receive: Any given payload is received eventually, as long as some messages eventually arrive correctly; specified in \texttt{read\_then\_eventually\_send\_if\_fair.mcf}. Encoded as \( \nu X.([\text{true}]X \land \forall d:D.([r1(d)](\nu Y.\mu Z.([\lnot s4(d)] \land [\lnot 1]Z \land [1]Y)))).\)

Other example problems

The “bakery” example problem involves Lamport’s bakery algorithm, described in \texttt{academic/bakery/bakery.mcrl2}. Among others, this system contains the \texttt{request(b)} and \texttt{enter(b)} actions, respectively describing the intent of party \( b \) to enter the critical section, and its success in that matter. The checked property, specified in \texttt{request\_must\_eventually\_enter.mcf}, is whether each requesting party will inevitably end up successfully entering the critical section, encoded as \( \nu X.([\text{true}]X \land \forall b:B.\texttt{request(b)})([\text{true}]Y \lor (\text{true}) \land (\text{true}) \lor \texttt{enter(b)}(\text{true})).\)

The “dining” example problem describes the dining philosophers problem; the mCRL2 example set contains several versions and formalizations of this system. The one we used for our tests is the \texttt{academic/dining/dining8.mcrl2} system. For this system, two properties are verified. The first, “nodeadlock” is whether this system is free from deadlocks, specified in \texttt{nodeadlock.mcf} as \( [\text{true}]([\text{true}])\mu Y.([\text{true}]Y \land (\text{true}) \lor \texttt{enter(b)}(\text{true})).\)

The “parallel” example problem describes a system consisting of three parallel deadlock-free ten-state automata, described in \texttt{academic/parallel/parallel.mcrl2}. The checked problem is whether this system is free from deadlocks, specified in \texttt{nodeadlock.mcf} as \( [\text{true}]([\text{true}]\mu Y.([\text{true}]Y \land (\text{true})\mu Y).\)

The “scheduler” example problem involves Milner’s scheduler, described in \texttt{academic/scheduler/scheduler.mcrl2}. For this system, the verified formula is whether this system is free from deadlocks, specified in \texttt{nodeadlock.mcf} as \( [\text{true}]([\text{true}]\mu Y.\).

Finally, the “trains” example problem describes a system of two trains using a piece of shared railroad, protected by a semaphore using Peterson’s algorithm.
This problem is described in academic/trains/trains.mcrl2, along with three variations using different mutual exclusion algorithms; a configuration variable in this file selects the mutual exclusion algorithm in use, which we modified to select Peterson's algorithm. For this system, as for the dining philosophers problem, two properties are verified. The first, “nodeadlock” is whether this system is free from deadlocks, specified in nodeadlock.mcf as "true∗\((true)\)true; the second, “fairness”, indicates that each train attempting to enter the critical section using an enter\(_p\) or enter\(_q\) action will eventually succeed in this, specified in infinitely\_often\_enabled\_then\_infinitely\_often\_taken.mcf as

\[(true∗\nu X.\mu Y.\nu Z.([enter\_p]X\land([enter\_p]\false\lor[\neg enter\_p]Y)\land[\neg enter\_p]Z))\land \]

\[(true∗\nu X.\mu Y.\nu Z.([enter\_q]X\land([enter\_q]\false\lor[\neg enter\_q]Y)\land[\neg enter\_q]Z)).\]