Evidence Extraction from Parameterised Boolean Equation Systems

Wieger Wesselink and Tim A.C. Willemse

Eindhoven University of Technology, Eindhoven, The Netherlands
{j.w.wesselink,t.a.c.willemse}@tue.nl

Abstract

Model checking is a technique for automatically assessing the quality of software and hardware systems and designs. Given a formalisation of both the system behaviour and the requirements the system should meet, a model checker returns either a *yes* or a *no*. In case the answer is not as expected, it is desirable to provide feedback to the user as to why this is the case. Providing such feedback, however, is not straightforward if the requirement is expressed in a highly expressive logic such as the modal μ-calculus, and when the decision problem is solved using intermediate formalisms. In this paper, we show how to extract witnesses and counterexamples from parameterised Boolean equation systems encoding the model checking problem for the first-order modal μ-calculus. We have implemented our technique in the modelling and analysis toolset mCRL2 and showcase our approach on a few illustrative examples.

1 Introduction

The complexity of the average computer-controlled system has reached a point at which it has become impossible to fully understand a system. By modelling a system and subsequently analysing whether the crucial safety and liveness requirements of the system are upheld, some confidence in the system's correctness can be obtained. The complexity of the average system, however, precludes that such an analysis can be conducted manually.

Model checking is an automated technique for assessing whether a requirement holds for a model of a system. This technique requires as input a mathematical description of the behaviour of a system, often given in terms of (a high-level description of) a *Kripke Structure* or *Labelled Transition System*, and a logical formula, often given in some appropriate temporal logic. By feeding both artefacts to a tool, colloquially referred to as the *model checker*, the *yes* or *no* verdict produced by the tool states whether (the model of) the system meets the requirement. Knowing that a system fails to meet a requirement, however, does not help to improve on the system design. For that, richer feedback in the form of evidence (*i.e.* a counterexample or a witness) of the model checker is required.

Depending on the logic that is required to perform the verification, however, it is not always clear what type of evidence must be extracted from a negative model checking exercise. While for linear time logic (LTL), a lasso, or a prefix of a lasso typically suffices, the problem becomes more pronounced for branching time logics such as CTL, CTL* or the modal μ-calculus. The reason for this is that the formulae over such logics are essentially interpreted over infinite computation trees.

In this paper, we describe how evidence can be constructed for an extension of the modal μ-calculus, *viz.* the *first-order modal μ-calculus* [12], within the context of the analysis toolset mCRL2 [5]. This logic extends the standard modal μ-calculus by adding first-order quantification and parameterised fixpoints. We draw inspiration from previous work [7] explaining how, in theory, evidence can be extracted from decision problems encoded in the logic of *Least Fixed Point* (LFP). The main idea is that a counterexample or witness for the model checking problem is a 'submodel' of the original model, which can be used to reconstruct the proof of the original negative model checking result.

One obstacle in applying the techniques outlined for LFP is that the model checker in mCRL2 uses *parameterised Boolean equation systems* (PBESs) [13] to solve the model checking problem. While LFP and PBESs have much in common, they differ in exactly those aspects used for evidence extraction. Our *first* contribution is therefore to show how this problem can effectively be overcome without changing the underlying theory for PBESs. Our *second* contribution is an implementation of our solution, illustrating the feasibility and appeal of the approach. As far as we are aware, ours is the first work demonstrating the feasibility of constructing diagnostics in this way for the full (first-order) modal μ-calculus with arbitrary alternation.

Related Work. We refer to Busard's PhD thesis [3] for a thorough overview of literature on generating diagnostics for model checking for the modal μ-calculus, but also other logics such as CTL and epistemic logics; we here give a concise overview of the most relevant related works. There are several works that address the problem of constructing diagnostics for the modal μ-calculus model checking problem. In [20], diagnostics is presented as a *game* played on the model checking game for the modal μ-calculus. A related approach is given by [14] who essentially suggests to generate *tableaux* as witnesses to the model checking problem. In [17], diagnostics are defined as explanations for the truth values of the underlying Boolean equation system that is used to solve a model checking problem for the alternation-free fragment of the modal μ-calculus. This technique is closely related to [20]. In contrast to our work, these approaches require the user conducting the verification to understand the underlying mechanism for conducting the verification. Finally, in the work by Tan and Cleaveland [21], which can be seen as a generalisation of [17], evidence is presented as information extracted from *(decorated) support sets*. These are closely related to the proof graphs underlying [6] and the idea of using decorations is reminiscent of the idea underlying [7] to allow first-order relations in proof graphs.

Outline. We give a cursory overview of the necessary background in Section 2; *i.e.* we briefly address the underlying theory for modelling data and system behaviours, the first-order modal μ-calculus, and we formalise the model checking problem. In Section 3, we introduce PBESs, we explain how the model checking problem can be converted to the problem of solving a PBES, and we illustrate the problem of extracting evidence from a PBES encoding a model checking problem. In Section 4, we illustrate how we can extract evidence from a PBES by modifying the encoding of the model checking problem and in Section 5, we illustrate how our solution works on practical examples. We conclude in Section 6.

2 Preliminaries

Our work is set in a context in which we rely on abstract data types to describe (and reason about) data. That is, we assume a given algebraic specification $\mathscr{S} = \langle S, \Sigma, E \rangle$ where S is a set of *sorts*, Σ is a family of *operation(s)* and E is a family of *equations*. As a convention, we write data sorts using letters D, E, *etcetera*. We have a set $\mathscr{D}$ of *data variables*, with typical elements $d, d_1, \ldots$

The semantics of an algebraic specification $\mathscr{S} = \langle S, \Sigma, E \rangle$ is given by a many-sorted Σ-algebra consisting of data domains corresponding to S and operations corresponding to Σ, satisfying the identities of E. We here adopt an initial algebra semantics point of view. For every sort D, $E, \ldots$, we denote the domains they represent by $\mathbb{D}$, $\mathbb{E}, \ldots$. For a closed term t of a given sort, say D (denoted $t{:}D$), we assume an interpretation function $[\![t]\!]$ that maps t to a value of $\mathbb{D}$ it represents. For open terms we use a *data environment* ε that maps each variable from $\mathscr{D}$ to a value of the proper domain. If we wish to indicate which variables may occur freely within a term t, we add these as 'parameters' to t; *i.e.* we write $t(d)$ to indicate that d is the only free variable in t. The interpretation of an open term t, denoted as $[\![t]\!]\varepsilon$ is obtained in the standard way. We write $\varepsilon[d \mapsto v]$ for the environment that maps variable d to value v and all other variables d' are mapped to $\varepsilon(d')$.

We require the presence of a sort B representing the Booleans $\mathbb{B} = \{true, false\}$. For convenience we also assume the existence of the sort N representing the natural numbers $\mathbb{N}$. For both sorts we assume the usual operators are available and we do not write constants or operators in the syntactic domain any different from their semantic counterparts. For example, the Boolean value *true* is represented by the constant *true* and the value *false* is represented by the constant *false*. The syntactic operator $\_ \wedge \_ {:} B \times B \to B$ corresponds to the usual, semantic conjunction $\_ \wedge \_ {:} \mathbb{B} \times \mathbb{B} \to \mathbb{B}$; *etcetera*.

Note that for readability, and without loss of generality, we use a single—possibly compound—data type in our definitions and formal statements.

2.1 Processes

The behaviours of software and hardware systems can be adequately modelled using labelled transition systems (LTSs). Modelling languages, such as process algebras and I/O-automata, provide language constructs such as conditional choice, interleaving parallelism and sequential composition through which one can compactly specify such systems. A useful normal form to which many such specifications can be compiled automatically is the *Linear process equation* (LPE) format, see *e.g.* [11].

An LPE typically models the (global) state of a (software or hardware) system by means of a finite vector of formal data parameters, ranging over adequately chosen sorts. The behaviours are described by a non-deterministic choice (denoted by the $+$ operator) among rules taken from a finite set of *condition-action-effect* rules, prescribing under which condition an action (modelling a message exchange or an event) is enabled, leading to an update of the vector of formal parameters. A formal definition of an LPE, employing syntax that stays true to its process-algebraic heritage, is given below.

Definition 1. A *linear process equation* is an equation of the following form:

$$L(d_L{:}D_L) = +\{ \sum_{e_\mathtt{a}:E_\mathtt{a}} c_\mathtt{a}(d_L, e_\mathtt{a}) \to \mathtt{a}(f_\mathtt{a}(d_L, e_\mathtt{a})) \cdot L(g_\mathtt{a}(d_L, e_\mathtt{a})) \mid \mathtt{a} \in \mathscr{A}ct\}$$

The sort D_L is used to represent the set of states and variable d_L represents a state; $\mathtt{a}$ is a (typed) *action label* taken from a finite set $\mathscr{A}ct$ of (typed) action labels. Each action label $\mathtt{a}$ is associated with a *local variable* $e_\mathtt{a}$ of sort $E_\mathtt{a}$, an expression $c_\mathtt{a}{:}B$, which acts as a *condition*, an expression $f_\mathtt{a}{:}D_\mathtt{a}$, which yields an argument emitted along $\mathtt{a}$ when executed, and an operation $g_\mathtt{a}{:}D_L$, which yields a new state. We require that d_L and e_L are the only free variables occurring in these expressions.

The *interpretation* of L with an *initial state* represented by closed term $e{:}D_L$, denoted by $L(e)$, is a labelled transition system $M = \langle S, A, \to, s_0 \rangle$, where:

- $S = \mathbb{D}_L$ with initial state $s_0 = [\![e]\!] \in S$;
- $A = \{\mathtt{a}(v_\mathtt{a}) \mid \mathtt{a} \in \mathscr{A}ct, v_\mathtt{a} \in \mathbb{D}_\mathtt{a}\}$ is the (possibly infinite) set of *actions*;
- $\to \subseteq S \times A \times S$ is the set of transitions, where $v \xrightarrow{\mathtt{a}(v_\mathtt{a})} w$ holds if and only if for some $u \in \mathbb{E}_\mathtt{a}$ and environment ε:
  - $[\![f_\mathtt{a}(d_L, e_\mathtt{a})]\!]\varepsilon[d_L \mapsto v, e_\mathtt{a} \mapsto u] = v_\mathtt{a}$ and $[\![g_\mathtt{a}(d_L, e_\mathtt{a})]\!]\varepsilon[d_L \mapsto v, e_\mathtt{a} \mapsto u] = w$,
  - $[\![c_\mathtt{a}(d_L, e_\mathtt{a})]\!]\varepsilon[d_L \mapsto v, e_\mathtt{a} \mapsto u]$ evaluates to true.

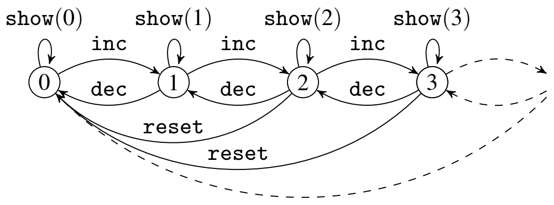
Throughout this paper, whenever we refer to an LPE L we implicitly mean the LPE as given by Def. 1, with d_L being the state parameter of sort D_L of process L. Note that an LPE L compactly expresses that in a state, represented by parameter d_L, whenever condition $c_\mathtt{a}(d_L, d_\mathtt{a})$ holds (for some non-deterministically chosen value for variable $d_\mathtt{a}$), then action $\mathtt{a}$ carrying data parameter $f_\mathtt{a}(d_L, d_\mathtt{a})$ can be executed, effectively changing the global state to $g_\mathtt{a}(d_L, d_\mathtt{a})$.

Notation 1. *In our examples we permit ourselves to be less strict in following the LPE format and allow for multiple summands ranging over the same action label. Moreover, in case a summand binds a local* *variable $d_{\mathtt{a}}$ which does not occur in the expressions $c_{\mathtt{a}}, f_{\mathtt{a}}$ and $g_{\mathtt{a}}$, we omit the $\sum$-symbol altogether. In our examples, action labels can carry zero or more arguments. Whenever condition $c_{\mathtt{a}}$ is the constant true, we omit both the condition and the $\rightarrow$ symbol.*

Example 1. As a running example, we consider a small system modelling a simple counter that can increase and decrease a parameter n, reset that parameter to 0 when it has a positive value, and show the current value. The system is described by $L(0)$, where LPE L is given below.

$$\begin{array}{rcl} L(n{:}N) & = & \mathtt{inc} \cdot L(n+1) \\ & + & (n > 0) \rightarrow \mathtt{dec} \cdot L(n-1) \\ & + & (n > 1) \rightarrow \mathtt{reset} \cdot L(0) \\ & + & \mathtt{show}(n) \cdot L(n) \end{array}$$

Parts of the (infinite) labelled transition system associated to the LPE are depicted below; the dashed edges indicate the presence of one (or more) edges.



□

2.2 First-Order Modal μ-Calculus

Model checking is concerned with checking whether a modal property holds for a given system or not. The *first-order modal μ-calculus* (μ-calculus for short) of [18, 12] is a highly-expressive language for stating such properties. This logic is based on the standard modal μ-calculus [2], extended with first-order quantification and parameterised fixpoints, adding data as a first-class citizen. We briefly review its syntax and semantics, and we demonstrate its use by means of several small examples.

Definition 2. The μ-calculus, ranging over a set of (typed) action labels $\mathscr{A}ct$ is given by the following BNF grammar; formula φ represents a *state formula* and formula α represents an *action formula*:

$$\begin{array}{rcl} \varphi & ::= & b \mid Z(e) \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid [\alpha]\varphi_1 \mid \forall d{:}D.\ \varphi \mid \nu Z(d{:}D = e).\ \varphi \\ \alpha & ::= & \mathtt{a}(e_{\mathtt{a}}) \mid b \mid \neg\alpha_1 \mid \alpha_1 \wedge \alpha_2 \mid \forall d{:}D.\ \alpha_1 \end{array}$$

b is an expression of sort B; e is a data expression of type D; $Z{:}D \rightarrow B$ is a fixpoint variable from a set of fixpoint variables $\mathscr{P}$; for simplicity, we assume all fixpoint variables range over the same sort D. Expressions of the form $(\nu Z(d{:}D = e).\ \varphi)$ are subject to the restriction that any free occurrence of Z in φ must be within the scope of an even number of negation symbols $\neg$. Finally, $\mathtt{a}$ is an action label from the set $\mathscr{A}ct$ and expression $e_{\mathtt{a}}$ of sort $D_{\mathtt{a}}$ is a parameter of $\mathtt{a}$.

For reference we include the semantics of a μ-calculus formula in Table 1. Note that the ordered set $\langle[\mathbb{D} \rightarrow 2^S], \sqsubseteq\rangle$ is a complete lattice, where $[\mathbb{D} \rightarrow 2^S]$ is the set of functions from $\mathbb{D}$ to subsets of S and $\sqsubseteq$ is defined as $f \sqsubseteq g$ iff for all $v \in \mathbb{D}$, we have $f(v) \subseteq g(v)$. Since the functionals $\Phi_{\rho\varepsilon}$ are monotonic over this lattice, the interpretation of fixpoint expressions is then justified [22].

In the remainder of this paper, we use the following standard abbreviations for μ-calculus formulae φ, action formulae α and (both μ-calculus formulae and action formulae) ψ.

$$\begin{array}{lcl} (\psi_1 \vee \psi_2) & = & \neg(\neg\psi_1 \wedge \neg\psi_2) \\ \langle\alpha\rangle\varphi & = & \neg[\alpha]\neg\varphi \\ \exists d{:}D.\psi & = & \neg\forall d{:}D.\neg\psi \\ \mu Z(d{:}D = e).\varphi & = & \neg\nu Z(d{:}D{=}e).\ \neg\varphi[Z := \neg Z] \end{array}$$

Table 1: The interpretation of a μ-calculus formula φ and action formula α, denoted by $[\![\varphi]\!]\rho\varepsilon$ and $\|\,\alpha\,\|\varepsilon$, respectively, in the context of environments ε and ρ and an LTS $M = \langle S, A, \rightarrow, s_0\rangle$.

$[\![b]\!]\rho\varepsilon$	$=$	$\begin{cases} S \text{ if } [\![b]\!]\varepsilon \text{ is true} \\ \emptyset \text{ otherwise} \end{cases}$
$[\![Z(e)]\!]\rho\varepsilon$	$=$	$\rho(Z)([\![e]\!]\varepsilon)$
$[\![\neg\varphi]\!]\rho\varepsilon$	$=$	$S \setminus [\![\varphi]\!]\rho\varepsilon$
$[\![\varphi_1 \wedge \varphi_2]\!]\rho\varepsilon$	$=$	$[\![\varphi_1]\!]\rho\varepsilon \cap [\![\varphi_2]\!]\rho\varepsilon$
$[\![[\alpha]\varphi]\!]\rho\varepsilon$	$=$	$\{w{\in}S \mid \forall w'{\in}S\ \forall a{\in}A\ (w \xrightarrow{a} w' \wedge a{\in}\|\,\alpha\,\|\varepsilon) \Rightarrow w'{\in}[\![\varphi]\!]\rho\varepsilon\}$
$[\![\forall d{:}D.\varphi]\!]\rho\varepsilon$	$=$	$\bigcap_{v\in\mathbb{D}}[\![\varphi]\!]\rho(\varepsilon[d \mapsto v])$
$[\![\nu Z(d{:}D{=}e).\ \varphi]\!]\rho\varepsilon$	$=$	$(\nu\Phi_{\rho\varepsilon})([\![e]\!]\varepsilon)$, *where* $\Phi_{\rho\varepsilon}{:}(\mathbb{D} \rightarrow 2^S) \rightarrow (\mathbb{D} \rightarrow 2^S)$ *is defined as:* $\Phi_{\rho\varepsilon}(F) = \lambda v \in \mathbb{D}.[\![\varphi]\!](\rho[Z \mapsto F])(\varepsilon[d \mapsto v])$ *for* $F{:}\mathbb{D} \rightarrow 2^S$
$\|\,b\,\|\varepsilon$	$=$	$\begin{cases} A \text{ if } [\![b]\!]\varepsilon \text{ is true} \\ \emptyset \text{ otherwise} \end{cases}$
$\|\,\mathtt{a}(e_{\mathtt{a}})\,\|\varepsilon$	$=$	$\{\mathtt{a}([\![e_{\mathtt{a}}]\!]\varepsilon)\}$
$\|\,\neg\alpha\,\|\varepsilon$	$=$	$A \setminus \|\,\alpha\,\|\varepsilon$
$\|\,\alpha_1 \wedge \alpha_2\,\|\varepsilon$	$=$	$\|\,\alpha_1\,\|\varepsilon \cap \|\,\alpha_2\,\|\varepsilon$
$\|\,\forall d{:}D.\alpha\,\|\varepsilon$	$=$	$\bigcap v{\in}D\ \|\,\alpha\,\|\varepsilon[d \mapsto v]$

A μ-calculus formula (in the language enriched with the above abbreviations) is in *Positive Normal Form* (PNF) whenever negation only occurs at the lowest level and all bound variables are distinct. We only consider formulae in PNF. Note that this is no restriction as every μ-calculus formula can be converted to PNF using suitable α-renaming and logical rules such as *De Morgan*. Moreover, we only consider μ-calculus formulae that are *closed*: data variables d only occur in the scope of a quantifier or a fixpoint binding it, and each fixpoint variable Z only occurs in the scope of a fixpoint that binds it. Since the semantics of closed formulae is independent from the environments ε and ρ, we typically write $[\![\varphi]\!]$ rather than $[\![\varphi]\!]\rho\varepsilon$ for closed φ. A formula φ is *normalised* whenever none of its subformulae of the form $\sigma X(d{:}D = e).\ \psi$ contain unbound data variables. Every closed formula can be converted (in linear time) to an equivalent normalised formula, see *e.g.* [16].

We are mainly concerned with the problem of deciding (and explaining) whether a given LPE $L(e)$ meets a logical specification φ given by a μ-calculus formula; this is known as the *model checking problem*. That is, we wish to decide whether $[\![e]\!] \in [\![\varphi]\!]$; we write $L(e) \models \varphi$ to denote just this.

We finish this section with a few illustrative examples of the use of data in formulae and parameterisation of fixpoints.

Example 2. Consider the LPE modelling the counter. Below are some simple properties of the counter, expressed in the μ-calculus.

1. the counter is *deadlock-free*: $\nu X.([\mathit{true}]X \wedge \langle \mathit{true}\rangle \mathit{true})$;
2. the counter can be incremented *ad infinitum*: $\nu X.\langle \mathtt{inc}\rangle X$;
3. the counter can alternatingly increase and decrease *ad infinitum*: $\nu X.\langle \mathtt{inc}\rangle\langle \mathtt{dec}\rangle X$;
4. the counter can be decreased *infinitely often*: $\nu X.\mu Y.(\langle \mathtt{dec}\rangle X \vee \langle \mathtt{inc}\rangle Y)$;
5. the counter can take on any natural number: $\forall n{:}N.\mu X.(\langle \mathtt{show}(n)\rangle \mathit{true} \vee \langle \mathit{true}\rangle X)$.

Table 2: The semantics $[\![\varphi]\!]\eta\varepsilon$ of a predicate formula φ is a truth assignment given in the context of a data environment ε and a *predicate* environment $\eta{:}\mathscr{X} \to (\mathbb{D}_{\mathscr{X}} \to \mathbb{B})$.

$$\begin{array}{lcl}
[\![b]\!]\eta\varepsilon & = & [\![b]\!]\varepsilon \\
[\![X(e)]\!]\eta\varepsilon & = & \eta(X)([\![e]\!]\varepsilon) \\
[\![\varphi_1 \vee \varphi_2]\!]\eta\varepsilon & = & [\![\varphi_1]\!]\eta\varepsilon \text{ or } [\![\varphi_2]\!]\eta\varepsilon \\
[\![\varphi_1 \wedge \varphi_2]\!]\eta\varepsilon & = & [\![\varphi_1]\!]\eta\varepsilon \text{ and } [\![\varphi_2]\!]\eta\varepsilon \\
[\![\exists d{:}D.\varphi]\!]\eta\varepsilon & = & [\![\varphi]\!]\eta(\varepsilon[d \mapsto v]) \text{ for some } v \in \mathbb{D} \\
[\![\forall d{:}D.\varphi]\!]\eta\varepsilon & = & [\![\varphi]\!]\eta(\varepsilon[d \mapsto v]) \text{ for all } v \in \mathbb{D}
\end{array}$$

6. on all paths, the counter can decrease as often as it has increased:
   $\nu X(m{:}N = 0).([\mathtt{inc}]X(m+1) \wedge [\mathtt{dec}]X(m-1) \wedge [\neg\mathtt{inc} \wedge \neg\mathtt{dec}]X(m) \wedge (m = 0 \vee \langle\mathtt{dec}\rangle \mathit{true}))$;

Note that all of the above properties hold for the initial state of the counter except for the last one: due to the reset action that can take place at any moment, the system may return to the initial state in which it can no longer perform a `dec` action. □

3 Model Checking using Parameterised Boolean Equation Systems

Solving the first-order modal μ-calculus model checking problem can be done in various ways. We here focus on the use of an intermediate formalism called *parameterised Boolean equation systems* [13]. These equation systems underlie several verification toolsets for specifying and analysing software and hardware systems, such as the CADP and mCRL2 toolsets. The advantage of using an intermediate formalism is that it allows for building dedicated techniques for that formalism [13, 12].

Parameterised Boolean Equation Systems are sequences of fixpoint equations where each equation is of the form $\nu X(d{:}D) = \varphi$ or $\mu X(d{:}D) = \varphi$. A parameterised Boolean equation is, in a sense, a simplified and equational variant of a first-order μ-calculus formula, lacking modal operators. We refer to the left-hand side variable of a parameterised Boolean equation as a *predicate variable*, whereas the right-hand side formula is called a *predicate formulae*.

Definition 3. A *parameterised Boolean equation system* $\mathscr{E}$ is a system of equations defined by:

$$\begin{array}{rl}
\mathscr{E} ::= & \emptyset \mid (\mu X(d_X{:}D_{\mathscr{X}}) = \varphi)\ \mathscr{E} \mid (\nu X(d_X{:}D_{\mathscr{X}}) = \varphi)\ \mathscr{E} \\
\varphi ::= & b \mid X(e) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists d{:}D.\varphi \mid \forall d{:}D.\varphi
\end{array}$$

b is an expression of sort B, X is a predicate variable taken from some sufficiently large set of typed predicate variables $\mathscr{X}$, d_X is a data parameter of sort $D_{\mathscr{X}}$, d is a data variable of sort D and e is a data expression of the appropriate sort. Again for simplicity, we assume that all predicate variables range over the same sort $D_{\mathscr{X}}$.

We only consider *well-formed, closed* PBESs in this paper. A PBES $\mathscr{E}$ is said to be *well-formed* iff every predicate variable $X \in \mathsf{bnd}(\mathscr{E})$, where $\mathsf{bnd}(\mathscr{E})$ is the set of *bound* variables (those predicate variables occurring at the left-hand side of the equations), occurs at the left-hand side of precisely one equation of $\mathscr{E}$. A PBES $\mathscr{E}$ is said to be *closed* iff (1) for each right-hand side predicate formula φ occurring in $\mathscr{E}$ we have $\mathsf{occ}(\varphi) \subseteq \mathsf{bnd}(\mathscr{E})$, where $\mathsf{occ}(\varphi)$ contains all predicate variables occurring in

φ, and, (2) whenever the only free data variable occurring in φ is the data parameter occurring at the left-hand side of φ's equation.

The interpretation of predicate formulae is listed in Table 2; for the fixpoint semantics of a PBES we refer to *e.g.* [13]. Instead of the fixpoint semantics we here focus on the equivalent *proof graph* semantics provided in [6]. This semantics is both more accessible (operational) and better suits our needs of computing counterexamples and witnesses. The proof graph semantics of a PBES explains the *value*, or *truth assignment* for each bound predicate variable by means of a directed graph with vertices ranging over the *signatures* of a PBES. A signature is a tuple (X,v) for $X \in \mathsf{bnd}(\mathcal{E})$ and $v \in \mathbb{D}_{\mathcal{X}}$ a value taken from the domain underlying the type of X; that is, we set $\mathsf{sig}(\mathcal{E}) = \{(X,v) \mid v \in \mathbb{D}_{\mathcal{X}}, X \in \mathsf{bnd}(\mathcal{E})\}$. We order each pair of variables $X,Y \in \mathsf{bnd}(\mathcal{E})$ as follows: $X \leq Y$ iff the equation for Y follows that of X and we write $X < Y$ iff $X \leq Y$ and $X \neq Y$. Moreover, we say that a variable X is a ν-variable whenever X occurs at the left-hand side of a greatest fixpoint equation; otherwise, X is a μ-variable.

Definition 4. A graph $\mathcal{G} = (V,E)$, for $V \subseteq \mathsf{sig}(\mathcal{E})$ and $E \subseteq V \times V$ is a *proof graph* iff:

1. for all equations $(\sigma X(d_X{:}D_{\mathcal{X}}) = \varphi)$ in $\mathcal{E}$ for which $(X,v) \in V$, $[\![\varphi]\!]\Theta_{(X,v)}\varepsilon[d_X \mapsto v]$ holds for some ε and where $\Theta_{(X,v)}$ is the environment defined as follows: $\Theta_{(X,v)}(Y) = \{w \in \mathbb{D}_{\mathcal{X}} \mid \langle (X,v),(Y,w) \rangle \in E\}$ for all Y;
2. for all infinite paths $(X_1,v_1)\ (X_2,v_2)\ \dots$ through $\mathcal{G}$, the smallest variable (w.r.t. the ordering $<$) occurring infinitely often on that path is a ν-variable.

The semantics (often referred to as the (partial) *solution*) of a PBES is as follows; the correspondence with the more traditional fixpoint semantics follows from, *e.g.* [6].

Definition 5. Let $\mathcal{E}$ be a PBES. The semantics of $\mathcal{E}$ is a predicate environment $[\![\mathcal{E}]\!]$ defined as follows: $v \in [\![\mathcal{E}]\!](X)$ iff $X \in \mathsf{bnd}(\mathcal{E})$ and there is a proof graph $\mathcal{G} = (V,E)$ such that $(X,v) \in V$.

The *dual* of a proof graph is called a *refutation graph*. Such a graph explains that a value v does *not* belong to the set of values defined by some variable X. The notion of a refutation graph is defined analogously to a proof graph, using complementation for $\Theta_{(X,v)}$ in condition 1 and requiring that the least variable occurring on any infinite path in the graph is a μ-variable.

A proof graph containing a vertex (X,v) provides *evidence* that the value v belongs to the set of values defined by X in the PBES. In that case, the first condition essentially states that v belongs to X because all successors of (X,v) together yield an environment that makes the right-hand side formula for X hold when parameter d_X is assigned value v. The second condition ensures that the graph respects the *parity condition* typically associated with (nested) fixpoint formulae.

Example 3. Consider the PBES $(\nu X(n{:}N) = Y(n))\ (\mu Y(n{:}N) = (n > 0 \land X(n-1)) \lor Y(n+1))$. A proof graph for this PBES is given below:

$$\begin{tikzcd}
(X,0) \arrow[r] & (Y,0) \arrow[r] & (Y,1) \arrow[ll, bend left]
\end{tikzcd}$$

Note that there are an infinite number of proof graphs containing $(X,0)$. An alternative proof graph is, *e.g.* the following:

$$\begin{tikzcd}
(X,0) \arrow[r] & (Y,0) \arrow[r] & (Y,1) \arrow[r] \arrow[ll, bend left] & (Y,2) \arrow[r] & (X,1) \arrow[ll, bend left]
\end{tikzcd}$$

From the first proof graph it follows that $\{0\} \subseteq [\![\mathcal{E}]\!](X)$, whereas from the second proof graph it follows that $\{0,1\} \subseteq [\![\mathcal{E}]\!](X)$ and $\{0,1,2\} \subseteq [\![\mathcal{E}]\!](Y)$. Note that it is straightforward to show, by extending the given proof graphs, that for any natural number $k \in \mathbb{N}$, we have $k \in [\![\mathcal{E}]\!](X)$ and $k \in [\![\mathcal{E}]\!](Y)$. $\square$

A proof graph is *minimal* if leaving out vertices or edges yields a graph that violates the proof graph conditions. We here note that the problem of computing a minimal proof graph (*i.e.* a subgraph of a proof graph that is as small as possible), is NP hard [6]. However, some techniques for computing a proof graph, such as solving a parity game induced by a PBES and using the winning strategies to filter unreachable vertices, yield minimal proof graphs by definition. Note that minimality of a proof graph does not imply the non-existence of a proof graph that is strictly smaller, see the example below.

Example 4. Consider the two proof graphs of Example 3. The first graph is a *minimal* proof graph and it is also the *smallest possible* proof graph. The second proof graph is clearly not minimal; however, by omitting the edge from $(Y,1)$ to $(X,0)$ we obtain another minimal proof graph. The thus obtained proof graph is clearly not the smallest possible proof graph. □

There are numerous ways in which a PBES $\mathcal{E}$ can be partially solved, *i.e.* for which we can decide, for a given bound predicate variable $X \in \mathsf{bnd}(\mathcal{E})$ and value v, whether $v \in [\![\mathcal{E}]\!](X)$. For instance, one can use *Gauß Elimination and symbolic approximation* [12], or by constructing and solving a parity game that is induced by a PBES [18].

As we remarked at the start of this section, the usefulness of PBESs lies in the observation that there is a linear-time reduction of the first-order modal μ-calculus model checking problem for LPEs to PBESs, see *e.g.* [12]. This transformation generalises the reduction of the (standard) modal μ-calculus for LTSs to *Boolean equation systems* [15]. For reference, we provide the details for this reduction in Table 3; in the next section, we present our modifications to this transformation. The correctness of the original transformation is given by the following theorem, taken from [12].

Theorem 1. Let formula $\sigma X(d{:}D = e').\ \psi$ be a closed, normalised first-order modal μ-calculus formula and $L(e)$ be an LPE. Then $L(e) \models \sigma X(d{:}D = e').\ \psi$ iff $([\![e]\!],[\![e']\!]) \in [\![\mathbf{E}_L(\sigma X(d{:}D = e').\ \psi)]\!](X)$. □

Example 5. We illustrate the transformation on the counter modelled by LPE $L(0)$ of Example 1. Say that we wish to analyse whether the counter can be decreased infinitely often, *i.e.* property (4) of Example 2: $\nu X.\mu Y.(\langle \mathtt{dec}\rangle X \vee \langle \mathtt{inc}\rangle Y)$. The PBES resulting from the transformation (after logical simplification, eliminating all subformulae obtained from summands not matching the action labels in the modal operators) is the PBES of Example 3. Note that, per Theorem 1, the proof graphs given there illustrate that the property holds for $L(0)$.

Next, suppose we wish to verify the property that we can infinitely often alternatingly increase and decrease parameter n, *i.e.* property (3) of Example 2. Recall that this is formalised by the following μ-calculus formula $\nu X.\langle \mathtt{inc}\rangle\langle \mathtt{dec}\rangle X$. The PBES $\mathcal{E}$ we obtain from this property (again after some logical simplification) is as follows:

$$\nu X(n{:}N) = n+1 > 0 \wedge X((n+1)-1)$$

It follows, by definition, that $0 \in [\![\mathcal{E}]\!](X)$, see the proof graph given below.

$$\begin{tikzcd} (X,0) \arrow[loop right] \end{tikzcd}$$

Consequently, by Theorem 1 we find that also $L(0) \models \nu X.\langle \mathtt{inc}\rangle\langle \mathtt{dec}\rangle X$. □

In [6], proof graphs and refutation graphs for PBESs were introduced in an effort to provide a meaningful explanation of the answer to a decision problem encoded in a PBES. While at the level of a PBES these graphs indeed meet their objective, as they provide the exact reasoning underlying the (partial) solution of a PBES, they do not aid in understanding the solution at the level of the decision problem that is encoded in the PBES. This is clearly illustrated in, *e.g.* the last proof graph for the PBES underlying the model checking problem of Example 5: the single vertex with a self-loop, constituting the proof graph, does not explain which transitions of the LTS of Example 5 are involved.

Table 3: Translation scheme for encoding the problem $L \models \sigma X(d{:}D_{\mathscr{X}} = e').\ \psi$ into an equation system. Recall that parameter d_L of sort D_L, occurring in the rules, originates from LPE L of Def. 1; likewise, the expressions c_{a}, f_{a} and g_{a} originate from this LPE.

$$\begin{array}{lcll}
\mathbf{E}_L(b) & = & \varepsilon & \\
\mathbf{E}_L(X(e)) & = & \varepsilon & \\
\mathbf{E}_L(\varphi \oplus \psi) & = & \mathbf{E}_L(\varphi)\ \mathbf{E}_L(\psi) & \text{for } \oplus \in \{\wedge, \vee\} \\
\mathbf{E}_L(\mathsf{Q}\, d{:}D.\varphi) & = & \mathbf{E}_L(\varphi) & \text{for } \mathsf{Q} \in \{\forall, \exists\} \\
\mathbf{E}_L([\alpha]\varphi) & = & \mathbf{E}_L(\varphi) & \\
\mathbf{E}_L(\langle\alpha\rangle\varphi) & = & \mathbf{E}_L(\varphi) & \\
\mathbf{E}_L(\sigma X(d{:}D_{\mathscr{X}} = e).\ \psi) & = & (\sigma X(d_L{:}D_L, d{:}D_{\mathscr{X}}) = \mathbf{RHS}_L(\psi))\ \mathbf{E}_L(\psi) & \\
\mathbf{RHS}_L(b) & = & b & \\
\mathbf{RHS}_L(X(e)) & = & X(d_L, e) & \\
\mathbf{RHS}_L(\varphi \oplus \psi) & = & \mathbf{RHS}_L(\varphi) \oplus \mathbf{RHS}_L(\psi) & \text{for } \oplus \in \{\wedge, \vee\} \\
\mathbf{RHS}_L(\mathsf{Q}\, d{:}D.\varphi) & = & \mathsf{Q}\, d{:}D.\ \mathbf{RHS}_L(\varphi) & \text{for } \mathsf{Q} \in \{\forall, \exists\} \\
\mathbf{RHS}_L([\alpha]\varphi) & = & \bigwedge_{\mathsf{a} \in \mathscr{A}ct} \forall e_{\mathsf{a}}{:}D_{\mathsf{a}}.\ (c_{\mathsf{a}}(d_L, e_{\mathsf{a}}) \wedge \mathsf{match}(\mathsf{a}(f_{\mathsf{a}}(d_L, e_{\mathsf{a}})), \alpha)) & \\
 & & \quad \implies (\mathbf{RHS}_L(\varphi)[g_{\mathsf{a}}(d_L, e_{\mathsf{a}})/d_L]) & \\
\mathbf{RHS}_L(\langle\alpha\rangle\varphi) & = & \bigvee_{\mathsf{a} \in \mathscr{A}ct} \exists e_{\mathsf{a}}{:}D_{\mathsf{a}}.\ (c_{\mathsf{a}}(d_L, e_{\mathsf{a}}) \wedge \mathsf{match}(\mathsf{a}(f_{\mathsf{a}}(d_L, e_{\mathsf{a}})), \alpha) & \\
 & & \quad \wedge (\mathbf{RHS}_L(\varphi)[g_{\mathsf{a}}(d_L, e_{\mathsf{a}})/d_L])) & \\
\mathbf{RHS}_L(\sigma X(d{:}D_{\mathscr{X}} = e).\ \varphi) & = & X(d_L, e) & \\
\mathsf{match}(\mathsf{a}(v), b) & = & b & \\
\mathsf{match}(\mathsf{a}(v), \mathsf{a}'(e')) & = & (v = e') \wedge (\mathsf{a} = \mathsf{a}') & \\
\mathsf{match}(\mathsf{a}(v), \neg\alpha) & = & \neg\mathsf{match}(\mathsf{a}(v), \alpha) & \\
\mathsf{match}(\mathsf{a}(v), \alpha \wedge \beta) & = & \mathsf{match}(\mathsf{a}(v), \alpha) \wedge \mathsf{match}(\mathsf{a}(v), \beta) & \\
\mathsf{match}(\mathsf{a}(v), \forall d{:}D.\ \alpha) & = & \forall d{:}D.\ \mathsf{match}(\mathsf{a}(v), \alpha) &
\end{array}$$

4 Evidence Extraction from PBESs

Whenever a model checking problem yields an unexpected verdict, evidence in the form of a *witness* or *counterexample* supporting that verdict can be helpful in analysing the root cause. Following [7], such a witness or counterexample is a fragment of the original labelled transition system (or LPE) that can be used to reconstruct the model checking verdict. However, a proof graph underlying a PBES that encodes a model checking problem lacks some essential information about the LPE to extract a counterexample or witness, see Example 5. This issue was recognised and further studied in [7] in a slightly different setting, *viz.* in the setting of the logic *Least Fixed Point* (LFP).

The solution proposed in [7] is to extend the proof graphs with information about *first-order* relation symbols from the structures involved in the encoded decision problem. Using proof graphs enriched in this way, evidence (*e.g.* counterexamples or witnesses) can be extracted from the proof graphs by projecting onto the vertices referring to the relational symbols of the desired structure(s). The thus obtained structures are weak substructures of the original structures that allow for reconstructing the proof graph underlying the original PBES.

Porting the proposed solution to the setting to PBESs is, however, not straightforward. The reason for this is that, unlike in LFP formulae, one cannot refer to first-order relational symbols; one can essentially only refer to Boolean expressions and predicate variables. Consequently, proof graphs for PBESs cannot be enriched in a similar fashion.

We propose to solve this problem by adding the information from the structures, relevant for constructing proper diagnostics, to the encoding of the decision problem. For instance, if the diagnostics for a model checking problem requires that a weak substructure of the original structures can be reconstructed, then we add information about that structure to our encoding. The extra information is added in the form of additional equations and by adding predicate variables that refer to those equations. For the model checking problem, this only requires changing the rules for $\mathbf{RHS}_L([\alpha]\varphi)$ and $\mathbf{RHS}_L(\langle\alpha\rangle\varphi)$:

$$\begin{array}{rcl}
\mathbf{RHS}_L([\alpha]\varphi) & = & \bigwedge_{\mathtt{a}\in\mathscr{A}ct} \forall e_{\mathtt{a}}{:}E_{\mathtt{a}}.\ (c_{\mathtt{a}}(d_L,e_{\mathtt{a}}) \wedge \mathsf{match}(\mathtt{a}(f_{\mathtt{a}}(d_L,e_{\mathtt{a}})),\alpha)) \\
& & \quad \implies \Big(\big(\mathbf{RHS}_L(\varphi)[g_{\mathtt{a}}(d_L,e_{\mathtt{a}})/d_L] \wedge L^+_{\mathtt{a}}(d_L,f_{\mathtt{a}}(d_L,e_{\mathtt{a}}),g_{\mathtt{a}}(d_L,e_{\mathtt{a}}))\big) \\
& & \qquad \vee L^-_{\mathtt{a}}(d_L,f_{\mathtt{a}}(d_L,e_{\mathtt{a}}),g_{\mathtt{a}}(d_L,e_{\mathtt{a}}))\Big) \\
\mathbf{RHS}_L(\langle\alpha\rangle\varphi) & = & \bigvee_{\mathtt{a}\in\mathscr{A}ct} \exists e_{\mathtt{a}}{:}E_{\mathtt{a}}.\ \Big(c_{\mathtt{a}}(d_L,e_{\mathtt{a}}) \wedge \mathsf{match}(\mathtt{a}(f_{\mathtt{a}}(d_L,e_{\mathtt{a}})),\alpha) \\
& & \quad \wedge \Big(\big(\mathbf{RHS}_L(\varphi)[g_{\mathtt{a}}(d_L,e_{\mathtt{a}})/d_L] \vee L^-_{\mathtt{a}}(d_L,f_{\mathtt{a}}(d_L,e_{\mathtt{a}}),g_{\mathtt{a}}(d_L,e_{\mathtt{a}}))\big) \\
& & \qquad \wedge L^+_{\mathtt{a}}(d_L,f_{\mathtt{a}}(d_L,e_{\mathtt{a}}),g_{\mathtt{a}}(d_L,e_{\mathtt{a}}))\Big)\Big)
\end{array}$$

All remaining rules remain as before. The two additional equations that must be added to the translation for each action label a, are as follows:

$$(\nu L^+_{\mathtt{a}}(d_L{:}D_L, d_{\mathtt{a}}{:}D_{\mathtt{a}}, d'_L{:}D_L) = \mathit{true}) \;\; (\mu L^-_{\mathtt{a}}(d_L{:}D_L, d_{\mathtt{a}}{:}D_{\mathtt{a}}, d'_L{:}D_L) = \mathit{false})$$

Note that their exact position in the resulting PBES does not matter (their solutions are independent of other equations as they are simple constants, so in a proof graph or refutation graph, vertices of the form $(L^+_{\mathtt{a}}, v, v_{\mathtt{a}}, w)$ or $(L^-_{\mathtt{a}}, v, v_{\mathtt{a}}, w)$ never need to be part of an infinite path); for simplicity, we add these equations at the end of the PBES. We denote the updated translation by $\mathbf{E}^c_L$.

Theorem 2. Let formula $\sigma X(d{:}D = e').\ \psi$ be a closed, normalised first-order modal μ-calculus formula and $L(e)$ be an LPE. Then $L(e) \models \sigma X(d{:}D = e').\ \psi$ iff $([\![e]\!],[\![e']\!]) \in [\![\mathbf{E}^c_L(\sigma X(d{:}D = e').\ \psi)]\!](X)$.

Proof. The correctness of the new transformation follows immediately from the fact that we can substitute 'solved' equations for their references, see [13]. That is, by replacing $L^+_{\mathtt{a}}$ by the constant *true* and replacing $L^-_{\mathtt{a}}$ by the constant *false* we can effectively reduce the new rules for translating $[\alpha]\varphi$ and $\langle\alpha\rangle\varphi$ to the ones from Table 3. □

Intuitively, the modification in the translation of the $\langle\alpha\rangle\varphi$ and $[\alpha]\varphi$ construction allows for extracting evidence from a proof graph because whenever the proof graph records information about which predicate variables are required to make φ hold true, also information about the *transition*, encoded by the predicate variable $L^+_{\mathtt{a}}$ must be recorded in the proof graph. This follows from the fact that, *e.g.* in the new translation for $\langle\alpha\rangle\varphi$, the expression $L^+_{\mathtt{a}}$ is added as a conjunct in the new translation. For similar reasons, whenever there is no α-matching transition leading to a state satisfying φ, the proof graph will contain *all* α-matching transitions, encoded by the predicate variable $L^-_{\mathtt{a}}$.

Before we formally define how we can extract diagnostic information from a proof graph, we illustrate the basic idea by showing how to solve the problem we encountered in Example 5.

Example 6. Reconsider the second model checking problem of Example 5, *i.e.* the problem of deciding whether the counter system satisfies property $\nu X.\langle\mathtt{inc}\rangle\langle\mathtt{dec}\rangle X$. Whereas the standard transformation yields a PBES consisting of only one equation, we now obtain a PBES $\mathscr{E}$ with seven equations (two of which are redundant):

$$\begin{array}{lcl}
\Big(\nu X(n{:}N) & = & n+1>0\land \\
& & \Big(\big(((X((n+1)-1)\land L^{+}_{\mathtt{dec}}(n+1,(n+1)-1))\lor L^{-}_{\mathtt{dec}}(n+1,(n+1)-1)) \\
& & \qquad \land L^{+}_{\mathtt{inc}}(n,n+1) \\
& & \quad \big)\lor L^{-}_{\mathtt{inc}}(n,n+1)\Big)\Big) \\
\multicolumn{3}{l}{(\nu L^{+}_{\mathtt{inc}}(n{:}N,n'{:}N) \ = \ \mathit{true})\ (\nu L^{+}_{\mathtt{dec}}(n{:}N,n'{:}N) \ = \ \mathit{true})\ (\nu L^{+}_{\mathtt{reset}}(n{:}N,n'{:}N) \ = \ \mathit{true})} \\
\multicolumn{3}{l}{(\mu L^{-}_{\mathtt{inc}}(n{:}N,n'{:}N) \ = \ \mathit{false})\ (\mu L^{-}_{\mathtt{dec}}(n{:}N,n'{:}N) \ = \ \mathit{false})\ (\mu L^{-}_{\mathtt{reset}}(n{:}N,n'{:}N) \ = \ \mathit{false})}
\end{array}$$

Note that the smaller proof graph of Example 5 we could use to demonstrate that $0\in[\![\mathscr{E}]\!](X)$ now no longer is appropriate since it misses relevant information about the signatures for $L^{+}_{\mathtt{inc}}$ and $L^{+}_{\mathtt{dec}}$. Adding this extra information yields the following proof graph:

$$\begin{tikzcd}
(L^{+}_{\mathtt{dec}},1,0) & (X,0) \arrow[l] \arrow[r] \arrow[loop above] & (L^{+}_{\mathtt{inc}},0,1)
\end{tikzcd}$$

The signatures containing the $L^{+}_{\mathtt{inc}}$ and $L^{+}_{\mathtt{dec}}$ predicate variables encode information about the transitions in the LPE that were involved in constructing the proof that the property we verify holds. Following the basic ideas outlined in [7], we extract the relevant information by filtering the relevant vertices from the proof graph and construct an LPE. In our case, we can extract the following LPE L^{+}:

$$\begin{array}{rcl}
L^{+}(n{:}N) & = & (n=0)\rightarrow \mathtt{inc}\cdot L^{+}(1) \\
& + & (n=1)\rightarrow \mathtt{dec}\cdot L^{+}(0)
\end{array}$$

The LTS induced by $L^{+}(0)$, providing diagnostics at the level of the system, is as follows:

$$\begin{tikzcd}
\boxed{0} \arrow[r, bend left, "\mathtt{inc}"] & \boxed{1} \arrow[l, bend left, "\mathtt{dec}"]
\end{tikzcd}$$

Note that the LTS is a subgraph of the LTS underlying $L(0)$ (see the original LTS of Example 1), which, moreover, provides a compact and intuitive argument why the property holds. Moreover, the proof graph underlying the PBES encoding the same μ-calculus model checking problem on $L^{+}(0)$ is isomorphic to the one we provided above. □

We next formalise the notions of *witness* and *counterexample*.

Definition 6. Let L be an LPE and φ a μ-calculus formula. Let $\mathscr{G}=(V,E)$ be a finite, minimal proof graph proving $L(e)\models\varphi$. Let $W(\mathtt{a})=\{(e_L,e_{\mathtt{a}},e'_L)\mid(L^{+},[\![e_L]\!],[\![e_{\mathtt{a}}]\!],[\![e'_L]\!])\in V\}$. The *witness* extracted from $\mathscr{G}$ is the LPE L_w defined as follows:

$$L_w(d_L{:}D_L)=+\{\sum_{(e_L,e_{\mathtt{a}},e'_L):D_L\times D_{\mathtt{a}}\times D_L}(e_L,e_{\mathtt{a}},e'_L)\in W(\mathtt{a})\rightarrow \mathtt{a}(e_{\mathtt{a}})\cdot L_w(e'_L)\mid \mathtt{a}\in\mathscr{A}ct\,\}$$

In a similar vein, a *counterexample* LPE L_c is obtained by filtering all L_c signatures from V in the refutation graph.

We note that the LTS underlying LPE L_w (and, likewise, the LTS underlying LPE L_c) is a substructure of the LTS underlying L; this follows from minimality of the proof graph from which these LPEs are extracted, plus the fact that the conditions under which the transitions are present are enforced in the translations of $\mathbf{RHS}_L([\alpha]\varphi)$ and $\mathbf{RHS}_L(\langle\alpha\rangle\varphi)$. The theorem below states that the witness extracted from the PBES indeed contains enough information to reconstruct the proof graph from which it was extracted; a dual result holds for counterexamples extracted from a refutation graph.

Theorem 3. Let L be an LPE and φ a μ-calculus formula. Let $\mathcal{G} = (V,E)$ be a finite, minimal proof graph proving $L(e) \models \varphi$, and let L_w be the witness LPE extracted from $\mathcal{G}$. Then any proof graph proving $L_w(e) \models \varphi$ is isomorphic to some proof graph proving $L(e) \models \varphi$.

Proof. This follows essentially from the observation that the LTS underlying the LPE L_w, extracted from $\mathcal{G}$, is a substructure of the LTS underlying L. The bijection that maps all vertices of the form (L_w^+, v) to (L^+, v) and all other vertices to their identities in the proof graph proving $L_w(e) \models \varphi$ then yields an isomorphic proof graph proving $L(e) \models \varphi$; see also Theorem 10 in [7]. $\square$

5 Applications

We have implemented the modified transformation in the mCRL2 toolset in the existing tool `lps2pbes`. Moreover, we have developed a new tool, called `pbessolve`, which extracts evidence from a PBES that is obtained via the new transformation. The latter tool uses *instantiation* to a parity game (much like the technique outlined in [23, 18]), which is then solved using Zielonka's recursive algorithm [24, 10] and from which we extract a witness or counterexample as per Def. 6.

We illustrate the effectiveness of the techniques we outlined in this paper through three examples taken from the mCRL2 repository. The first example is a scheduling problem sometimes referred to as the *bridge-crossing puzzle*. Our second example concerns three communication protocols. The third example we consider is the *Storage Management System* [19] of the DIRAC Community Grid Solution for the LHCb experiment at CERN.

5.1 Bridge Crossing

The bridge-crossing puzzle centres around a scheduling problem with limited access to resources. The problem is essentially as follows. A family of four people (person A, B, C and D), chased by a pack of wolves, needs to cross a narrow bridge at night. The bridge can only hold two persons at a time and the damages to the bridge require the persons to carry a torch to avoid falling off the bridge. Unfortunately, there is only one torch; its battery is running out and can only last another 18 minutes. Persons A and B can cross the bridge in 1, resp. 2 minutes but person C requires 5 minutes and D even requires 10 minutes to cross the bridge. The problem is whether they have a strategy to safely cross the bridge before the battery of the torch runs out.

We model the puzzle as an LPE, where we model the state space by keeping track of the positions of the four persons (*i.e.* which side of the bridge they are), and the time that has passed since switching on the torch. We consider two types of move actions: $\texttt{move}(p_1, l)$, modelling the person p_1 who is carrying a torch to move to location l (which can be s for safe, or d, for dangerous) and $\texttt{move}(p_1, p_2, l)$, modelling the person p_1 who is carrying a torch, crossing the bridge together with p_2. Moreover, action $\texttt{safe}(i)$ signals that the family managed to safely cross the bridge, taking i minutes; action `fail` indicates that the family did not manage to cross the bridge before the battery of the torch ran out. To verify whether the family has a strategy to safely cross the bridge, we verify the following formula:

$$\mu X.(\langle true\rangle X \vee \langle \exists i{:}N.\texttt{safe}(i)\rangle true)$$

The witness proving the formula holds is depicted below, with 0 being the initial state. It shows a schedule by which the family can safely cross the bridge.

$$\begin{tikzcd}
\textcircled{0} \arrow[r, "\texttt{move}(B,A,s)"] & \textcircled{1} \arrow[r, "\texttt{move}(A,d)"] & \textcircled{2} \arrow[r, "\texttt{move}(D,C,s)"] & \textcircled{3} \arrow[r, "\texttt{move}(B,d)"] & \textcircled{4} \arrow[r, "\texttt{move}(B,A,s)"] & \textcircled{5} \arrow[r, "\texttt{safe}(17)"] & \textcircled{6}
\end{tikzcd}$$

5.2 Communication Protocols

Communication protocols allow for exchanging information between devices through networks using strict rules and conventions. In general, the communication medium (*i.e.* the network) may not be perfectly reliable: it may re-order, scramble or even lose data that it transports. Part of the problem solved by a communication protocol is to disassemble and reassemble messages sent via such a network, working around the assumed characteristics of the network to achieve reliable information exchange.

A variety of communication protocols exist; we here consider the classic *Alternating Bit Protocol* (ABP), the *One Bit Sliding Window Protocol*, which is a simple bidirectional sliding window protocol with piggy backing and window sizes as the receiving and sending side of size 1 [1], and the full *Sliding Window Protocol* [9]. The property that we check for all three protocols is the following: there are no paths on which reading of a datum d is enabled infinitely often, but occurs only finitely often. Assuming that the data domain we range over is D and $\texttt{read}(d)$ models reading datum d, this can be formalised as follows (see also, *e.g.* [2]):

$$\forall d{:}D.\nu X.\mu Y.\nu Z.([\texttt{read}(d)]X \wedge ([\texttt{read}(d)]\mathit{false} \vee [\neg\texttt{read}(d)]Y) \wedge [\neg\texttt{read}(d)]Z)$$

None of the three protocols satisfy the property for $|D| > 1$. The counterexamples we can extract all have a similar flavour but differ in the number of actions involved and the underlying reason for violating the property. We depict these counterexamples in Figure 1; we have omitted all other involved actions in these counterexamples.

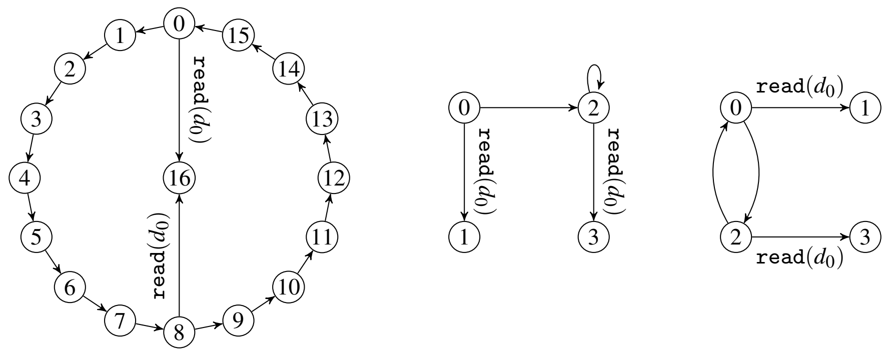


Figure 1: Counterexamples for the *if reading of a datum is infinitely often enabled then it occurs infinitely often* requirement. Left: counterexample for the ABP, middle: counterexample for the One Bit protocol, right: counterexample for the Sliding Window Protocol. In all cases, state 0 indicates the initial state; non-$\texttt{read}(0)$ edge-labels have been omitted from our graphs.

All three counterexamples show the presence of an infinite path along which a $\texttt{read}(d_0)$ is enabled, but never taken. Note that since this is a typical branching-time property with a strong fairness constraint, the counterexample cannot simply be represented by an infinite path represented by a lasso.

5.3 The DIRAC Storage Management System

DIRAC (Distributed Infrastructure with Remote Agent Control) is a grid solution that is designed to support both production activities as well as user data analysis for the Large Hadron Collider 'beauty' experiment. It consists of distributed services that cooperate with light-weight agents that deliver workload to the grid resources: services accept requests from running jobs and agents, whereas agents actively work towards specific goals. The logic of each individual agent is fairly simple but the main source of complexity arises from their cooperation. Agents communicate using the services' databases as a shared memory for synchronising the state transitions. A formal model and analysis of two critical subsystems of DIRAC is described in [19]; one of these is the *Storage Management System* (SMS). In [19], it was shown that this system violated several requirements. Most of these requirements were safety requirements. A violation of such a requirement is a simple trace in the system, which is fairly easy to generate and visualise. A requirement that defied such a simple approach is the following liveness requirement:

$$\begin{aligned}&\nu X.([\mathit{true}]X\wedge\\ &\qquad[\mathtt{state}([\mathit{tStaged}])\vee\mathtt{state}([\mathit{tFailed}])]\nu Y.([\neg\mathtt{state}([\mathit{tDeleted}])]Y\wedge\\ &\qquad\qquad\mu Z.(\langle\mathit{true}\rangle Z\vee\langle\mathtt{state}([\mathit{tDeleted}])\rangle\mathit{true})))\end{aligned}$$

The requirement essentially states that each task that is in a terminating state (*i.e.* in state *tFailed* or *tStaged*) is eventually removed from the DIRAC system (*i.e.* *tDeleted*). The only action of concern is the $\mathtt{state}(s)$ action, which emits the current state s of a task. The counterexample to the property is depicted in Figure 2; the original LTS contains 142 states, which can be further reduced (without affecting the behaviours encoded by the LTS) to the depicted 26 states.

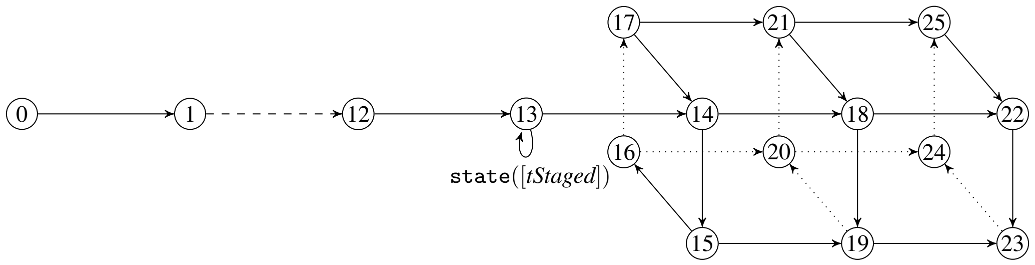


Figure 2: Counterexamples for the requirement that *each task in a terminating state is eventually removed* for the Storage Management Systems. State 0 indicates the initial state; we omitted all edge labels except for the trigger $\mathtt{state}([\mathit{tStaged}])$. The dashed line between state 1 and 12 indicates a single path through states 2, 3,..., 12; the dotted transitions are 3D artefacts.

The counterexample clearly indicates a path towards a part of the system where the task, once staged (in state 13), will never be removed from the system.

6 Conclusions and Future Work

We studied and implemented the extraction of useful evidence from parameterised Boolean equation systems (PBESs) encoding the model checking problem for the first-order modal μ-calculus. Our solution is inspired by the LFP-approach outlined (but not implemented) in [7]. Our solution, which we have also implementation and which is made available through the mCRL2 [5] toolset, shows the appeal of the technique even when used for explaining the failure for complex requirements to hold.

Apart from the model checking problem, PBESs can also be used for checking behavioural equivalence of processes [4]. Diagnostics for such problems are typically presented in the form of a game [8]. We expect to be able to apply the ideas outlined in the current paper to such problems as well, leading to evidence in the form of substructures for *both* input models, which, combined, explain the differences between both models. Implementing this problem and investigating whether such a solution would provide intelligible feedback to the user is left for future work.

References

[1] M. Bezem and J.F. Groote. A correctness proof of a one-bit sliding window protocol in μCRL. *Comput. J.*, 37(4):289–307, 1994.

[2] J. Bradfield and C. Stirling. Modal logics and mu-calculi. In *Handbook of Process Algebra*, pages 293–332. Elsevier, North-Holland, 2001.

[3] S. Busard. *Symbolic Model Checking of Multi-Modal Logics: Uniform Strategies and Rich Explanations*. PhD thesis, Université catholique de Louvain (UCL), 2017.

[4] T. Chen, B. Ploeger, J. van de Pol, and T.A.C. Willemse. Equivalence checking for infinite systems using parameterized Boolean equation systems. In *CONCUR*, volume 4703 of *LNCS*, pages 120–135. Springer, 2007.

[5] S. Cranen, J.F. Groote, J.J.A. Keiren, F.P.M. Stappers, E.P. de Vink, W. Wesselink, and T.A.C. Willemse. An overview of the mCRL2 toolset and its recent advances. In *TACAS*, volume 7795 of *LNCS*, pages 199–213. Springer, 2013.

[6] S. Cranen, B. Luttik, and T.A.C. Willemse. Proof graphs for parameterised Boolean equation systems. In *CONCUR*, volume 8052 of *LNCS*, pages 470–484. Springer, 2013.

[7] S. Cranen, B. Luttik, and T.A.C. Willemse. Evidence for fixpoint logic. In *CSL*, volume 41 of *LIPIcs*, pages 78–93. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[8] D. de Frutos-Escrig, J.J.A. Keiren, and T.A.C. Willemse. Games for bisimulations and abstraction. *Logical Methods in Computer Science*, 13(4), 2017.

[9] W. Fokkink, J.F. Groote, J. Pang, B. Badban, and J. van de Pol. Verifying a sliding window protocol in μCRL. In *AMAST*, volume 3116 of *LNCS*, pages 148–163. Springer, 2004.

[10] M. Gazda and T.A.C. Willemse. Zielonka's recursive algorithm: dull, weak and solitaire games and tighter bounds. In *GandALF*, volume 119 of *EPTCS*, pages 7–20, 2013.

[11] J.F. Groote and M.R. Mousavi. *Modeling and Analysis of Communicating Systems*. MIT Press, 2014.

[12] J.F. Groote and T.A.C. Willemse. Model-checking processes with data. *Sci. Comput. Program.*, 56(3):251–273, 2005.

[13] J.F. Groote and T.A.C. Willemse. Parameterised Boolean equation systems. *Theor. Comput. Sci.*, 343(3):332–369, 2005.

[14] A. Kick. Tableaux and witnesses for the μ-calculus, 1995. Tech. Rep. 44/95, University of Karlsruhe.

[15] A. Mader. *Verification of Modal Properties Using Boolean Equation Systems*. PhD thesis, Technische Universität München, 1997.

[16] R. Mateescu. *Vérification des propriétés temporelles des programmes parallèles*. PhD thesis, Institut National Polytechnique de Grenoble, 1998.

[17] R. Mateescu. Efficient diagnostic generation for Boolean equation systems. In *TACAS*, volume 1785 of *LNCS*, pages 251–265. Springer, 2000.

[18] B. Ploeger, W. Wesselink, and T.A.C. Willemse. Verification of reactive systems via instantiation of parameterised Boolean equation systems. *Inf. Comput.*, 209(4):637–663, 2011.

[19] D. Remenska, T.A.C. Willemse, K. Verstoep, J. Templon, and H.E. Bal. Using model checking to analyze the system behavior of the LHC production grid. *Future Generation Comp. Syst.*, 29(8):2239–2251, 2013.

[20] P. Stevens and C. Stirling. Practical model-checking using games. In *TACAS*, volume 1384 of *LNCS*, pages 85–101. Springer, 1998.

[21] L. Tan and R. Cleaveland. Evidence-based model checking. In *CAV*, volume 2404 of *LNCS*, pages 455–470. Springer, 2002.

[22] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pac. J. Math.*, 5(2):285–309, June 1955.

[23] A. van Dam, B. Ploeger, and T.A.C. Willemse. Instantiation for parameterised Boolean equation systems. In *ICTAC*, volume 5160 of *LNCS*, pages 440–454. Springer, 2008.

[24] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998.