Static Analysis Techniques for Parameterised Boolean Equation Systems*

Simona Orzan, Wieger Wesselink, and Tim A.C. Willemse

Eindhoven University of Technology, The Netherlands

Abstract. Parameterised Boolean Equation Systems (PBESs) can be used to encode and solve various types of model checking and equivalence checking problems. PBESs are typically solved by symbolic approximation or by instantiation to Boolean Equation Systems (BESs). The latter technique suffers from something similar to the state space explosion problem and we propose to tackle it by static analysis techniques, which we tailor for PBESs. We introduce a method to eliminate redundant parameters and a method to detect constant parameters. Both lead to a better performance of the instantiation and they can sometimes even reduce problems that are intractable due to the infinity of the underlying BES to tractable ones.

1 Introduction

Model checking and equivalence checking techniques are very sensitive to the size of the state space. A *static analysis* can be used to reduce the state space size; most often, it employs some form of flow analysis to detect what values a given subexpression of a process description can possibly evaluate to at run-time [9], and this information can subsequently be used to achieve state space reductions. A further minimisation might be obtained if the analysis is tailored to the properties to be verified. This constitutes a major challenge, as it requires analysing both the verification question and the specification. One can avoid this by encoding the verification problem in a single high-level formalism; *Parameterised Boolean Equation Systems* [11,7] allow to do just that.

Parameterised Boolean Equation Systems (PBESs) have emerged as a versatile framework for studying and solving verification problems. Prime examples are the PBES encoding of the first-order modal μ-calculus model checking problem over (possibly infinite) labelled transition systems [11,7] and equivalence checking of various bisimulations on (possibly infinite) labelled transition systems [1]. Intuitively, the PBES encoding of a given verification problem only requires the aspects of the specification that influence the property that has to be verified.

Problems encoded in the PBES framework can be solved by computing the solution to the respective PBES. Even though the latter is an undecidable problem, a number of techniques have been developed to obtain solutions in practice, including *symbolic approximation* [7], *pattern matching* [8], *invariant* techniques [13] and *instantiation* [11,12,4].

* This research has been partially funded by the Netherlands Organisation for Scientific Research (NWO) under FOCUS/BRICKS grant number 642.000.602.

In this paper, we are concerned with the latter, i.e. instantiation of PBESs to Boolean Equation Systems (BES); BESs constitute a decidable fragment of PBESs.

We develop two methods, based on static analysis, that allow to automatically reduce the complexity of a PBES. These methods take inspiration from [6], where comparable methods are applied to a symbolic state space description. We first investigate, and prove the correctness of a method that allows to eliminate a class of redundant data parameters from a PBES; second, we develop an algorithm, and prove its correctness, that computes a special type of PBES invariant [13], which can subsequently be used to eliminate data parameters and simplify the right-hand sides of a PBES.

The practical significance of the two complexity reduction methods is assessed by means of a set of experiments derived from typical model checking problems. These demonstrate dramatic improvements in the time needed to compute a BES from a PBES, and the size of these BESs; some intractable problems even reduce to tractable ones.

A third contribution of our paper is the introduction of a *normal form* for PBESs. The existence of the normal form has both theoretical and practical implications. On the one hand, it allows for more concise definitions and a uniform presentation of manipulation methods, and, on the other hand, it will help to find, e.g., new patterns [8]. Apart from using the normal form in our definitions of the two mentioned complexity reduction methods, we also use it to simplify the characterisation of invariants for PBESs, paving the way for automating the detection and checking of complex invariants.

Related Work. As mentioned, we take inspiration from [6], where similar static analysis techniques are applied to reduce state spaces. Other forms of static analysis techniques include abstract interpretation, initiated in [3] and used in, e.g. [10,14], influence analysis in programming languages [5], and the so-called *cone of influence reduction* (also known as *slicing* or *localisation reduction*) technique that reduces the size of the state space for synchronous circuits in specific (see [2]) and systems in general (see [16]). Compared to these works, our methods deal with a more advanced setting and have the potential to immediately (and soundly) reduce the complexity of (encoded) verification problems. As such, we can solve verification questions that cannot readily be answered by only reducing the state space (see e.g. our Example 1).

Outline. In Section 2 the basic PBES theory is repeated. Section 3 describes a normal form for PBESs and its implications for invariants. The two complexity reduction methods are described in Section 4, and an analysis of the impact of these algorithms on typical model checking problems can be found in Section 5. Section 6 summarises the main results of this paper and discusses future work.

2 Preliminaries

2.1 Data

We work in the setting of *abstract data types*, i.e., we assume that there are nonempty data sorts and operations on these sorts. We typically use letters $D_1, D_2, \ldots$ to denote data sorts. Furthermore, we assume to have a set $\mathcal{D}$ of *sorted data variables*, with typical elements $d, d_1, \ldots$, *etcetera*. We write $\boldsymbol{d}$, which stands for a vector of variables $(d_1, \ldots, d_n)$; this notation extends to vectors of terms $\boldsymbol{e}$, vectors of sorts $\boldsymbol{D}$ and vectors of values $\boldsymbol{v}$ in the semantic domain. A vector of sort declarations $\boldsymbol{d}{:}\boldsymbol{D}$ should be read as $(d_1{:}D_1, \ldots, d_n{:}D_n)$. The i-th element of $\boldsymbol{d}$ is denoted $\boldsymbol{d}[i]$.

With every syntactic sort D we associate a semantic set $\mathbb{D}$ such that every syntactic term of type D and all the operations on the sort D can be mapped to the elements and operations of $\mathbb{D}$ they represent. For the interpretation of closed data terms, we assume an interpretation function $[\![\_]\!]$ that maps every term t of sort D to the data element $[\![t]\!]$ of $\mathbb{D}$ it represents. For open terms we use an *environment* ε that maps each variable from $\mathcal{D}$ to a data element of the associated type. The interpretation $[\![t]\!]\,\varepsilon$ of an open term t is given by $\varepsilon(t)$, where ε is extended to terms in the standard way.

For arbitrary environment θ, we write $\theta[v/d]$ to represent the environment that is defined as $(\theta[v/d])(d') = \theta(d')$ for $d \neq d'$ and $(\theta[v/d])(d) = v$. For substitution on vectors we define $\theta[\boldsymbol{v}/\boldsymbol{d}]$ to be equivalent to the simultaneous substitution $\theta[\boldsymbol{v}[1]/\boldsymbol{d}[1], \ldots, \boldsymbol{v}[n]/\boldsymbol{d}[n]]$.

For convenience, we assume the existence of a sort $B = \{\top, \bot\}$ representing the Booleans $\mathbb{B}$. For this sort, 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, we have $\mathbb{B} = \{\top, \bot\}$ and the syntactic operator $\_ \wedge \_{:}B \times B \to B$ corresponds to the usual, semantic conjunction $\_ \wedge \_{:}\mathbb{B} \times \mathbb{B} \to \mathbb{B}$.

2.2 Parameterised Boolean Equation Systems

Parameterised Boolean Equation Systems (PBESs, or *equation systems* for short) are empty (denoted ϵ) or finite sequences of fixed point equations, where each equation is of form $\big(\mu X(\boldsymbol{d}{:}\boldsymbol{D}) = \phi\big)$ or $\big(\nu X(\boldsymbol{d}{:}\boldsymbol{D}) = \phi\big)$. The left-hand side of each equation consists of a *fixed point symbol*, where μ indicates a least and ν a greatest fixed point, and a *sorted predicate variable* X of sort $\boldsymbol{D} \to B$, taken from some countable domain of sorted predicate variables $\mathcal{X}$. The right-hand side of each equation is a *predicate formula* as defined below.

Definition 1. Predicate formulae ϕ *are defined by the following grammar:*

$$\phi ::= b \mid X(\boldsymbol{e}) \mid \phi \oplus \phi \mid \mathsf{Q}d{:}D.\ \phi$$

where $\oplus \in \{\wedge, \vee\}$*,* $\mathsf{Q} \in \{\forall, \exists\}$*, b is a data term of sort B, X is a predicate variable, d is a data variable of sort D and $\mathbf{e}$ is a vector of data terms. The interpretation of ϕ in the context of environments η for predicate variables and ε for data variables is denoted $[\![\phi]\!]\,\eta\varepsilon$, where:*

$$\begin{array}{llll}
[\![b]\!]\,\eta\varepsilon & =_{def} \varepsilon(b) & [\![\phi_1 \oplus \phi_2]\!]\,\eta\varepsilon & =_{def} [\![\phi_1]\!]\,\eta\varepsilon \oplus [\![\phi_2]\!]\,\eta\varepsilon \\
[\![X(\boldsymbol{e})]\!]\,\eta\varepsilon & =_{def} \eta(X)(\varepsilon(\boldsymbol{e})) & [\![\mathsf{Q}d{:}D.\ \phi]\!]\,\eta\varepsilon & =_{def} \mathsf{Q}v \in \mathbb{D}.\ [\![\phi]\!]\,\eta\varepsilon[v/d]
\end{array}$$

We denote the freely occurring data variables in a formula ϕ by $\mathcal{FV}(\phi)$. In line with [13], predicate formulae that do not contain predicate variables are called *simple* predicate formulae; Pred is the set of simple formulae. A simple predicate formula ϕ satisfies the property $[\![\phi]\!]\,\eta\varepsilon = [\![\phi]\!]\,\eta'\varepsilon$, for arbitrary η, η'. As a convention, we denote simple predicate formulae using letters g, h, *etcetera*. Observe that negation does not occur in predicate formulae, except as an operator in Boolean terms. We frequently write $h \implies \phi$ instead of $\neg h \vee \phi$.

The set of predicate variables that occur in a predicate formula ϕ, denoted by $\mathsf{occ}(\phi)$, is defined recursively as follows, for any formulae ϕ_1, ϕ_2:

$$\begin{array}{llll} \mathsf{occ}(b) & =_{def} \emptyset & \mathsf{occ}(X(\boldsymbol{e})) & =_{def} \{X\} \\ \mathsf{occ}(\phi_1 \oplus \phi_2) & =_{def} \mathsf{occ}(\phi_1) \cup \mathsf{occ}(\phi_2) & \mathsf{occ}(\mathsf{Q}d{:}D.\ \phi_1) & =_{def} \mathsf{occ}(\phi_1). \end{array}$$

Extended to equation systems, $\mathsf{occ}(\mathcal{E})$ is the union of all variables occurring at the right-hand side of equations in $\mathcal{E}$. For any equation system $\mathcal{E}$, the set of *binding predicate variables*, $\mathsf{bnd}(\mathcal{E})$, is the set of variables occurring at the left-hand side of some equation in $\mathcal{E}$. Formally, we define:

$$\begin{array}{ll} \mathsf{bnd}(\epsilon) =_{def} \emptyset & \mathsf{bnd}((\sigma X(\boldsymbol{d}{:}\boldsymbol{D}) = \phi)\ \mathcal{E}) =_{def} \mathsf{bnd}(\mathcal{E}) \cup \{X\} \\ \mathsf{occ}(\epsilon) =_{def} \emptyset & \mathsf{occ}((\sigma X(\boldsymbol{d}{:}\boldsymbol{D}) = \phi)\ \mathcal{E}) =_{def} \mathsf{occ}(\mathcal{E}) \cup \mathsf{occ}(\phi). \end{array}$$

The set of freely occurring predicate variables in $\mathcal{E}$, denoted $\mathsf{free}(\mathcal{E})$ is defined as $\mathsf{occ}(\mathcal{E}) \setminus \mathsf{bnd}(\mathcal{E})$. An equation system $\mathcal{E}$ is said to be *well-formed* iff every binding predicate variable occurs at the left-hand side of precisely one equation of $\mathcal{E}$. We only consider well-formed equation systems in this paper.

An equation system $\mathcal{E}$ is called *closed* if $\mathsf{free}(\mathcal{E}) = \emptyset$ and *open* otherwise. An equation $(\sigma X(\boldsymbol{d}{:}\boldsymbol{D}) = \phi)$, where σ denotes either the fixed point sign μ or ν, is called *data-closed* if the set of data variables that occur freely in ϕ is contained in the set of variables induced by the vector of variables $\boldsymbol{d}$. An equation system is called *data-closed* iff each of its equations is data-closed.

An equation $(\sigma X(\boldsymbol{d}{:}\boldsymbol{D}) = \phi)$ gives rise to a fixed point over the set of functions with domain $\mathbb{D}$ and co-domain $\mathbb{B}$. We introduce the notation $\phi_{\langle \boldsymbol{d} \rangle}$, which lifts the predicate formula ϕ to the (syntactic) functional $(\lambda \boldsymbol{d}{:}\boldsymbol{D}.\ \phi)$. The interpretation of $\phi_{\langle \boldsymbol{d} \rangle}$, denoted $[\![\phi_{\langle \boldsymbol{d} \rangle}]\!]\eta\varepsilon$, is given by the functional $(\lambda \boldsymbol{v}{\in}\mathbb{D}.\ [\![\phi]\!]\eta\varepsilon[\boldsymbol{v}/\boldsymbol{d}])$. The set of (total) functions $f{:}\mathbb{D} \to \mathbb{B}$, denoted by $\mathbb{B}^{\mathbb{D}}$, equipped with the point-wise ordering $\sqsubseteq$ leads to a complete lattice. Assuming that the domain of the predicate variable X is of sort $\boldsymbol{D}$, the functional $[\![\phi_{\langle \boldsymbol{d} \rangle}]\!]\eta\varepsilon$ yields the monotone predicate formula transformer $\lambda g{\in}\mathbb{B}^{\mathbb{D}}.\ ([\![\phi_{\langle \boldsymbol{d} \rangle}]\!]\eta[g/X]\varepsilon)$. The existence of the least and greatest fixed points of such transformers is guaranteed by Tarski's fixed point Theorem [15].

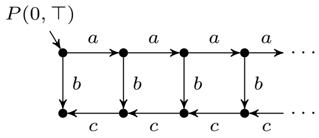
Definition 2. *The* solution of an equation system *in the context of a predicate environment η and a data environment ε is inductively defined as follows, for any $\mathcal{E}$:*

$$\begin{array}{ll} [\![\epsilon]\!]\eta\varepsilon & =_{def}\ \eta \\ [\![(\sigma X(\boldsymbol{d}{:}\boldsymbol{D}) = \phi)\mathcal{E}]\!]\eta\varepsilon & =_{def}\ [\![\mathcal{E}]\!](\eta[\sigma f \in \mathbb{B}^{\mathbb{D}}.\ [\![\phi_{\langle \boldsymbol{d} \rangle}]\!]([\![\mathcal{E}]\!]\eta[f/X]\varepsilon)\varepsilon/X])\varepsilon. \end{array}$$

The solution of an equation system prioritises the fixed point signs of equations that come first over the signs of equations that follow, while respecting the equivalences of the equations. It follows that the solution is sensitive to the order of equations in an equation system. We illustrate the use of equation systems by means of an academic example using the encoding of [7] of the first-order modal μ-calculus model checking problem.

Example 1. Consider an infinite-state process that can perform an arbitrary number of a actions, then performs a b action and then performs as many c actions as a actions that were performed. A partial visualisation of this process, and a process algebraic description using condition-action-effect rules is given below:

$$\begin{aligned} & P(n{:}N, d{:}B) \\ & = d \longrightarrow a \cdot P(n+1, d) \\ & + d \longrightarrow b \cdot P(n, \neg d) \\ & + \neg d \wedge n > 0 \longrightarrow c \cdot P(n-1, d) \end{aligned}$$



Given this process, we might wish to verify whether it is possible to perform an infinite number of a actions ($\nu X. \langle a \rangle X$), or, whether along every a path, always a b action is attainable ($\nu V.\ ([a]V \wedge \mu W.\ (\langle a \rangle W \vee \langle b \rangle \top))$). The first verification problem is encoded by (1); the second by (2), given below:

$$\big(\nu X(n{:}N, d{:}B) = d \wedge X(n+1, d)\big) \tag{1}$$

$$\begin{aligned} &\big(\nu V(n{:}N, d{:}B) = \ (d \implies V(n+1, d)) \wedge W(n, d)\big) \\ &\big(\mu W(n{:}N, d{:}B) = d \vee (d \wedge W(n+1, d))\big) \end{aligned} \tag{2}$$

Currently, instantiation of the above equation systems leads to two infinite BESs. We will use Eqn. (2) as a running example in Section 4. □

3 Predicate Formula Normal Form

Manipulations and comparisons of formal objects typically benefit from the use (and existence) of a normal form for such objects. For this reason, we introduce a normal form for predicate formulae, which immediately implies a normal form for equation systems. Throughout this paper, we will then assume equation systems in normal form.

Definition 3. *A predicate formula is said to be in* Predicate Formula Normal Form *(PFNF) if it has the following form:*

$$\mathsf{Q}_1 v_1{:}V_1. \cdots \mathsf{Q}_n v_n{:}V_n.\ h \wedge \bigwedge_{i \in I} \Big(g_i \implies \bigvee_{j \in J_i} X^j(\boldsymbol{e^j})\Big)$$

where $X^j \in \mathcal{X}$, $\mathsf{Q}_i \in \{\forall, \exists\}$, I is a (possibly empty) finite index set, each J_i is a non-empty finite index set, and h and every g_i are simple formulae.

Note that here J_i is used to index a set of occurrences of not necessarily different variables. For instance, $(n > 0 \implies X(3) \vee X(5) \vee Y(6))$ is a formula complying to the definition of PFNF. As long as it does not lead to confusion, we stick to the convention to drop the typing of the quantified variables v_i.

Proposition 1. *Every predicate formula can be rewritten to an equivalent predicate formula in PFNF.*

The proof is constructive, by means of a structural induction; this immediately provides the basis for a normalisation algorithm. We should remark that transforming the disjunction of two PFNF formulae into a PFNF formula leads to an undesirable blow-up of the formula size. However, when used, as here, in the context of equation systems, this blow-up can be reduced to a linear blow-up by introducing new equations.

A static invariance check. PBES invariants [13] are simple predicates characterising a closed set of attainable values of the data parameters in an equation system. They are very useful for simplifying and solving complex equation systems, as demonstrated in several case studies [13,8]. However, finding the right invariants is not easy, partly because the invariance condition quantifies over arbitrary predicate variable environments. Using PFNF, however, the invariance condition of [13] can be recast to one that can be checked statically. For completeness' sake, we first repeat the definition of an invariant.

Definition 4. *The simple function $f{:}V \to \mathsf{Pred}$ is said to be a* global invariant *for an equation system $\mathcal{E}$ iff $\mathcal{X} \supseteq V \supseteq \mathsf{bnd}(\mathcal{E})$ and for each $(\sigma X(\boldsymbol{d}_X{:}\boldsymbol{D}_X) = \phi)$ occurring in $\mathcal{E}$, we have:*

$$f(X) \wedge \phi \;\leftrightarrow\; (f(X) \wedge \phi)\left[_{X_i \in V} (f(X_i) \wedge X_i(\boldsymbol{d}_{X_i}))_{\langle \boldsymbol{d}_{X_i}\rangle}/X_i\right].$$

Note that $\psi\left[_{X_i \in V}\ \phi(X_i)/X_i\right]$ stands for a simultaneous syntactic subtitution of $\phi(X_i)$ for every X_i from V in ψ. The invariance condition basically states that the right-hand sides of all equations should be insensitive to strengthening all predicate variable occurrences with their corresponding simple formula. The following theorem provides an easy to check criterion implying the invariance condition.

Theorem 1. *Let $\mathcal{E}$ be an equation system where every equation k is in PFNF:*

$$\Big(\sigma_k X_k(\boldsymbol{d}_{X_k}{:}\boldsymbol{D}_k) = \mathsf{Q}^k_1 v_1.\cdots \mathsf{Q}^k_{n_k} v_{n_k}.\ (h^k \wedge \bigwedge_{i \in I_k} (g^k_i \implies \bigvee_{j \in J_i} X^j(\boldsymbol{e}^j)))\Big).$$

Then the simple function $f{:}V \to \mathsf{Pred}$ is a global invariant for $\mathcal{E}$ if for each k:

$$\bigwedge_{i \in I_k} \bigwedge_{j \in J_i} \Big((f(X_k) \wedge h^k \wedge g^k_i) \to f(X^j)[\boldsymbol{e}^j/\boldsymbol{d}_{X^j}]\Big) \qquad (\iota_k)$$

□

The proof is a tedious and semantically rather involved exercise leading to f satisfying Definition 4 under all data and predicate environments. It makes essential use of the fact that condition (ι_k) implicitly converts all quantifiers $\mathsf{Q}^k_1 \ldots \mathsf{Q}^k_{n_k}$ into universal quantifiers. Note that (ι_k) is not a necessary condition for f to be an invariant, as demonstrated by the following example which makes use of the fact that all existential quantifiers are converted to universal quantifiers.

Example 2. Consider the equation system $\mathcal{E}$ given below:

$$\mathcal{E} :\equiv \Big(\mu X(n{:}N) = \exists m{:}N.\ (m > 5 \implies Y(n))\Big)\ \Big(\nu Y(n{:}N) = Y(n+1)\Big).$$

Let us define the simple function f as $f(X) = \top, f(Y) = \bot$. Condition (ι_k) in this case requires $\forall m{:}N.\ (\top \wedge m > 5 \implies \bot)$, which does not hold. Still, f is an invariant for $\mathcal{E}$, since it satisfies the invariant condition: $(\top \wedge \exists m{:}N.\ (m > 5 \implies Y(n))) \leftrightarrow (\top \wedge \exists m{:}N.\ (m > 5 \implies (\bot \wedge Y(n))))$. □

It can be proven (not trivially) that the condition above is actually necessary in the case of quantifier-free equation systems, but we pose it as an interesting open problem whether condition (ι_k) can be modified (or some other condition can be thought up) to serve as a sufficient and necessary condition without employing a quantification over predicate environments.

4 Redundant and Constant Parameter Detection and Elimination

A part of the complexity of an equation system stems from the arity of the involved predicate variables and the types of these. Reducing an equation system's complexity can thus be achieved by minimising the complexity of the formal data parameters, either by removing them or by (implicitly) reducing their types. However, such operations are not always sound. In Sections 4.1 and 4.2, we develop algorithms that achieve these types of reductions without compromising soundness.

4.1 Parameter Elimination

The type of a predicate variable X is said to contain *redundancy* with respect to an environment if it relies on one or more values that do not manifest themselves in the truth of X using this environment.

Definition 5. *Given an environment η and a predicate variable X with signature $D_1 \times \cdots \times D_n \to B$, then a sort D_i ($1 \leq i \leq n$) is redundant with respect to η if for all values $v, w \in \mathbb{D}_i$, and $v_j \in \mathbb{D}_j$ for $j \neq i$, we have*

$$\eta(X)(v_1, \ldots, v, \ldots, v_n) = \eta(X)(v_1, \ldots, w, \ldots, v_n)$$

A semantic analysis of redundancy is neither feasible, nor desirable for most complex equation systems, so the best that can be achieved is to approximate the set of redundant sorts. We start by formalising the concept of influence.

Definition 6. *Let ρ be a predicate function in PFNF:*

$$\mathsf{Q}_1 v_1. \cdots \mathsf{Q}_n v_n.\, h \wedge \bigwedge_{i \in I} \Big(g_i \implies \bigvee_{j \in J_i} X^j(\boldsymbol{e}^j)\Big)$$

We define the dependence set $\mathsf{dep}(\rho)$ *and the significance set* $\mathsf{sig}(\rho)$ *as follows:*

1. $\mathsf{dep}(\rho) =_{\mathit{def}} \bigcup_{i \in I} \{X^j(\boldsymbol{e}^j) \mid j \in J_i\}$
2. $\mathsf{sig}(\rho) =_{\mathit{def}} \bigcup_{i \in I} \mathcal{FV}(\mathsf{Q}_1 v_1. \cdots \mathsf{Q}_n v_n.\, h \wedge g_i)$

The influence of a set of predicate functions on predicate variables is modelled by means of an influence graph. Recall that the i-th element of a vector $\boldsymbol{d}$ is denoted $\boldsymbol{d}[i]$.

Definition 7. *Let $\mathcal{E} = (\sigma_1 X_1(\boldsymbol{d}_{X_1}{:}\boldsymbol{D}_{X_1}) = \phi_{X_1}) \cdots (\sigma_n X_n(\boldsymbol{d}_{X_n}{:}\boldsymbol{D}_{X_n}) = \phi_{X_n})$ be an equation system. The* marked influence graph *$G(\mathcal{E}) = (V, \longrightarrow, M)$ of $\mathcal{E}$ is a directed graph where:*

- *$V = \{(X_i, j) \mid 1 \leq i \leq n$ and $1 \leq j \leq \mathsf{arity}(\boldsymbol{D}_{X_i})\}$;*
- *$\longrightarrow \subseteq V \times V$ is the transition relation, defined by*

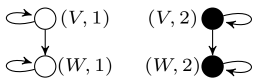
$$(X_i, k) \longrightarrow (X_j, l) \text{ iff } X_j(\boldsymbol{e}) \in \mathsf{dep}(\phi_{X_i}) \text{ and } \boldsymbol{d}_{X_i}[k] \in \mathcal{FV}(\boldsymbol{e}[l])$$

- *$M \subseteq V$ is the* initial marking, *defined by*

$$M = \{(X_i, j) \mid 1 \leq i \leq n \text{ and } \boldsymbol{d}_{X_i}[j] \in \mathsf{sig}(\phi_{X_i})\}$$

Intuitively, in a marked influence graph $G(\mathcal{E})$, the initial marking M is the set of variables that influences the truth of the simple formulae that occur in the predicate formulae of the equation system $\mathcal{E}$. The transition relation $\longrightarrow$ formalises the direct and indirect influence that formal parameters can have on the value of other formal parameters.

Example 3. Consider the equation system from Eqn. (2) (Example 1), first brought into PFNF. The marked influence graph is depicted on the right, where the marked states are black and non-marked states are white.



□

Next, we define the set of positive redundant variables as follows:

$$\mathcal{R} = \{\boldsymbol{d}_{X_i}[j] \mid (X_i, j) \not\longrightarrow^* (X_k, l) \text{ and } (X_k, l) \in M\} \tag{3}$$

Computing the set $\mathcal{R}$ requires $\mathcal{O}(|\longrightarrow|)$ steps at most using, e.g., a standard least fixed point computation, depth-first or breadth-first search. We refrain from spelling out such an algorithm.

Definition 8. *Let* $\mathcal{E} = (\sigma_1 X_1(\boldsymbol{d}_{X_1}{:}\boldsymbol{D}_{X_1}) = \phi_{X_1}) \cdots (\sigma_n X_n(\boldsymbol{d}_{X_n}{:}\boldsymbol{D}_{X_n}) = \phi_{X_n})$*, where each* ϕ_{X_k} *is of the form:*

$$\mathsf{Q}^k_1 v_1. \cdots \mathsf{Q}^k_{n_k} v_{n_k}.\ h^k \wedge \bigwedge_{i \in I^k} \Big(g^k_i \implies \bigvee_{j \in J^k_i} X^j(\boldsymbol{e}^j)\Big)$$

The reduction *of* $\mathcal{E}$*, denoted* $\widehat{\mathcal{E}}$ *is the equation system*

$$(\sigma_1 \widehat{X_1}(\widehat{\boldsymbol{d}_{X_1}}{:}\widehat{\boldsymbol{D}_{X_1}}) = \widehat{\phi_{X_1}}) \cdots (\sigma_n \widehat{X_n}(\widehat{\boldsymbol{d}_{X_n}}{:}\widehat{\boldsymbol{D}_{X_n}}) = \widehat{\phi_{X_n}})$$

where for every k *(*$1 \leq k \leq n$*), we define the following:*

1. $\widehat{\boldsymbol{d}_{X_k}}$ *is the vector* $\boldsymbol{d}_{X_k}$ *from which the parameters* $\boldsymbol{d}_{X_k}[i] \in \mathcal{R}$ *have been removed;*
2. $\widehat{\boldsymbol{D}_{X_k}}$ *is the vector* $\boldsymbol{D}_{X_k}$ *from which the types of* $\boldsymbol{d}_{X_k}[i]$ *have been removed;*
3. $\widehat{\phi_{X_k}} :\equiv \mathsf{Q}^k_1 v_1. \cdots \mathsf{Q}^k_{n_k} v_{n_k}.\ h^k \wedge \bigwedge_{i \in I^k} \Big(g^k_i \implies \bigvee_{j \in J^k_i} \widehat{X^j}(\widehat{\boldsymbol{e}^j})\Big)$*, where* $\widehat{\boldsymbol{e}^j}$ *is the vector* $\boldsymbol{e}^j$ *from which expressions* $\boldsymbol{e}^j[i]$ *with* $\boldsymbol{d}_{X^j}[i] \in \mathcal{R}$ *have been removed.*

The reduction of an equation system basically consists of a syntactic manipulation of predicate variable typings and predicate variable occurrences. It introduces a new set of predicate variables that are linked to the original predicate variables in the equation system. The typing of these newly introduced predicate variables is of lesser complexity than the typing of the original predicate variables. In particular, if the original equation system contains a predicate variable X_i of type $\mathbb{D}_{X_i}$, then the associated predicate variable $\widehat{X_i}$ is of type $\widehat{\mathbb{D}_{X_i}}$, which is based on $\widehat{\boldsymbol{D}_{X_i}}$. For elements $\boldsymbol{w}$ of type $\mathbb{D}_{X_i}$, we denote the corresponding reduced element by $\widehat{\boldsymbol{w}}$. We have the following two properties:

Lemma 1. *If* $\mathcal{E}$ *is data-closed, then so is* $\widehat{\mathcal{E}}$*.*

Proof. From the observation that $\widehat{\mathcal{E}}$ can only contain a free data variable if this is a formal parameter $\boldsymbol{d}_{X_k}[j]$ for some equation for $\widehat{X_k}$ that does not occur in the parameter list of this equation. This leads to a contradiction, based on the definition of $\mathcal{R}$ and $\widehat{\mathcal{E}}$. □

Lemma 2. *Let $\mathcal{E}$ be a data-closed equation system. Let $(\sigma_k X_k(\boldsymbol{d}_{X_k}{:}\boldsymbol{D}_{X_k}) = \phi_{X_k})$ be an arbitrary equation in $\mathcal{E}$. Let η be an environment for which $\eta(X)(\boldsymbol{v}) = \eta(\widehat{X})(\widehat{\boldsymbol{v}})$ for all $\boldsymbol{v}$ and $X \in \mathsf{occ}(\phi_{X_k})$. Then:*

$$\forall \varepsilon : \ [\![\phi_k]\!]\eta\varepsilon = [\![\widehat{\phi_k}]\!]\eta\varepsilon$$

Proof. Without loss of generality, ϕ_{X_k} is in PFNF. The proof then follows from a repeated application of the semantics. □

The following theorem demonstrates that the elimination of positively redundant formal parameters from an equation system does not affect the solution of the equation system.

Theorem 2. *Let $\mathcal{E}$ be a data-closed equation system. Let η, ε be arbitrary environments. If for all $X \in \mathsf{free}(\mathcal{E})$ we have $\eta(X)(\boldsymbol{v}) = \eta(\widehat{X})(\widehat{\boldsymbol{v}})$ for all $\boldsymbol{v}$, then for all $X_k \in \mathsf{bnd}(\mathcal{E})$ and all $\boldsymbol{v}$:*

$$[\![\mathcal{E}]\!]\eta\varepsilon(X_k(\boldsymbol{v})) = [\![\widehat{\mathcal{E}}]\!]\eta\varepsilon(\widehat{X_k}(\widehat{\boldsymbol{v}}))$$

Proof. By means of an induction on $|\mathcal{E}|$. The base case follows immediately. The induction requires a case distinction, a transfinite approximation and Lemmas 1 and 2. □

Corollary 1. *In case $\mathcal{E}$ is data-closed* and *closed, we obtain the following result:*

$$\forall \eta, \varepsilon : \ \forall X_l \in \mathsf{bnd}(\mathcal{E}) : \ [\![\mathcal{E}]\!]\eta\varepsilon(X_l(\boldsymbol{v})) = [\![\widehat{\mathcal{E}}]\!]\eta\varepsilon(\widehat{X_l}(\widehat{\boldsymbol{v}})))$$

Example 3. Consider the equation system from Eqn. (2) from Example 1, brought into PFNF, and name it $\mathcal{E}$. From its marked influence graph, we find $\mathcal{R} = \{(V,2),(W,2)\}$. This means that equation system (2) can be reduced to the equation system $\widehat{\mathcal{E}}$, where:

$$\widehat{\mathcal{E}} :\equiv \Big(\nu\widehat{V}(d{:}B) = (d \implies \widehat{V}(d)) \wedge \widehat{W}(d)\Big)\ \Big(\mu\widehat{W}(d{:}B) = d \wedge (\neg d \implies \widehat{W}(d))\Big)$$

We find that for all $j \in \mathbb{N}$ and $b \in \mathbb{B}$, we have $[\![\mathcal{E}]\!]\eta\varepsilon(X(j,b)) = [\![\widehat{\mathcal{E}}]\!]\eta\varepsilon(\widehat{X}(b))$ for $X \in \{V, W\}$ and all η, ε. Moreover, a full instantiation of $\widehat{\mathcal{E}}$ (see [4]) leads to a BES consisting of four equations:

$$(\nu\widehat{V}_\top = \widehat{V}_\top \wedge \widehat{W}_\top)\ (\nu\widehat{V}_\bot = \widehat{W}_\bot)\ (\mu\widehat{W}_\top = \top)\ (\mu\widehat{W}_\bot = \bot)$$

where variable $\widehat{X}_d$ encodes $\widehat{X}(d)$ for $d \in \mathbb{B}$ and $X = V, W$. The above BES immediately leads to the answer *true* for variables $\widehat{V}_\top, \widehat{W}_\top$ and *false* for the variables $\widehat{V}_\bot, \widehat{W}_\bot$. Hence, using redundant parameter elimination, instantiation allows for solving equation systems that could not be solved before using this technique, solving verification problems that we could not solve before. □

4.2 Detection of Constants

As in the previous section, let $\mathcal{E}$ be an equation system, where every equation l ($1 \leq l \leq N$) is in PFNF, and let κ be a *target predicate formula* (i.e., a formula whose truth we wish to assess in the context of $\mathcal{E}$), without any free data variables, in PFNF. In this section, we develop an algorithm that automatically computes an invariant that associates constants to the formal parameters of predicate variables. To this end, we will be using a special type of simple functions called *ground functions*.

Definition 9. *A predicate formula $p \in \mathsf{Pred}$ is a* ground predicate *for a variable X of an equation system $\mathcal{E}$ if $p \equiv \bot$ or $p \equiv (\boldsymbol{d}_X = \boldsymbol{c})$, where $\boldsymbol{c}$ is a partially instantiated list, i.e. for all indices j, $\boldsymbol{c}[j] \in \boldsymbol{D}_X[j] \cup \{\boldsymbol{d}_X[j]\}$. A simple function g:$\mathsf{occ}(\mathcal{E}) \to \mathsf{Pred}$ is a* ground function *if, for all variables X, $g(X)$ is a ground predicate.*

Here $\boldsymbol{d}_X = \boldsymbol{c}$ is a shortcut for $\bigwedge_{1 \leq i \leq \mathsf{arity}(\boldsymbol{X})} (\boldsymbol{d}_X[i] = \boldsymbol{c}[i])$. Ground predicates formalise assertions about the values associated to the data parameters. To each parameter i, either a constant from its value domain, or its name $\boldsymbol{d}_X[i]$ is associated. In the latter case, the corresponding assertion is thus $\boldsymbol{d}_X[i] = \boldsymbol{d}_X[i]$, evaluating to $\top$.

The set of ground predicates for X in $\mathcal{E}$ is $\mathsf{GPred}_{\mathcal{E},X}$ and the set of ground functions for $\mathcal{E}$ is $\mathsf{GFunc}_{\mathcal{E}}$. The binary operator $\mathtt{SUP}$ takes two ground predicates of the same variable and yields the supremum of these ground predicates; it remains undefined for ground predicates of different variables.

$$\mathtt{SUP}((\boldsymbol{d}_X = \boldsymbol{c}), (\boldsymbol{d}_X = \boldsymbol{c}')) =_{def} (\boldsymbol{d}_X = \boldsymbol{c}'')), \text{where for all } 1 \leq i \leq n\text{:}$$
$$\boldsymbol{c}''[i] = \begin{cases} \boldsymbol{c}[i], & \text{if } \boldsymbol{c}[i] = \boldsymbol{c}'[i] \\ \boldsymbol{d}_X[i], & \text{otherwise} \end{cases} \tag{4}$$

For instance, if X has arity 3, then $\mathtt{SUP}((\boldsymbol{d}_X = \langle 2, 0, \boldsymbol{d}_X[3]\rangle),\ (\boldsymbol{d}_X = \langle 2, 5, \boldsymbol{d}_X[3]\rangle))$ yields $(\boldsymbol{d}_X = \langle 2, \boldsymbol{d}_X[2], \boldsymbol{d}_X[3]\rangle)$. The operator $\mathtt{SUP}$ extends naturally to sets of ground predicates of the same variable; as a convention, we set $\mathtt{SUP}(\emptyset) =_{def} \bot$.

We call $X(\boldsymbol{e})$, with $\boldsymbol{e}$ a list of data expressions, an *instantiated occurrence* of X and we denote the set of all instantiated occurrences of predicate variables in ϕ by $i\mathsf{occ}(\phi)$. From any such occurrence $X(\boldsymbol{e})$, we can extract a ground predicate by retaining only those data expressions which are constants:

$$\mathtt{gpred}(X(\boldsymbol{e})) =_{def} (\boldsymbol{d}_X = \boldsymbol{c}),\ \text{with } \boldsymbol{c}[i] = \begin{cases} c, & \text{if } \boldsymbol{e}[i] \leftrightarrow c \text{ and } c \in \boldsymbol{D}_X[i] \\ \boldsymbol{d}_X[i], & \text{otherwise.} \end{cases} \tag{5}$$

Finally, let ϕ be a formula in PFNF: $\mathsf{Q}_1 v_1 \ldots \mathsf{Q}_n v_n . h \wedge \bigwedge_{i \in I} g_i \implies \bigvee_{j \in J_i} X^j(\boldsymbol{e}^j)$. Then the *guard* of an instantiation $X^j(\boldsymbol{e}^j)$ in ϕ, written $\mathtt{guard}(X^j(\boldsymbol{e}^j), \phi)$ is defined as $h \wedge g_i$.

A constant detection algorithm. The recursive function ga generates successive ground functions that approximate the parameter lists of $\mathcal{E}$'s predicate variables reached when starting instantiation of $\mathcal{E}$ from target predicate formula κ.

```
ConstElm($\mathcal{E},\kappa$))

  for $X \in \mathsf{occ}(\mathcal{E})$,
        $\mathsf{ga}_0(X) \coloneqq \mathtt{SUP}(\{\mathtt{gpred}(X(\boldsymbol{e})) \mid X(\boldsymbol{e}) \in \imath\mathsf{occ}(\kappa) \wedge \mathtt{guard}(X(\boldsymbol{e}),\kappa) \not\leftrightarrow \bot\})$
  for $X \in \mathsf{occ}(\mathcal{E})$,
        $\mathsf{ga}_{n+1}(X) \coloneqq \mathtt{SUP}(\{\mathsf{ga}_n(X)\} \cup \bigcup_{\mathsf{ga}_n(Y)\equiv(\boldsymbol{d}_Y=\boldsymbol{e})} \{\mathtt{gpred}(X(\boldsymbol{e'})) \mid$
              $X(\boldsymbol{e'}) \in \imath\mathsf{occ}(\phi_Y[\boldsymbol{e}/\boldsymbol{d}_Y]) \wedge \mathtt{guard}(X(\boldsymbol{e'}),\phi_Y[\boldsymbol{e}/\boldsymbol{d}_Y]) \not\leftrightarrow \bot\})$
        Output : $\mathsf{gi} \equiv \bigvee_{n\geq 0} \mathsf{ga}_n$
```

Both in the definition of GPred and in the algorithm, the existence of a sound decision method for $\phi \leftrightarrow \psi$ is assumed. Usually a very simple one, like syntactic equivalence, will produce meaningful enough invariants.

Correctness. We next give a formal argument for the correctness of the constant detection algorithm; that is, we show that the output of the algorithm yields an invariant of the original PBES which preserves the truth of κ.

The function ga_n captures the information gathered from the arguments of the predicate variables after n substitution steps. $\mathsf{ga}_n(X_k) = \bot$ has the intuitive meaning that X_k is unreachable from κ within n substitution steps and $\mathsf{ga}_n(X_k) = \top$ holds when none of X_k's arguments remain constant. We have the following simple observation:

Lemma 3. *For all $n \geq 0$ and all $X \in \mathsf{occ}(\mathcal{E})$, $\mathsf{ga}_n(X) \rightarrow \mathsf{ga}_{n+1}(X)$.*

Proof (sketch). It is easy to prove that $p \rightarrow \mathtt{SUP}(S)$ for all $p \in S$ with S a set of ground predicates for X. Then, from the way ga_{n+1} is constructed, namely $\mathsf{ga}_{n+1}(X) = \mathtt{SUP}(\{\mathsf{ga}_n(X)\} \cup S)$, we immediately conclude that $\mathsf{ga}_n(X) \rightarrow \mathsf{ga}_{n+1}(X)$. □

Lemma 4. $\mathtt{ConstElm}(\mathcal{E},\kappa)$ *terminates for every $\mathcal{E}$ and κ.*

Proof (sketch). In the $\mathtt{ConstElm}(\mathcal{E},\kappa)$ computation, an index N is reached for which $\mathsf{ga}_N \equiv \mathsf{ga}_{N+1}$ (for every predicate variable X, $\mathsf{ga}_N(X) \equiv \mathsf{ga}_{N+1}(X)$); for this N, we have $\mathsf{ga}_N = \mathsf{ga}_{N+k}$ for every k (i.e., ga is stable). The first part of this claim follows from the finiteness of $\mathsf{bnd}(\mathcal{E})$ and the observation that there is a decreasing measure that can be associated to the $\mathsf{ga}_0 \ldots \mathsf{ga}_n \ldots$ sequence (viz., the number of constants occurring in ga_n). The second part follows by induction. The output of the algorithm is then $\bigvee_{0\leq i\leq N} \mathsf{ga}_N$; following Lemma 3, this is equivalent to ga_N. □

The theorem below states that the algorithm indeed yields valid invariants.

Theorem 3. *The output of the algorithm* $\mathtt{ConstElm}(\mathcal{E},\kappa)$ *is a global invariant for $\mathcal{E}$.*

Proof (sketch). Lemma 4 shows that gi is in fact ga_N for some sufficiently large N. We prove by contradiction that ga_N satisfies the sufficient condition from Theorem 1. The argument essentially uses Lemma 3. Consequently, ga_N is a global invariant. □

Using the invariant gi computed by ConstElm, we can now strengthen the original PBES. The strengthened system is, according to the definition from [13]:

$$\begin{aligned}
&\mathsf{red}(\epsilon) &&= \epsilon\\
&\mathsf{red}((\sigma X(\boldsymbol{d}_X{:}\boldsymbol{D}_X) = \phi)\ \mathcal{E}') &&= (\sigma X(\boldsymbol{d}_X{:}\boldsymbol{D}_X) = \mathsf{gi}(X) \wedge \phi)\ \mathsf{red}(\mathcal{E}')
\end{aligned}$$

Note that if $\mathsf{gi}(X) \equiv (\boldsymbol{d}_X = \boldsymbol{c})$ then $\mathsf{gi}(X) \wedge \phi$ is in fact logically equivalent to $\phi[\boldsymbol{c}/\boldsymbol{d}_X]$, meaning that the number of free variables in ϕ decreases. Using the redundant parameter elimination technique of the previous section, the equation system can then be reduced. So, red indeed reduces the complexity of equation systems. It suffices to show that this strengthening preserves the truth for κ:

Theorem 4. *Let η and ϵ be arbitrary predicate and data environments, respectively. Then $[\![\kappa]\!]([\![\mathcal{E}]\!]\eta\epsilon)\epsilon = [\![\kappa]\!]([\![\mathsf{red}(\mathcal{E})]\!]\eta\epsilon)\epsilon$.* □

Proof (sketch). The preservation of κ's solution follows from Corollary 2 of [13] and the identity $\kappa \leftrightarrow \kappa \left[_{X_i \in V} (\mathsf{gi}(X_i) \wedge X_i(d_{X_i}))_{\langle d_{X_i} \rangle}/X_i\right]$, which can be shown by a series of calculations. □

Example 4. Let κ be the target formula $\forall v{:}\mathbb{N}.\ X(1,v)$, and $\mathcal{E}$ defined as:

$$\begin{array}{ll}(\mu X(m{:}\mathbb{N}, n{:}\mathbb{N}) & = m \leq 10 \Rightarrow (X(m, n+1) \vee Y(m)))\\ (\nu Y(p{:}\mathbb{N}) & = X(p,0) \wedge (p \geq 5 \Rightarrow Z(p)))\\ (\mu Z(q{:}\mathbb{N}) & = q \leq 10)\end{array}$$

The `ConstElm` algorithm produces the following approximations:

$$\begin{array}{lll}\mathsf{ga}_0(X) = (m = 1 \wedge n = n) & \mathsf{ga}_0(Y) = \bot & \mathsf{ga}_0(Z) = \bot\\ \mathsf{ga}_1(X) = (m = 1 \wedge n = n) & \mathsf{ga}_1(Y) = (p = 1) & \mathsf{ga}_1(Z) = \bot\\ \mathsf{ga}_2(X) = (m = 1 \wedge n = n) & \mathsf{ga}_2(Y) = (p = 1) & \mathsf{ga}_2(Z) = \bot\end{array}$$

Strengthening each equation with the invariant gi (which assigns $m = 1$ to X, $p = 1$ to Y and $\bot$ to Z), and simplifying the resulting right-hand sides of the equations, we obtain the following reduced equation system:

$$\begin{array}{ll}(\mu X(m{:}\mathbb{N}, n{:}\mathbb{N}) & = X(1, n+1) \vee Y(1))\\ (\nu Y(p{:}\mathbb{N}) & = X(1,0))\\ (\mu Z(q{:}\mathbb{N}) & = \bot)\end{array}$$

Using the technique from the previous section, we find that formal parameters m, n, p and q all become redundant. The solution to this system of equations has $\bot$ as a solution for X, Y and Z, which leads to the solution $\bot$ for κ. Note that the instantiation solving technique does not terminate on the original equation system $\mathcal{E}$, nor does the redundant parameter elimination technique remove formal parameters from $\mathcal{E}$. □

Complexity and implementation. Denote the number of predicate variables in $\mathsf{occ}(\mathcal{E})$ by v, the maximum length for the argument list of a predicate variable by l, and the maximum number of occurrences of one variable in all the right-hand sides of $\mathcal{E}$'s equations by o. For each iteration n, ga_n can be computed in $\mathcal{O}(v \times l \times o)$ (ignoring the complexity of rewriting that may be necessary), since there are v variables and for each $\mathsf{ga}_n(X)$, the `SUP` of a set of at most o ground predicates is computed. This requires comparisons of arrays of length l. Every variable $\boldsymbol{d}_X[i]$ from an argument list can appear in the ground approximations of X as a constant ($\boldsymbol{d}_X[i] = c \in \boldsymbol{D}_X[i]$) or as variable $\boldsymbol{d}_X[i] = \boldsymbol{d}_X[i]$. Once its assertion becomes $\boldsymbol{d}_X[i] = \boldsymbol{d}_X[i]$, it will never become of the constant type again. Since the total number of constant assertions is decreasing with every iteration (see also the proof of Lemma 3), and since there are at most $v \times l$ data variables in the system, the maximum number of iterations until stabilisation is $v \times l$. Hence, an upper bound on the total cost of `ConstElm` is $\mathcal{O}(v^2 \times l^2 \times o)$.

In practice, checking whether the guards are unsatisfiable requires careful bookkeeping and can be inefficient. A sound solution is to over-approximate all guards to $\top$, leading to a much quicker algorithm (which may compute weaker invariants), while preserving soundness (the same proof applies).

5 Experiments

For conducting our experiments, we have used the tool-suite mCRL2[1], which implements techniques from [7,4] and for which we implemented the redundant parameter elimination and the constant parameter elimination methods. All experiments described in the remainder of this section have been conducted on a Dual Core, 2.6GHz AMD Opteron Processor running Linux with 128Gb memory.

Redundant Parameter Elimination. The first series of experiments consists of an analysis of several communication protocols from the literature: the Alternating Bit Protocol, the Positive Acknowledgement Retransmission Protocol, the Concurrent Alternating Bit Protocol and variations of the Sliding Window Protocol with different buffer size. Note that none of these system descriptions could be simplified using the related parameter elimination techniques from [6] without manually changing the descriptions.

The verification problem that we encoded for each of these protocols is deadlock-freedom. We varied the size of the set of messages M that can be communicated from $2, 4, 8$ to ∞, and measured the increase in the size of the BES that is obtained by instantiating the respective PBES before and after applying the redundant parameter elimination technique of Section 4.1, see Table 1. Running the latter algorithm takes less than $.1$ seconds for every equation system (the number of data parameters for each equation system varies from 13 to 22 per equation). Clearly, the size of the set of message has a significant impact on the size of the BESs, as witnessed e.g., for the SWP with buffer size 4: instantiating the equation system for the SWP with $|M| = 4$ is not viable in a reasonable amount of time. In contrast, the redundant parameter elimination method detects that the content of the messages is irrelevant for deadlock-freedom and therefore eliminates all references to messages from the equation system, resulting in small BESs. A similar reduction in size can be obtained by first abstracting out the data from the protocol descriptions and then checking the resulting systems for deadlock-freedom; this, however, requires in-depth knowledge about the behaviours of the systems.

A second batch of experiments conducted using the same protocols and the same setup is to verify whether a sender can infinitely often send a particular (constant) message m, which is formalised by the following formula of fixed point alternation depth 2: $\nu X.\ \mu Y.\ \langle r(m) \rangle X \vee \langle \neg r(m) \rangle Y$, where the action r models the sending of the message to the communications protocol (which is $\underline{r}$eceiving the message). The number of data parameters for the equation systems again varies from 13 to 22 per equation). Unlike for

[1] `http://www.mcrl2.org`

Table 1. The effect of redundant parameter elimination in equation systems encoding (1) deadlock-freedom ($\nu X.\ [\top]X \wedge \langle\top\rangle\top$) and (2) infinitely-often sending a constant message ($\nu X.\mu Y.\ \langle r(m)\rangle X \vee \langle\neg r(m)\rangle Y$) of various communication protocols using a set M of messages. Here, the x in x/y stands for the size of the BES before elimination of redundancy and the y in x/y stands for the size of the BES after elimination of redundancy.

Property (1)

$\lvert M\rvert \rightarrow$ Protocol $\downarrow$	2	4	8	∞
ABP	74 / 38	146 / 38	290 / 38	∞ / 38
PAR	91 / 47	179 / 47	355 / 47	∞ / 47
CABP	464 / 224	1,040 / 224	2,576 / 224	∞ / 224
One-bit SWP	324 / 144	900 / 144	2,916 / 144	∞ / 144
SWP (buffer size 2)	14,064 / 1,860	140,352 / 1,860	$1.7 * 10^6$ / 1,860	∞ / 1,860
SWP (buffer size 4)	$2.6 * 10^6$ / 43,320	nc / 43,320	nc / 43,320	∞ / 43,320

Property (2)

$\lvert M\rvert \rightarrow$ Protocol $\downarrow$	2	4	8	∞
ABP	77 / 41	149 / 41	293 / 41	∞ / 41
PAR	95 / 51	183 / 51	359 / 51	∞ / 51
CABP	513 / 257	1,121 / 257	2,721 / 257	∞ / 257
One-bit SWP	379 / 181	991 / 181	3,079 / 181	∞ / 181
SWP (buffer size 2)	17,809 / 2,545	163,393 / 2,545	$1.9 * 10^6$ / 2,545	∞ / 2,545
SWP (buffer size 4)	$3.5 * 10^6$ / 64,609	nc / 64,609	nc / 64,609	∞ / 64,609

the deadlock freedom property, the presence of the data message in the system description is vital, which means that abstracting the data away is no option. First applying the redundant parameter elimination technique from [6] therefore yields no improvement. The results of our experiments are depicted in the second matrix in Table 1 and show a similar trend as the first batch of experiments. It is important to note that the resulting BESs are also alternating. Thus, size really becomes a problem as the best, known algorithms for solving BESs are exponential in the fixed point alternation depth and have as base the size of the BES. Large BESs thus quickly become intractable.

Constant Parameter Elimination. As demonstrated in [13,8], well chosen invariants help to simplify equation systems, so that rewriting becomes less of a bottleneck in instantiating. Figure 1 clearly illustrates this effect: applying a constant parameter elimination

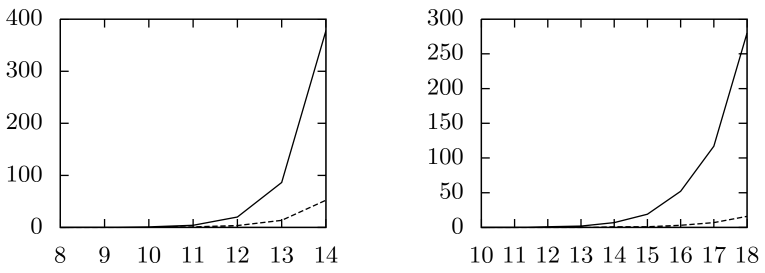


Fig. 1. The effect of constant parameter elimination on the instantiation time required to obtain a BES from an equation system encoding the deadlock-freedom property for n dining philosophers (left) and for n Milner schedulers (right). Time is measured in minutes (y-axis).

in the equation systems that encode deadlock-freedom in n dining philosophers and a token ring of n Milner schedulers, respectively. The time required to solve the original equation systems increases exponentially with increasing n in both cases (continuous lines). In contrast, after applying a constant parameter elimination, the increase in time to compute the resulting equation system is modest (dashed lines). Removing the constant parameters requires little time: < 10 seconds for the most complex case.

6 Summary

We have devised algorithms that reduce the complexity of PBESs; this is achieved by detecting and removing parameters that do not contribute to the solution of a PBES and by removing constant parameters. Our experiments show that the algorithms are very effective at reducing the size of the BESs that can be computed from the PBESs. This means that the complexity of solving the PBES via a resolution of the BES can be reduced as well, which is particularly important for alternating BESs.

As we observed and proved, a sufficient invariance condition for PBESs can be stated that does not require a quantification over arbitrary predicate environments. Our constant detection algorithm is a special case of an invariance detection algorithm for PBESs, based on the above-mentioned observation; it is therefore interesting future work to extend the constant detecting algorithm in order to detect more complex classes of invariants.

References

1. Chen, T., Ploeger, B., van de Pol, J., Willemse, T.A.C.: Equivalence checking for infinite systems using parameterized boolean equation systems. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 120–135. Springer, Heidelberg (2007)
2. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. The MIT Press, Cambridge (1999)
3. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL (1977)
4. van Dam, A., Ploeger, B., Willemse, T.A.C.: Instantiation for parameterised boolean equation systems. In: Fitzgerald, J.S., Haxthausen, A.E., Yenigun, H. (eds.) ICTAC 2008. LNCS, vol. 5160, pp. 440–454. Springer, Heidelberg (2008)
5. Gallardo, M.M., Joubert, C., Merino, P.: Implementing influence analysis using parameterised boolean equation systems. In: Proc. ISOLA 2006. IEEE Comp. Soc. Press, Los Alamitos (2006)
6. Groote, J.F., Lisser, B.: Computer assisted manipulation of algebraic process specifications. SIGPLAN Notices 37(12), 98–107 (2002)
7. Groote, J.F., Willemse, T.A.C.: Model-checking processes with data. Sci. Comput. Program 56(3), 251–273 (2005)
8. Groote, J.F., Willemse, T.A.C.: Parameterised boolean equation systems. Theor. Comput. Sci. 343(3), 332–369 (2005)
9. Hentze, N., McAllester, D.: Linear-time subtransitive control flow analysis. In: Proc. PLDI 1997. ACM, New York (1997)
10. Huth, M., Jagadeesan, R., Schmidt, D.A.: Modal transition systems: A foundation for three-valued program analysis. In: Sands, D. (ed.) ESOP 2001. LNCS, vol. 2028, p. 155. Springer, Heidelberg (2001)

11. Mateescu, R.: Local model-checking of an alternation-free value-based modal mu-calculus. In: Proc. 2nd Int'l Workshop on VMCAI (September 1998)
12. Mateescu, R., Thivolle, D.: A model checking language for concurrent value-passing systems. In: Cuellar, J., Maibaum, T., Sere, K. (eds.) FM 2008. LNCS, vol. 5014, pp. 148–164. Springer, Heidelberg (2008)
13. Orzan, S., Willemse, T.A.C.: Invariants for parameterised boolean equation systems. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 187–202. Springer, Heidelberg (2008)
14. van de Pol, J.C., Valero Espada, M.: Modal abstractions in μCRL*. In: Proc. AMAST (2004)
15. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific J. Mathematics 5(2), 285–309 (1955)
16. Watanabe, H., Nishizawa, K., Takaki, O.: A coalgebraic representation of reduction by cone of influence. In: Proc. of Workshop on Coalgebraic Methods in Computer Science, vol. 164(1), pp. 177–194 (2006)