

(Re)moving Quantifiers to Simplify Parameterised Boolean Equation Systems*

Thomas Neele

Eindhoven University of Technology, Eindhoven, The Netherlands

Abstract

We investigate the simplification of parameterised Boolean equation systems (PBESs), a first-order fixpoint logic, by means of quantifier manipulation. Whereas the occurrence of quantifiers in a PBES can significantly increase the effort required to solve a PBES, *i.e.*, compute its semantics, existing syntactic transformations have little to no support for quantifiers. We resolve this, by proposing a static analysis algorithm that identifies which quantifiers may be moved between equations such that their scope is reduced. This syntactic transformation can drastically reduce the effort required to solve a PBES. Additionally, we identify an improvement to the computation of *guards*, which can be used to strengthen our static analysis.

1. Introduction

A *parameterised Boolean equation system* (PBES) [2] is a sequence of equations of the shape $\sigma X(\mathbf{d}:\mathbf{D}) = \varphi$, where $\sigma \in \{\mu, \nu\}$ is a fixpoint sign, X is a predicate variable, \mathbf{d} of type \mathbf{D} is a sequence of parameters and φ is a predicate formula over \mathbf{d} . This equation specifies a *solution* for X , which is a set of values \mathbf{d} may take on and which depends recursively on other predicate variables in the PBES; the fixpoint sign indicates whether we are looking for a least (μ) or greatest (ν) set of values.

PBESs are the first-order extension of *Boolean equation systems* (BESs) or, equivalently, *parity games*. Similar to BESs, they can be used to encode decision problems, and hence see various applications, including model checking of modal μ -calculus formulae [3] and equivalence checking of processes [4]. PBESs are strictly more expressive than BESs, however, since some decision problems cannot be finitely encoded and require the first-order data of a PBES. The general concept of equation systems with fixpoints over complete lattices is captured in *hierarchical equation systems* [5].


Since directly computing the solution of a PBES is not straightforward, the preferred approach is to *instantiate* [6] its underlying BES, if that is finite (for infinite BESs, symbolic techniques can be a reasonable alternative [7]). However, this BES may be prohibitively large, thus preventing us in practice from obtaining a solution through instantiation.


ARQNL 2022: Automated Reasoning in Quantified Non-Classical Logics, 11 August 2022, Haifa, Israel

*This work previously appeared in the PhD thesis of the author [1, Chapter 7].

✉ t.s.neele@tue.nl (T. Neele)

ORCID 0000-0001-6117-9129 (T. Neele)

 © 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

This problem has been studied in the past, resulting in the creation of various syntactic transformations [8, 9, 10] for PBESs that aim to reduce the size of the underlying BES. A shortcoming, though, is that those techniques mostly have no support for quantifiers that may occur in the predicate formulae that form the right-hand side of equations. At the same time, these quantifiers can be (partially) responsible for the large size of the underlying BES.

We address this, and achieve a larger reduction than previous work. Our approach is based on the observation that, under certain conditions, a quantifier may be moved from one equation to another, an operation which we call *propagation*. This effectively reduces the scope of said quantifier, and limits the impact it has on the size of the underlying BES. Since propagation cannot be applied to PBESs with cyclic dependencies, we instead propose to perform static analysis to determine which predicate variables always occur in the same quantifier context, either directly or indirectly. The resulting information, which we call *global propagated values*, can be used to move quantifiers to other equations or even eliminate them completely. Our algorithm is a generalisation of the constant parameter elimination technique from [9]. Experiments with PBESs from the literature show that this may result in significant reduction of the size of the underlying BES, in one case even reducing an infinite BES to a finite one. The experiments also help us identify a couple of limitations, including the fact that our technique cannot deal with quantified variables whose domain is bounded by an expression.

The static analysis can be strengthened by the application of *guards*: predicate formulae that characterise the dependency between predicate variables. We show how guards can be computed such that information about quantifiers is preserved. As a result, we produce stronger guards than those in [11].

Related Work We informally describe several existing syntactic transformation techniques for PBESs. The first technique is *constant elimination* [9], which identifies parameters whose value never changes. The analysis takes constant values from a *target expression* κ and propagates them through the PBES to determine whether their value stays constant. If so, this invariant [2, 12] is used to simplify the right-hand side of equations. Constant elimination by itself never results in state space reduction, but the resulting simplifications may speed up PBES instantiation and also enable other reduction techniques.

The same paper also discusses *redundant parameter elimination* [9], which identifies parameters that do not influence any of the conditions in the PBES. This is achieved by extracting from the PBES an influence graph over parameters, together with a set of parameters that are considered influential. Backwards reachability yields the set of all transitively influential parameters. Eliminating non transitively influential parameters from the PBES preserves its semantics, but reduces the size of the underlying BES. A generalisation is the state graph technique of [8], which distinguishes *control flow* parameters—parameters which can take a limited number of values and are updated deterministically—from data parameters, and performs an analysis of influential data parameters per control flow location. A data parameter that is not influential in a certain control flow location is reset to a fixed value when taking a transition to that location.

Finally, abstractions may also benefit PBESs [10], and can for example be used to perform model checking of real-time systems. The approach in [10] requires a manual definition of the necessary abstraction, and is thus not fully-automated like the other works discussed above.

Overview We introduce basic concepts in Section 2. Section 3 provides a motivating example. Section 4 shows how to compute and apply global propagated values to simplify PBESs. We discuss guards in Section 5. Section 6 presents a small experiment and discusses limitations of the ideas. Finally, Section 7 concludes.

2. Preliminaries

Sequences We adopt the following notation for finite sequences. Firstly, the empty sequence is denoted ϵ . Given a sequence \mathbf{v} , $\mathbf{v}[i]$ denotes its i th element. If \mathbf{v} is a sequence of length n , then a strictly increasing, total function $f : \{1, \dots, n'\} \rightarrow \{1, \dots, n\}$ characterises a *subsequence* \mathbf{v}' of length $n' \leq n$, viz., $\mathbf{v}'[i] = \mathbf{v}[f(i)]$ for all $1 \leq i \leq n'$. The fact that \mathbf{v}' is a subsequence of \mathbf{v} with characterising function f is denoted $\mathbf{v}' \sqsubseteq_f \mathbf{v}$. We omit the subscript f if it is not relevant. Furthermore, given two sequences \mathbf{v}' and \mathbf{v} such that $\mathbf{v}' \sqsubseteq_f \mathbf{v}$, we write $\mathbf{v}[j] \in \mathbf{v}'$ iff there is an i such that $f(i) = j$; for such cases the partial inverse function f^{-1} yields $f^{-1}(j) = i$. We use list comprehensions to filter a sequence and obtain a subsequence, e.g., $[\mathbf{v}[i] \mid i \bmod 2 = 0]$ is the unique subsequence that contains the elements with an even index in \mathbf{v} and only those. Lastly, we can calculate the complement of a subsequence, notation $\mathbf{v} \setminus \mathbf{v}'$, which is defined as $[\mathbf{v}[i] \mid \mathbf{v}[i] \notin \mathbf{v}']$. To concatenate two sequences, we use $++$.

Abstract Data In this work, we consider abstract data types and expressions involving those types. Syntactic and semantic data types are distinguished: syntactic data *sorts* are denoted with D, D' etc., while the corresponding semantic *domains* are denoted \mathbb{D}, \mathbb{D}' etc. We assume that the domains are non-empty; we typically use letters v, w, \dots for the values they contain. In our examples, we may use B and N (domains \mathbb{B} and \mathbb{N}) to represent the sorts of Booleans and natural numbers, respectively. Expressions contain data variables (we typically use names such as d, d', d_1, \dots) from the set \mathcal{V} . The notation $d:D$ indicates that the sort of variable d is D .

Interpreting expressions is done by the interpretation function $\llbracket \cdot \rrbracket \delta$, which requires a *data environment* δ that maps variables to a value from their corresponding semantic domain. Intuitively, $\llbracket f \rrbracket \delta$ denotes the semantics of the expression f in the context of δ . The data environment is not relevant for interpreting an expression that does not contain variables, called a *ground term*, in that case we write $\llbracket f \rrbracket$. An update to an environment δ is written $\delta[v/d]$, defined as $\delta[v/d](d) = v$ and $\delta[v/d](e) = \delta(e)$ for all $e \neq d$. We also allow sequences of values and sequences of variables in data environment updates, i.e., given a sequence of distinct variables \mathbf{d} and a sequence of values \mathbf{v} of equal length and equal sort, $\delta[\mathbf{v}/\mathbf{d}]$ is the data environment $\delta[\mathbf{v}[1]/\mathbf{d}[1]] \dots [\mathbf{v}[n]/\mathbf{d}[n]]$.

Equation Systems A *parameterised Boolean equation system* (PBES) is a sequence of equations over predicate formulae, where each equation is labelled with either a least or a greatest fixpoint. Here, we shortly introduce the relevant concepts, the interested reader is referred to [2] for a detailed discussion.

Definition 1. A *predicate formula* is defined by the following grammar:

$$\varphi ::= b \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \neg \varphi \mid \exists d:D. \varphi \mid \forall d:D. \varphi \mid X(\mathbf{e})$$

where b is an expression of sort B , d is a variable of sort D , X is a *predicate variable* of sort $\mathbf{D}_X \rightarrow B$, which is taken from some set \mathcal{X} of sorted predicate variables and argument \mathbf{e} of sort \mathbf{D}_X is a sequence of expressions. A shorthand for $\neg\varphi \vee \psi$ is $\varphi \Rightarrow \psi$. A predicate formula φ is interpreted in the context of a data environment δ and *predicate environment* η , which maps each predicate variable $X \in \mathcal{X}$ to a subset of the corresponding semantic domain \mathbb{D}_X . This is denoted $\llbracket \varphi \rrbracket_{\eta\delta}$ and inductively defined as:

$$\begin{aligned} \llbracket b \rrbracket_{\eta\delta} &\Leftrightarrow \llbracket b \rrbracket_{\delta} & \llbracket X(\mathbf{e}) \rrbracket_{\eta\delta} &\Leftrightarrow \llbracket \mathbf{e} \rrbracket_{\delta} \in \eta(X) \\ \llbracket \varphi \wedge \psi \rrbracket_{\eta\delta} &\Leftrightarrow \llbracket \varphi \rrbracket_{\eta\delta} \text{ and } \llbracket \psi \rrbracket_{\eta\delta} \text{ hold} & \llbracket \varphi \vee \psi \rrbracket_{\eta\delta} &\Leftrightarrow \llbracket \varphi \rrbracket_{\eta\delta} \text{ or } \llbracket \psi \rrbracket_{\eta\delta} \text{ hold} \\ \llbracket \neg\varphi \rrbracket_{\eta\delta} &\Leftrightarrow \llbracket \varphi \rrbracket_{\eta\delta} \text{ does not hold} \\ \llbracket \forall d:D. \varphi \rrbracket_{\eta\delta} &\Leftrightarrow \text{for all } v \in \mathbb{D}, \llbracket \varphi \rrbracket_{\eta\delta}[v/d] \text{ holds} \\ \llbracket \exists d:D. \varphi \rrbracket_{\eta\delta} &\Leftrightarrow \text{for some } v \in \mathbb{D}, \llbracket \varphi \rrbracket_{\eta\delta}[v/d] \text{ holds} \end{aligned}$$

An expression of the shape $X(\mathbf{e})$ is called a *predicate variable instance* (PVI). The set of all PVIs occurring in a formula φ is $\text{iocc}(\varphi)$. A predicate formula is *monotone* if and only if all PVIs occur in the scope of an even number of negations. Substitution of a subformula ψ by ψ' in φ is denoted $\varphi[\psi'/\psi]$; this is also lifted to sequences, so we allow $\varphi[\psi'_1, \dots, \psi'_n/\psi_1, \dots, \psi_n]$ in the same way as for environment updates. The set of data variables occurring freely in φ , *i.e.*, those not bound in a quantifier, is $\text{vars}(\varphi)$.

Definition 2. A *parameterised Boolean equation system* (PBES) is a sequence of equations, as specified by the following grammar:

$$\mathcal{E} ::= \emptyset \mid (\mu X(\mathbf{d}:\mathbf{D}) = \varphi)\mathcal{E} \mid (\nu X(\mathbf{d}:\mathbf{D}) = \varphi)\mathcal{E}$$

where \emptyset is the empty PBES, μ denotes the least and ν denotes the greatest fixpoint, $X \in \mathcal{X}$ is a predicate variable, \mathbf{d} of sort \mathbf{D} is a sequence that contains the parameters of X , which are pairwise distinct, and φ is a monotone predicate formula.

As we will see later in the definition of the semantics of a PBES, the order of equations is relevant, and a PBES is thus not simply a set of equations. We do not write the trailing \emptyset and typically use σ to refer to an arbitrary fixpoint. The variables that occur on the left-hand side of an equation in a PBES \mathcal{E} are *bound* in \mathcal{E} ; $\text{bnd}(\mathcal{E})$ is the set of variables bound in \mathcal{E} . A PBES \mathcal{E} is *well-formed* if and only if for every $X \in \text{bnd}(\mathcal{E})$, there is precisely one equation in \mathcal{E} . We say \mathcal{E} is *closed* if it does not contain free (predicate) variables, *i.e.*, all variables occurring in the right-hand side of an equation $\sigma X(\mathbf{d}:\mathbf{D}) = \varphi$ are either bound as parameter in \mathbf{d} or by a quantifier that occurs in φ , while all predicate variables occurring in φ must be contained in $\text{bnd}(\mathcal{E})$. From here on, we only consider closed, well-formed PBESs. Therefore, each $X \in \text{bnd}(\mathcal{E})$ uniquely identifies the components of the corresponding equation, and we thus denote the parameters of X as $\mathbf{d}_X:\mathbf{D}_X$ and its right-hand side as φ_X . The *signature* of \mathcal{E} is defined as $\text{sig}(\mathcal{E}) = \{(X, \mathbb{D}_X) \mid X \in \text{bnd}(\mathcal{E})\}$. Lastly, a PBES that does not contain parameters, *i.e.*, $\mathbf{d}_X = \epsilon$ for all X , is also called a *Boolean equation system* (BES).

The denotational semantics of a PBES is a predicate environment, called the *solution*.

Definition 3. Given environments η and δ , the *solution* of a PBES \mathcal{E} is inductively defined as:

$$\llbracket \emptyset \rrbracket_{\eta\delta} = \eta$$

$$\begin{aligned} \llbracket (\mu X(\mathbf{d}_X:\mathbf{D}_X) = \varphi_X) \mathcal{E} \rrbracket \eta \delta &= \llbracket \mathcal{E} \rrbracket \eta [\mu T_X / X] \delta \\ \llbracket (\nu X(\mathbf{d}_X:\mathbf{D}_X) = \varphi_X) \mathcal{E} \rrbracket \eta \delta &= \llbracket \mathcal{E} \rrbracket \eta [\nu T_X / X] \delta \end{aligned}$$

where T_X is the function $T_X(R) = \{v \in \mathbb{D}_X \mid \llbracket \varphi_X \rrbracket (\llbracket \mathcal{E} \rrbracket \eta [R/X] \delta) \delta [v/\mathbf{d}_X]\}$.

Since, for bound variables, the solution of a closed PBES is independent of the environments η and δ , we may also plainly write $\llbracket \mathcal{E} \rrbracket$. The solution $\eta = \llbracket \mathcal{E} \rrbracket$ satisfies each equation in \mathcal{E} (it holds that $v \in \eta(X) \Leftrightarrow \llbracket \varphi_X \rrbracket \eta \delta [v/\mathbf{d}]$ for all $X \in \text{bnd}(\mathcal{E})$ and δ), while giving priority to the fixpoints of the equations that occur early in the PBES. Since each predicate formula is monotone, the function T_X is also monotone and, by Tarski's theorem, it has unique least and greatest fixpoints in the complete lattice $(2^{\mathbb{D}}, \subseteq)$.

Example 1. Consider the PBES \mathcal{E} , defined as

$$\begin{aligned} \mu X(b:B) &= Y(b) \vee X(\text{true}) \\ \nu Y(b:B) &= Y(b) \wedge (b \Rightarrow X(b)) \end{aligned}$$

The semantics $\llbracket \mathcal{E} \rrbracket \eta \delta$ unfolds to $\eta [\mu T_X / X] [\nu T_Y^{\eta [\mu T_X / X]} / Y]$. Here, the greatest fixpoint νT_Y^{η} of the function

$$\begin{aligned} T_Y^{\eta}(P) &= \{v \in \mathbb{B} \mid \llbracket Y(b) \wedge (b \Rightarrow X(b)) \rrbracket (\llbracket \emptyset \rrbracket \eta [P/Y] \delta) \delta [v/b]\} \\ &= \{v \in \mathbb{B} \mid \llbracket Y(b) \wedge (b \Rightarrow X(b)) \rrbracket \eta [P/Y] \delta [v/b]\} \\ &= \{v \in \mathbb{B} \mid \llbracket Y(b) \rrbracket \eta [P/Y] \delta [v/b] \wedge \llbracket b \Rightarrow X(b) \rrbracket \eta [P/Y] \delta [v/b]\} \\ &= \{v \in \mathbb{B} \mid v \in P \wedge (v \Rightarrow v \in \eta(X))\} \end{aligned}$$

depends on $\eta(X)$. If $\text{true} \in \eta(X)$, then $\nu T_Y^{\eta} = \mathbb{B}$, otherwise $\nu T_Y^{\eta} = \{\text{false}\}$. Hence, $\text{false} \in \nu T_Y^{\eta}$ holds regardless of η . Consequently, for the least fixpoint μT_X of the function

$$\begin{aligned} T_X(R) &= \{v \in \mathbb{B} \mid \llbracket Y(b) \vee X(\text{true}) \rrbracket (\llbracket \nu Y(b:B) = Y(b) \wedge (b \Rightarrow X(b)) \rrbracket \eta [R/X] \delta) \delta [v/d]\} \\ &= \{v \in \mathbb{B} \mid \llbracket Y(b) \vee X(\text{true}) \rrbracket (\llbracket \emptyset \rrbracket \eta [R/X] [\nu T_Y^{\eta [R/X]} / Y] \delta) \delta [v/d]\} \\ &= \{v \in \mathbb{B} \mid \llbracket Y(b) \vee X(\text{true}) \rrbracket \eta [R/X] [\nu T_Y^{\eta [R/X]} / Y] \delta [v/d]\} \\ &= \{v \in \mathbb{B} \mid v \in \nu T_Y^{\eta [R/X]} \vee \text{true} \in R\} \end{aligned}$$

we must have $\text{false} \in \mu T_X$. The set $\{\text{false}\}$ is thus the least fixpoint μT_X . It follows that $\llbracket \mathcal{E} \rrbracket \eta \delta (X) = \llbracket \mathcal{E} \rrbracket \eta \delta (Y) = \{\text{false}\}$.

Note that if the equations of \mathcal{E} are swapped to create \mathcal{E}' , then the greatest fixpoint of Y becomes dominating and we have $\llbracket \mathcal{E}' \rrbracket \eta \delta (X) = \llbracket \mathcal{E}' \rrbracket \eta \delta (Y) = \mathbb{B}$. \square

Typically, we are not interested in the complete solution of a PBES, but we only want to know whether $\llbracket \hat{\mathbf{e}} \rrbracket \in \llbracket \mathcal{E} \rrbracket (\hat{X})$ holds for some *target PVI* $\hat{X}(\hat{\mathbf{e}})$, where \mathbf{e} is a sequence of ground terms. The most common way to *partially solve* a PBES \mathcal{E} , *i.e.*, compute $\llbracket \hat{\mathbf{e}} \rrbracket \in \llbracket \mathcal{E} \rrbracket (\hat{X})$, is through a procedure called *instantiation* [6], akin to state-space exploration in model checking. Starting from the target PVI, the successors of a node $X(\mathbf{e})$ are discovered by traversing the right-hand side $\varphi_X[\mathbf{e}/\mathbf{d}_X]$. The result is a BES (or sometimes a parity game) containing one equation $X_{\mathbf{e}}$ for each $X(\mathbf{e})$ reachable from $\hat{X}(\hat{\mathbf{e}})$. This BES can be solved through Gauss elimination [13] or one of the popular parity game algorithms, *e.g.*, Zielonka's recursive algorithm [14].

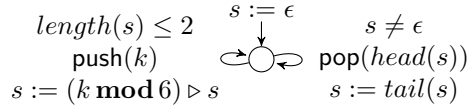


Figure 1: Extended finite state machine modelling a stack.

3. Motivating Example

To motivate our approach, we consider a small example of a stack that can hold at most two natural numbers. However, for every number k that one tries to push onto the stack, the remainder $k \bmod 6$ is stored. See the extended finite state machine in Figure 1. The stack is stored in variable s , which is initially set to the empty list. Each transition is labelled with a condition, an action with parameters and a variable update (in that order). The leftmost transition can be executed for any natural number k . Here, ϵ denotes the empty list and the functions $head$ and $tail$ respectively take the first element and the remainder of a list. Elements are prepended to a list with the operator \triangleright . This machine has a state space consisting of 43 states and an infinite number of transitions.

Suppose that we want to check the property “there is a number n for which there is an execution in which $push(n)$ is enabled infinitely often”. In the first-order modal μ -calculus, this can be expressed by the formula φ , defined as

$$\varphi = \exists n:N. \nu X. \overbrace{(\underbrace{\mu Y. \langle \top \rangle Y}_{\varphi_2} \vee \underbrace{\langle \langle push(n) \rangle true \wedge X \rangle}_{\varphi_3})}_{\varphi_1},$$

where \top indicates the set of all actions and subformulae φ_1 , φ_2 and φ_3 are as indicated. The question whether the bounded stack satisfies this formula is answered by the solution for $Z(\epsilon)$ in the following PBES (how this PBES originates from (subformulae of) φ is indicated on the right):

$$\begin{aligned} \nu Z(s:List(N)) &= \exists n:N. X(s, n) && \} \varphi \\ \nu X(s:List(N), n:N) &= Y(s, n) && \} \varphi_1 \\ \mu Y(s:List(N), n:N) &= (\exists k:N. length(s) \leq 2 \wedge Y(k \bmod 6 \triangleright s, n)) && \} \varphi_2 \\ &\quad \vee (s \neq \epsilon) \wedge Y(tail(s), n) \\ &\quad \vee (length(s) \leq 2) \wedge X(s, n) && \} \varphi_3 \end{aligned}$$

Here, we used the parameterised *List* data sort to store a finite stack of natural numbers. In the right-hand side of Y , the quantifier $\exists k:N$ in the first disjunct stems from the fact that action $push(k)$ can be executed for any k . The second disjunct corresponds to executing $pop(head(s))$. The last disjunct expresses that $push(n)$ is enabled (subformula φ_3), which is the case if and only if $length(s) \leq 2$.

Due to the quantifier in the right-hand side of the equation for Z , we are not able to finitely instantiate this PBES. A first observation that may help to resolve this issue is that in some

cases we can use (reverse) substitution of predicate variables to move quantifiers. For example, assuming W only occurs in the right-hand side of V , the following operations preserve [2] the semantics of V :

$$\begin{aligned} & \llbracket (\nu V = \forall d':D. W(d'))(\mu W(d:D) = \varphi_W) \rrbracket \eta\delta(V) \\ &= \llbracket (\nu V = \forall d':D. \varphi_W[d'/d]) \rrbracket \eta\delta(V) \\ &= \llbracket (\nu V = W)(\mu W = \forall d':D. \varphi_W[d'/d]) \rrbracket \eta\delta(V) \end{aligned}$$

In the first step, we substitute the right-hand side of Y for the PVI $W(d')$; in the second step we do the reverse, but also move the quantifier $\forall d':D$. We managed to *propagate* the quantifier from the right-hand side of V through the PVI $W(d)$ to the right-hand side of W , in the process eliminating W 's parameter and thus reducing the signature of the PBES. A similar transformation can be applied if W occurs multiple times in the surrounding PBES, as long as it always occurs in the context of the same quantifiers.

In our motivating example, this approach cannot be applied due to the circular dependency between X and Y . However, observe that after a value is chosen for n in the right-hand side of Z , it does not change any more. Thus, however many substitutions we perform in the right-hand side of Z , all occurrences of X and Y have the shape $X(f, n)$ and $Y(f, n)$, where f is some expression. This suggests that all PVIs for X and Y indirectly occur in the context of the quantifier $\exists n:N$. In the next section we show how to identify such cases and how to exploit them to reduce the signature of a PBES.

4. Global Propagated Values

As illustrated by our motivating example, we need an analysis of the complete PBES, and generally cannot rely on local propagation of quantifiers between equations. Our analysis yields so called *global propagated values*, which indicate when a predicate variable (indirectly) occurs in the context of the same quantifiers throughout the PBES. Before we discuss these ideas in detail, we first introduce several more preliminary concepts.

We generally use $\mathbf{Q} \in \{\forall, \exists\}$ to denote an arbitrary quantifier and $\mathbf{Q} \in (\{\forall, \exists\} \times \mathcal{V})^*$ to denote a finite sequence of quantified variables. Each element in such a sequence is a pair of a quantifier and a sorted variable, which is not necessarily unique in \mathbf{Q} . The function v extracts the variables, *i.e.*, $v((\mathbf{Q}, d)) = d$; lifted to sequences, we have $v(\mathbf{Q}) = v(\mathbf{Q}[1]), \dots, v(\mathbf{Q}[n])$.

As a generalisation of PVIs, we also consider *quantified predicate variable instances* (QPVI). These have the shape $\mathbf{Q}. X(\mathbf{e})$: a PVI preceded by zero or more quantifiers. The recursive function `qiocc` extracts maximal QPVI from a predicate formula:

$$\begin{aligned} \text{qiocc}(\mathbf{Q}, b) &= \emptyset & \text{qiocc}(\mathbf{Q}, \neg\varphi) &= \text{qiocc}(\epsilon, \varphi) \\ \text{qiocc}(\mathbf{Q}, \varphi \wedge \psi) &= \text{qiocc}(\epsilon, \varphi) \cup \text{qiocc}(\epsilon, \psi) & \text{qiocc}(\mathbf{Q}, \varphi \vee \psi) &= \text{qiocc}(\epsilon, \varphi) \cup \text{qiocc}(\epsilon, \psi) \\ \text{qiocc}(\mathbf{Q}, \forall d:D. \varphi) &= \text{qiocc}(\mathbf{Q} \# [(\forall, d)], \varphi) & \text{qiocc}(\mathbf{Q}, \exists d:D. \varphi) &= \text{qiocc}(\mathbf{Q} \# [(\exists, d)], \varphi) \\ \text{qiocc}(\mathbf{Q}, X(\mathbf{e})) &= \{\mathbf{Q}. X(\mathbf{e})\} \end{aligned}$$

where \mathbf{Q} is a sequence of quantified variables. Furthermore, we overload `qiocc`, such that we have $\text{qiocc}(\varphi) = \text{qiocc}(\epsilon, \varphi)$.

To reason about QPVI, parts of the expression sequence \mathbf{e} they carry and the quantifier context they occur in, we introduce the concept of *propagated values*.

Definition 4. A *propagated value* is a tuple $(\mathbf{Q}_p, \mathbf{d}_p, \mathbf{e}_p)$, where \mathbf{Q}_p is a sequence of quantified variables, \mathbf{d}_p is a sequence of variables and \mathbf{e}_p is a sequence of expressions and \mathbf{d}_p and \mathbf{e}_p are of equal length and sort.

Henceforth we will denote a propagated value with $\mathbf{Q}_p. \mathbf{d}_p := \mathbf{e}_p$. The propagated value $(\epsilon, \epsilon, \epsilon)$ is denoted \perp . The relation between QPVI and propagation values is captured in the *propagation relation*.

Definition 5. The propagation relation \hookrightarrow , which relates QPVI and propagated values, is the largest relation such that for all $(\mathbf{Q}. X(\mathbf{e})) \hookrightarrow \mathbf{Q}_p. \mathbf{d}_p := \mathbf{e}_p$ the following conditions hold:

- \mathbf{Q}_p is a suffix of \mathbf{Q} ; and
- there is a characterising function f such that $\mathbf{e}_p \sqsubseteq_f \mathbf{e}$ and $\mathbf{d}_p \sqsubseteq_f \mathbf{d}_X$; and
- $\text{vars}(\mathbf{e} \setminus \mathbf{e}_p) \cap \text{v}(\mathbf{Q}_p) = \emptyset$ and $\text{vars}(\mathbf{e}_p) \subseteq \text{v}(\mathbf{Q}_p)$.

Note that every element in \mathbf{d}_X is unique, so for every sequence \mathbf{d}_p , there is at most one function f such that $\mathbf{d}_p \sqsubseteq_f \mathbf{d}_X$. Thus, given $\mathbf{Q}. X(\mathbf{e}) \hookrightarrow \mathbf{Q}_p. \mathbf{d}_p := \mathbf{e}_p$, the sequence \mathbf{d}_p uniquely determines \mathbf{e}_p .

Let $\text{prop}(\mathbf{Q}. X(\mathbf{e}))$ denote left projection of \hookrightarrow , i.e., $\text{prop}(\mathbf{Q}. X(\mathbf{e})) = \{y \mid \mathbf{Q}. X(\mathbf{e}) \hookrightarrow y\}$. A propagated value $y = \mathbf{Q}'. \mathbf{d} := \mathbf{e}$ is associated to a predicate variable X iff $y \in \bigcup_{\mathbf{Q}, \mathbf{e}} \text{prop}(\mathbf{Q}. X(\mathbf{e}))$. To indicate this, we write $\mathbf{Q}'. \mathbf{d} \stackrel{X}{\asymp} \mathbf{e}$. A propagated value may be associated to more than one predicate variable, e.g., the propagated value \perp is associated to all predicate variables. Given some predicate variable X and two propagated values $y = \mathbf{Q}_p. \mathbf{d}_p \stackrel{X}{\asymp} \mathbf{d}_p$ and $y' = \mathbf{Q}'_p. \mathbf{d}'_p \stackrel{X}{\asymp} \mathbf{e}'_p$, we write $y \preceq_X y'$ if and only if, for some f , $\mathbf{d}_p \sqsubseteq_f \mathbf{d}'_p$, $\mathbf{e}_p \sqsubseteq_f \mathbf{e}'_p$ and \mathbf{Q}_p is a suffix of \mathbf{Q}'_p . Note again that there is at most one such f , due to the uniqueness of elements in \mathbf{d}_X .

In general, the preorder \preceq_X gives rise to a unique infimum. When considering only the set $\text{prop}(\mathbf{Q}. X(\mathbf{e}))$, the preorder \preceq_X also yields a unique supremum. Moreover $\text{prop}(\mathbf{Q}. X(\mathbf{e}))$ is finite, so $(\text{prop}(\mathbf{Q}. X(\mathbf{e})), \preceq_X)$ is a complete lattice. Although the correctness of our theory does not depend on choosing a specific propagated value in $\text{prop}(\mathbf{Q}. X(\mathbf{e}))$, we are often interested in the supremum, denoted with $\bigvee \text{prop}(\mathbf{Q}. X(\mathbf{e}))$, since it contains the most information about $\mathbf{Q}. X(\mathbf{e})$.

Example 2. Let X be a predicate variable, with parameters $\mathbf{d}_X = (d_1, d_2, d_3)$. We have $\text{prop}(\forall m:N. \exists n:N. X(d_3, m + d_2, 2n)) = \{\perp, \exists n:N. d_3 := 2n\}$. The parameters d_1 and d_2 do not occur in a propagated value since the corresponding expressions d_3 and $m + d_2$ respectively contain variables d_3 and d_2 that are not bound in the surrounding quantifiers, and hence violate the condition $\text{vars}(\mathbf{e}_p) \subseteq \text{v}(\mathbf{Q}_p)$. Consequently, $\text{vars}(\mathbf{e} \setminus \mathbf{e}_p)$ contains at least m , d_2 and d_3 ; the quantifier $\forall m$ thus cannot occur in a propagated value since this would violate the condition $\text{vars}(\mathbf{e} \setminus \mathbf{e}_p) \cap \text{v}(\mathbf{Q}_p) = \emptyset$. For the QPVI $\varphi = \exists n:N. X(3, d_2, n + 1)$, we have $\bigvee \text{prop}(\varphi) = \exists n:N. d_1, d_3 := 3, n + 1$. \square

In general, the procedure of quantifier propagation based on forward and backward substitution (outlined in Section 3) does not preserve the semantics if the variable we substitute occurs

in multiple QPVI's or in a dependency cycle. To perform an analysis of such cases, we need to reason about information that is common among two QPVI's. This information is exactly captured by the lattice $(\text{prop}(\mathbf{Q}. X(\mathbf{e})) \cap \text{prop}(\mathbf{Q}'. X(\mathbf{e}')), \preceq_X)$.

Furthermore, we also want to reason about which predicate variables are unreachable from the target PVI $\hat{X}(\hat{\mathbf{e}})$. For this, we introduce the special value \top such that $y \preceq_X \top$ for all propagated values y . The definition of the infimum \wedge in the lattice $(\bigcup_{\mathbf{Q}, \mathbf{e}} \text{prop}(\mathbf{Q}. X(\mathbf{e})) \cup \{\top\}, \preceq_X)$ follows in the standard way. We lift \wedge to sets in the ordinary way (note that \wedge is commutative and associative); \wedge of an empty set yields \top .

Algorithm 1: Computing shared information of two propagated values.

Input: $\mathbf{Q}_1. \mathbf{d}_1 : \stackrel{X}{=} \mathbf{e}_1, \mathbf{Q}_2. \mathbf{d}_2 : \stackrel{X}{=} \mathbf{e}_2$

- 1 Set f_1, f_2 such that $\mathbf{d}_1 \sqsubseteq_{f_1} \mathbf{d}$ and $\mathbf{d}_2 \sqsubseteq_{f_2} \mathbf{d}$;
- 2 $\mathbf{Q} :=$ longest common suffix of \mathbf{Q}_1 and \mathbf{Q}_2 ;
- 3 $\mathbf{d} := [\mathbf{d}_X[i] \mid \mathbf{d}_X[i] \in \mathbf{d}_1 \wedge \mathbf{d}_X[i] \in \mathbf{d}_2 \wedge \mathbf{e}_1[f_1^{-1}(i)] = \mathbf{e}_2[f_2^{-1}(i)] \wedge \text{vars}(\mathbf{e}_1[f_1^{-1}(i)]) \subseteq \text{vars}(\mathbf{e}_2[f_2^{-1}(i)])]$;
- 4 $\mathbf{e} := [\mathbf{e}_1[f_1^{-1}(i)] \mid \mathbf{d}_X[i] \in \mathbf{d}]$;
- 5 $\text{del} := \text{vars}(\mathbf{e}_1 \setminus \mathbf{e}) \cup \text{vars}(\mathbf{e}_2 \setminus \mathbf{e})$;
- 6 **while** $\text{del} \neq \emptyset$ **do**
- 7 $\mathbf{Q}' :=$ longest suffix \mathbf{Q}' of \mathbf{Q} such that $\text{vars}(\mathbf{Q}') \cap \text{del} = \emptyset$;
- 8 $\mathbf{d} := [\mathbf{d}[i] \mid \text{vars}(\mathbf{e}[i]) \subseteq \text{vars}(\mathbf{Q}')] ;$
- 9 $\mathbf{e}' := [\mathbf{e}[i] \mid \text{vars}(\mathbf{e}[i]) \subseteq \text{vars}(\mathbf{Q}')] ;$
- 10 $\text{del} := \text{vars}(\mathbf{e} \setminus \mathbf{e}')$;
- 11 $\mathbf{e} := \mathbf{e}'$;
- 12 **return** $\mathbf{Q}. \mathbf{d} : \stackrel{X}{=} \mathbf{e}$

To support the intuition, Algorithm 1 shows how to compute $\mathbf{Q}_1. \mathbf{d}_1 : \stackrel{X}{=} \mathbf{e}_1 \wedge \mathbf{Q}_2. \mathbf{d}_2 : \stackrel{X}{=} \mathbf{e}_2$. Initially, we take the quantifiers shared by \mathbf{Q}_1 and \mathbf{Q}_2 (line 2) and the parameters that are shared by \mathbf{d}_1 and \mathbf{d}_2 and for which the corresponding expressions in \mathbf{e}_1 and \mathbf{e}_2 are matching and bound in \mathbf{Q} (line 3). The list of expressions \mathbf{e} is initialised to match \mathbf{d} (line 4). Then, we collect variables from expressions in \mathbf{e}_1 and \mathbf{e}_2 that did not make it into \mathbf{e} (line 5). If such variables exist, we ensure that $\text{vars}(\mathbf{Q})$ does not contain deleted variables (line 7) and update \mathbf{d} and \mathbf{e} to take into account variables that are no longer bound in \mathbf{Q} (lines 8, 9 and 11). The update of \mathbf{e} might cause more variables to be deleted, so we update del (line 10) and iterate. Once this process finishes, the conditions $\text{vars}(\mathbf{Q}) \cap (\text{vars}(\mathbf{e}_1 \setminus \mathbf{e}) \cup \text{vars}(\mathbf{e}_2 \setminus \mathbf{e})) = \emptyset$ and $\text{vars}(\mathbf{e}) \subseteq \text{vars}(\mathbf{Q})$ are satisfied, and we return.

Example 3. Consider the following PBES:

$$\begin{aligned} \nu X(n:N) &= \mathbf{Q}_1. X(\mathbf{e}_1) \wedge \mathbf{Q}_2. X(\mathbf{e}_2) \\ \mu Y(n_1, n_2, n_3, n_4, n_5:N) &= X(n_1 + n_2 + n_3 + n_4 + n_5) \end{aligned}$$

where

$$\mathbf{Q}_1. X(\mathbf{e}_1) = \forall m_1:N. \exists m_2:N. \forall m_3:N. \exists m_4:N. Y(n + m_1, m_2, m_3 + 2, m_3, m_4)$$

$$\mathbf{Q}_2. X(\mathbf{e}_2) = \forall m_1:N. \forall m_2:N. \forall m_3:N. \exists m_4:N. Y(n + m_1, m_2, 5m_3, m_3, m_4)$$

We have

$$\bigvee \text{prop}(\mathbf{Q}_1. X(\mathbf{e}_1)) = \exists m_2:N. \forall m_3:N. \exists m_4:N. n_2, n_3, n_4, n_5 : \overset{X}{=} m_2, m_3 + 2, m_3, m_4$$

$$\bigvee \text{prop}(\mathbf{Q}_2. X(\mathbf{e}_2)) = \forall m_2:N. \forall m_3:N. \exists m_4:N. n_2, n_3, n_4, n_5 : \overset{X}{=} m_2, 5m_3, m_3, m_4$$

$$\mathbf{Q}. \mathbf{d} : \overset{X}{=} \mathbf{e} = \bigvee \text{prop}(\mathbf{Q}_1. X(\mathbf{e}_1)) \wedge \bigvee \text{prop}(\mathbf{Q}_2. X(\mathbf{e}_2)) = (\exists m_4:N. n_5 := m_4)$$

Variable n is not bound in \mathbf{Q}_1 or \mathbf{Q}_2 , so $n + m_1$ does not occur in either propagated value by the condition $\text{vars}(\mathbf{e}) \subseteq \nu(\mathbf{Q})$. The longest common suffix of quantifiers is $(\forall, m_3)(\exists, m_4)$. Thus, m_2 is also not contained in \mathbf{e} , by the same condition. Furthermore, the expressions $m_3 + 2$ and $5m_3$ are not equivalent, so parameter n_3 is excluded from \mathbf{d} . Consequently, $(\forall, m_3) \notin \mathbf{Q}$, to satisfy the condition $\text{vars}(\mathbf{e}_1 \setminus \mathbf{e}) \cap \nu(\mathbf{Q}) = \emptyset$, and $m_3 \notin \mathbf{e}$, to satisfy $\text{vars}(\mathbf{e}) \subseteq \nu(\mathbf{Q})$. \square

Let $\hat{X}(\hat{\mathbf{e}})$ be a target PVI for which we want to know the solution. Then, given a PBES \mathcal{E} , we can detect globally quantified parameters with the following algorithm. Here, the function qi distributes quantifiers over other logical operators. For now, assume that $\text{guard}^{X(\mathbf{e})}(\varphi)$ always returns *true*; its function will be explained later.

$$\text{pv}_0(\hat{X}) = (\mathbf{d}_{\hat{X}} : \overset{X}{=} \hat{\mathbf{e}})$$

for every $X \in \text{bnd}(\mathcal{E}) \setminus \{\hat{X}\}$,

$$\text{pv}_0(X) = \top$$

for every $X \in \text{bnd}(\mathcal{E})$,

$$\text{pv}_{i+1}(X) = \text{pv}_i(X) \wedge \bigwedge \{ \bigvee \text{prop}(\mathbf{Q}'. X(\mathbf{e}')) \mid \exists Y \in \text{bnd}(\mathcal{E}).$$

$$\begin{aligned} \text{pv}_i(Y) &= (\mathbf{Q}. \mathbf{d} : \overset{Y}{=} \mathbf{e}) \wedge (\mathbf{Q}'. X(\mathbf{e}')) \in \text{qiocc}(\text{qi}(\mathbf{Q}. \varphi_Y[\mathbf{e}/\mathbf{d}])) \\ &\quad \wedge \exists \delta. \llbracket \text{guard}^{X(\mathbf{e})}(\mathbf{Q}. \varphi_Y[\mathbf{e}/\mathbf{d}]) \rrbracket \delta \} \end{aligned}$$

$$\text{Output: gpv} = \bigwedge_{i \geq 0} \text{pv}_i$$

Initially, we take the ground terms from $\hat{\mathbf{e}}$ and construct a propagated value $\mathbf{d}_{\hat{X}} : \overset{X}{=} \hat{\mathbf{e}}$. For other variables, the initial propagated value is set to \top . In every iteration, we construct right-hand sides based on the propagated values we have found so far, in a similar fashion as quantifier propagation. From the new right-hand sides, we extract quantified predicates and compute their infimum with the current propagated value. This propagation of information continues until we reach a fixpoint gpv . We remark that this algorithm generalises the algorithm for finding constant parameters [9], since any constant value computed by the latter algorithm is also found by our algorithm.

The information that gpv yields is applied as follows. In an equation $\sigma X(\mathbf{d}:\mathbf{D}) = \varphi_X$, where $\text{gpv}(X) = \mathbf{Q}. \mathbf{d} : \overset{X}{=} \mathbf{e}$, we apply the substitution $\mathbf{d} := \mathbf{e}$ to φ_X and also add the quantifiers \mathbf{Q} to φ_X . Formally, given a PBES \mathcal{E} , a target PVI $\hat{X}(\hat{\mathbf{e}})$ and the global propagated value function gpv , this is captured by the function global-prop :

$$\text{global-prop}(\emptyset) = \emptyset$$

$$\text{global-prop}((\sigma X(\mathbf{d}_X:\mathbf{D}_X) = \varphi_X)\mathcal{E}') = \begin{cases} (\sigma X(\mathbf{d}_X:\mathbf{D}_X) = \mathbf{Q}. \varphi_X[\mathbf{e}/\mathbf{d}])\text{global-prop}(\mathcal{E}') & \text{if } \text{gpv}(X) = \mathbf{Q}. \mathbf{d} : \overset{\times}{=} \mathbf{e} \\ \text{global-prop}(\mathcal{E}') & \text{if } \text{gpv}(X) = \top \end{cases}$$

First, note that if $\text{gpv}(X) = \perp$ for all $X \in \text{bnd}(\mathcal{E})$, then global-prop does not modify \mathcal{E} . Otherwise, if $\text{gpv}(X) = \mathbf{Q}. \mathbf{d} : \overset{\times}{=} \mathbf{e}$, the parameters of \mathbf{d}_X that are also contained in \mathbf{d} do not occur any longer in $\mathbf{Q}. \varphi_X[\mathbf{e}/\mathbf{d}]$, the new right-hand side of X . Those parameters are now redundant, and they can be eliminated with the redundant parameter elimination technique from [9]. Subsequently, the quantifier-inside rewriter can eliminate quantifiers from QPVI's where parameter elimination has reduced the set of free variables. If $\text{gpv}(X) = \top$, then the equation for X is unreachable from $\hat{X}(\hat{\mathbf{e}})$, and it is removed from \mathcal{E} .

Example 4. Consider the target PVI $X(4)$ and the PBES below:

$$\begin{aligned} \mu X(n:N) &= \forall m':N. Y(n, m') \\ \nu Y(n, m:N) &= Z(n, m) \wedge (n \geq 2 \Rightarrow Y(n/2, m)) \wedge (n \leq 5 \Rightarrow Y(n+7, m)) \\ \mu Z(n, m:N) &= (n < m \wedge Z(n, m-1)) \vee (n \geq m) \end{aligned}$$

The globally propagated values relative to $X(4)$ are iteratively computed as follows:

$$\begin{array}{lll} \text{pv}_0(X) = n := 4 & \text{pv}_0(Y) = \top & \text{pv}_0(Z) = \top \\ \text{pv}_1(X) = n := 4 & \text{pv}_1(Y) = \forall m':N. n, m := 4, m' & \text{pv}_1(Z) = \top \\ \text{pv}_2(X) = n := 4 & \text{pv}_2(Y) = \forall m':N. m := m' & \text{pv}_2(Z) = \forall m':N. n, m := 4, m' \\ \text{pv}_3(X) = n := 4 & \text{pv}_3(Y) = \forall m':N. m := m' & \text{pv}_3(Z) = n, m := n, m \\ \text{pv}_4(X) = n := 4 & \text{pv}_4(Y) = \forall m':N. m := m' & \text{pv}_4(Z) = \perp \\ \text{gpv}(X) = n := 4 & \text{gpv}(Y) = \forall m':N. m := m' & \text{gpv}(Z) = \perp \end{array}$$

Parameter n is not constant in the equation for Y , since its value may change through the PVI $Y(n+7, m)$. That updated value can also reach Z through the PVI $Z(n, m)$, hence n is also not constant in Z . Applying the global propagated values yields:

$$\begin{aligned} \mu X(n:N) &= \forall m':N. Y(4, m') \\ \nu Y(n, m:N) &= \forall m':N. (Z(n, m') \wedge (n \geq 2 \Rightarrow Y(n/2, m')) \wedge (n \leq 5 \Rightarrow Y(n+7, m'))) \\ \mu Z(n, m:N) &= (n < m \wedge Z(n, m-1)) \vee (n \geq m) \end{aligned}$$

After redundant parameter elimination, which removes parameter n of X and m of Y , and distributing quantifiers over other operators (eliminating them when the variable they bind does not occur), we obtain

$$\begin{aligned} \mu X &= Y(4) \\ \nu Y(n:N) &= (\forall m':N. Z(n, m')) \wedge (n \geq 2 \Rightarrow Y(n/2)) \wedge (n \leq 5 \Rightarrow Y(n+7)) \\ \mu Z(n, m:N) &= (n < m \wedge Z(n, m-1)) \vee (n \geq m) \end{aligned}$$

This PBES has fewer parameters and should thus be easier to solve. In particular, we avoid the instantiation of many nodes $Y(n, m)$ for all different values of m . For $Z(n, m)$, though, we still require infinitely many nodes for all values of m . \square

Next, we shortly revisit the motivating example from Section 3.

Example 5. Consider again the PBES from Section 3. The global propagated values we obtain for the target PVI $Z(\epsilon)$ are:

$$\text{gpv}(Z) = s := \epsilon \quad \text{gpv}(X) = \exists n:N. n := n \quad \text{gpv}(Y) = \exists n:N. n := n$$

Thus, the simplified PBES obtained after applying the global-prop function, redundant parameter elimination and quantifier elimination is:

$$\begin{aligned} \nu Z &= X(\epsilon) \\ \nu X(s:List(N)) &= Y(s) \\ \nu Y(s:List(N)) &= (\exists k:N. length(s) \leq 2 \wedge Y(k \bmod 6 \triangleright s)) \\ &\quad \vee (s \neq \epsilon) \wedge Y(tail(s)) \\ &\quad \vee length(s) \leq 2 \wedge X(s) \end{aligned}$$

All references to the variables n introduced by the quantifier in the mu-calculus formula of Section 3 have been eliminated. The simplified PBES has a finite underlying BES, and is thus much more easy to solve than the PBES we started out with. \square

We have two theorems stating the correctness of the function global-prop. The first theorem claims that the computation of global propagated values terminates.

Theorem 1. *Quantified predicate analysis terminates for every PBES \mathcal{E} .*

Proof. Let \mathcal{E} be a PBES. In every iteration i , there is at least one $X \in \text{bnd}(\mathcal{E})$ for which either (i) $\text{pv}_i(X)$ changes from \top to $\mathbf{Q}. \mathbf{d} : \overset{X}{\underline{\quad}} \mathbf{e}$; or (ii) if $\text{pv}_i(X) = \mathbf{Q}. \mathbf{d} : \overset{X}{\underline{\quad}} \mathbf{e}$, then some $\mathbf{d}[i]$ is removed from \mathbf{d} or some prefix from \mathbf{Q} is removed (or both). Since \mathcal{E} consists of a finite number of equations, each with a finite number of parameters, this process must terminate. \square

The second theorem states that the function global-prop preserves the semantics of the target PVI. A proof can be found in [1, Theorem 7.25].

Theorem 2. *Let \mathcal{E} be a closed PBES and $\hat{X}(\hat{\epsilon})$ some target PVI. Then $\llbracket \hat{\epsilon} \rrbracket \in \llbracket \mathcal{E} \rrbracket(\hat{X})$ if and only if $\llbracket \hat{\epsilon} \rrbracket \in \llbracket \text{global-prop}(\mathcal{E}) \rrbracket(\hat{X})$.*

5. Guards for Predicate Formulae

The analysis of the previous section relies solely on static dependencies between predicate variables, by extracting all QPVIs $\mathbf{Q}. X(\mathbf{e})$ in a right-hand side with the function `qiocc`. It does not take into account any other parts of the predicate formula, which might cause subformula $X(\mathbf{e})$ to be irrelevant. For example, in the formula $n \leq 2 \wedge X(m)$, $X(m)$ is irrelevant if the constant $n = 7$ was deduced. A formula that characterises for which values of m the PVI $X(m)$ is relevant, is called a *guard* (a formal definition follows).

Guards also play an important role in other types of static analysis of PBESs. The concept of guards first appeared in [9], although it only shows how to construct a guard for a normal

form called *predicate formula normal form* (PFNF). Keiren *et al.* [8] propose a function that over-approximates guards for arbitrary predicate formulae. Correctness results for this guard function can be found in Keiren's thesis [11], although the proof contains a small issue (see the discussion following Theorem 3). Here, we improve on these results by strengthening the guards we compute.

Before we give a formal definition of a guard, we first introduce several auxiliary notions. Firstly, we have *logical equivalence*: $\varphi \equiv \psi$ iff for all η and δ , $\llbracket \varphi \rrbracket \eta \delta = \llbracket \psi \rrbracket \eta \delta$. The number of PVI's occurring in φ is denoted with $\text{npred}(\varphi)$ and the i th PVI in φ is $\text{PVI}(\varphi, i)$. We write $\varphi[i \mapsto \psi]$ to indicate the syntactic replacement of the i th PVI in φ (counted from left to right) by ψ . To denote the simultaneous substitution of the i th PVI by some formula ψ_i for all $1 \leq i \leq \text{npred}(\varphi)$ that satisfy the condition f , we write $\varphi[i \mapsto \psi_i]_{f(i)}$.

We now formally introduce the concept of *guards*, which originates from [11, Lemma 6.27].

Definition 6. Given a predicate formula φ , a formula ψ which does not contain PVI's is a *guard* for the i th PVI of φ if and only if $\varphi \equiv \varphi[i \mapsto \psi \wedge \text{PVI}(\varphi, i)]$.

To understand the intuition behind guards, consider an equation $\sigma X(d:D) = \varphi$, the i th PVI in φ , $\text{PVI}(\varphi, i) = Y(e)$ and associated guard ψ . The guard ψ expresses for which values of e the PVI $Y(e)$ is relevant in φ . After all, if $\llbracket \psi \rrbracket \delta$ is *false*, then $\llbracket Y(e) \rrbracket \eta \delta$ is not relevant to the value of the subformula $\psi \wedge Y(e)$.

Thus, in general, a guard over-approximates the true dependency between X and Y , as far as that dependency originates from the occurrence $\text{PVI}(\varphi, i)$. Remark that *true* is a guard for any PVI. We further demonstrate the concept in the next example.

Example 6. Consider the predicate formula

$$\varphi = (\exists m:N. m = 6 \wedge (W(m) \vee X)) \vee (Y \wedge (\neg b \Rightarrow Z))$$

The PVI's $W(m)$ and X are guarded in the same way: $m = 6$ is a guard for both these PVI's. From this, we can deduce that replacing $W(m)$ by $W(6)$ preserves logical equivalence of φ . For Y , the only guard is *true*; Z also has $\neg b$ as guard. Hence, if we somehow deduce that b never takes on the value *false*, the PVI Z can be eliminated from φ . \square

The following example shows that the substitution applied in the definition of a guard can yield unexpected results when a variable d has both bound and free occurrences.

Example 7. Consider the formula $\varphi = (n < 2) \wedge \forall n:N. X(n)$. The PVI $X(n)$ occurs in the conjunctive context of $n < 2$, which at first sight leads to believe that $n < 2$ is a guard. However, we have the logical inequivalence $\varphi \not\equiv \varphi[1 \mapsto (n < 2) \wedge X(n)]$, since the new occurrence of n introduced by the substitution is bound instead of free. Hence, $n < 2$ is not a guard. \square

To identify situations such as in the latter example, we say φ is *capture-avoiding* iff no free variable of φ has a bound occurrence in φ , *i.e.*, there is no subformula $Qd:D. \varphi'$ of φ such that $d \in \text{vars}(\varphi)$. Remark that any predicate formula can be transformed into an equivalent capture-avoiding formula by performing the appropriate alpha conversion, that is, renaming the variables bound in quantifiers.

The function `guard` computes guards for capture-avoiding formulae.

Definition 7. Let φ be a capture-avoiding predicate formula and $i \leq \text{npred}(\varphi)$. Then, the function guard is defined inductively as follows:

$$\begin{aligned}
\text{guard}^i(Y(\mathbf{e})) &= \text{true} \\
\text{guard}^i(\neg\varphi) &= \text{guard}^i(\varphi) \\
\text{guard}^i(\varphi \wedge \psi) &= \begin{cases} s(\varphi) \wedge \text{guard}^{i-\text{npred}(\varphi)}(\psi) & \text{if } i > \text{npred}(\varphi) \\ s(\psi) \wedge \text{guard}^i(\varphi) & \text{if } i \leq \text{npred}(\varphi) \end{cases} \\
\text{guard}^i(\varphi \vee \psi) &= \begin{cases} s(\neg\varphi) \wedge \text{guard}^{i-\text{npred}(\varphi)}(\psi) & \text{if } i > \text{npred}(\varphi) \\ s(\neg\psi) \wedge \text{guard}^i(\varphi) & \text{if } i \leq \text{npred}(\varphi) \end{cases} \\
\text{guard}^i(\forall d:D. \varphi) &= s(\forall d:D. \varphi) \wedge \text{guard}^i(\varphi) \\
\text{guard}^i(\exists d:D. \varphi) &= s(\neg\exists d:D. \varphi) \wedge \text{guard}^i(\varphi)
\end{aligned}$$

where

$$\begin{aligned}
s(\varphi) &= \varphi[i \mapsto \text{true}]_{i \leq \text{npred}(\varphi)} \\
s(\neg\varphi) &= \neg\varphi[i \mapsto \text{false}]_{i \leq \text{npred}(\varphi)}
\end{aligned}$$

The definition does include the case $\text{guard}^i(b)$, since it cannot occur due to $i \leq \text{npred}(\varphi)$. In [8], the function s was defined as $s(\varphi) = \varphi$ if $\text{npred}(\varphi) = 0$, and true otherwise. Furthermore, quantifiers were not taken into account. That results in guard yielding a weaker formula than our guard function does.

The next theorem shows that our function guard indeed yields guards and that these may be simultaneously applied to all PVI's in a predicate formula. For the proof of this theorem, we refer to [1, Theorem 7.38].

Theorem 3. *For all capture-avoiding formulae φ , it holds that*

$$\varphi \equiv \varphi[i \mapsto (\text{guard}^i(\varphi) \wedge \text{PVI}(\varphi, i))]_{i \leq \text{npred}(\varphi)}$$

Remark that the assumption that φ is capture-avoiding is necessary for the correctness of the guard function. Without this assumption, we may compute $\text{guard}^1((n \leq 2) \wedge \forall n:N. X(n)) = (n \leq 2)$, which is not a guard (see also Example 7). This predicate formula is also a counterexample to the correctness of [11, Lemma 6.27], which is the equivalent of Theorem 3 in the current work. Adding the assumption that φ is capture-avoiding resolves the issue.

6. Implementation

The proposed generalisations for constant elimination and guards have been implemented as an extension of the tool `pbescstelm`, which implements constant elimination and is part of the mCRL2 toolset [15], available through <https://mcl2.org>. The analysis of quantifiers can be enabled by the command line option `--check-quantifiers`. Before computing global propagated values with the fixpoint algorithm of Section 4, all guards and occurrences of QPVI's are first obtained by recursively traversing each right-hand side in the PBES. Furthermore, we

gather information on the free occurrences of variables, such that the effect of the quantifier-inside rewriter can be approximated. Hence, no traversal of the right-hand side (which can be very large) is required during the fixpoint computation itself.

To demonstrate a possible application of our ideas, we first perform a small experiment with a model of the *alternating bit protocol* (ABP) [16], where the data domain D has been restricted to only five elements¹. The transition system corresponding to this model has 182 states. We consider the property

$$\forall d:D. \nu W. ([\top]W \wedge \nu X. \mu Y. \nu Z. ([r(d)]X \wedge \langle r(d) \rangle true \Rightarrow [\overline{r(d)}]Y) \wedge [\overline{r(d)}]Z)$$

which expresses that whenever $r(d)$ is enabled infinitely often, then it also taken infinitely often, *i.e.*, it is treated fairly. This formula occurred earlier in [11]. Observe that the universally quantified value d does not occur meaningfully in the fixpoint W . We thus expect that the same quantifier in the corresponding PBES can be eliminated.

We take the PBES that encodes this formula on the ABP and instantiate it to a BES, which has 3641 nodes. Applying the original constant elimination algorithm from [9] on the PBES does not reduce this number. After applying our generalised algorithm that deals with quantifiers, the instantiated BES is reduced to 2913 nodes.

Our second experiment concerns the *cache coherence protocol* (CCP), as modelled in [17]. The instance we consider has two threads, two processes and one region. We slightly altered the specification, such that process identifiers are modelled with natural number instead of a dedicated finite sort; this does not change its behaviour. We consider the property that, if the system is stable, each region has no more copies than the number of processors minus one, formulated in [11] as

$$\begin{aligned} \forall procId:N. \neg(\mu X. \langle \top \rangle \vee (\langle c_copy \rangle true \wedge \langle lockempty(procId) \rangle true \\ \wedge \langle homequeueempty(procId) \rangle true \wedge \langle remotequeueempty(procId) \rangle true)) \end{aligned}$$

The corresponding PBES contains a QPVI of the shape $\forall procId:N. X(d, procId)$ and has an underlying BES of infinite size; the PBES can hence not be instantiated. However, after applying global propagation, the size of the BES is reduced to 50507 nodes. Global propagation also achieves a reduction when applied to the original specification, where the sort of variable $procId$ contains only two elements. In that case, the size of the BES is reduced from 101197 to 50507 nodes.

Experiments with several other model checking PBESs yielded no additional reduction over what is achieved by standard constant elimination. In nearly all cases, this is caused by the fact that the PBES contains quantifiers that cannot be sufficiently distributed over other operators. That is, in the computation of global propagated values, $qiocc(qi(\mathbf{Q}. \varphi_Y[e/d]))$ yields QPVI $\mathbf{Q}'. X(e')$ such that \mathbf{Q}' is empty. This is especially common when the μ -calculus formula used to construct the PBES mixes \forall and \exists or the diamond modality $\langle a \rangle$ (respectively \exists and \wedge or the box modality $[a]$). Note, however, that many formal properties of interest are linear (they can thus be encoded in LTL) and do not mix said operators.

¹The files used for the experiments are archived at <https://doi.org/10.5281/zenodo.6793890>

The effectiveness of our technique is also limited when quantifiers are accompanied by bounds. In some cases, the PBES contains a subformula of the shape $\forall d:D. f \wedge X(\mathbf{e})$ (respectively $\exists d:D. f \Rightarrow X(\mathbf{e})$), where f is an expression over d that restricts its domain. In other cases, such subformulae appear during the computation of $\text{qi}(\mathbf{Q}. \varphi_Y[\mathbf{e}/\mathbf{d}])$. Our technique cannot deal with either case, although a reduction might be possible in theory.

7. Conclusion

We saw that although quantifiers can prevent PBES instantiation, they have barely received attention in the literature on syntactic PBES transformations. Hence, we proposed to simplify PBESs based on global propagated values. Our experiment with global propagation indicates that it can indeed achieve a reduction of the state space. Furthermore, we identified an improvement for the computation of guards.

As discussed on Section 6, one of the main limitations of our technique is that we cannot deal with bounds on quantified variables. We believe the support of quantifier bounds will greatly benefit its practical applicability, so we plan to handle these cases in future work.

References

- [1] T. Neele, Reductions for Parity Games and Model Checking, Ph.D. thesis, Eindhoven University of Technology, 2020.
- [2] J. F. Groote, T. A. C. Willemse, Parameterised boolean equation systems, *Theoretical Computer Science* 343 (2005) 332–369. doi:10.1016/j.tcs.2005.06.016.
- [3] J. F. Groote, T. A. C. Willemse, Model-checking processes with data, *Science of Computer Programming* 56 (2005) 251–273. doi:10.1016/j.scico.2004.08.002.
- [4] T. Chen, B. Ploeger, J. van de Pol, T. A. C. Willemse, Equivalence Checking for Infinite Systems using Parameterized Boolean Equation Systems, in: *CONCUR 2007*, volume 4703 of *LNCS*, 2007, pp. 120–135. doi:10.1007/978-3-540-74407-8_9.
- [5] H. Seidl, Fast and simple nested fixpoints, *Information Processing Letters* 59 (1996) 303–308. doi:10.1016/0020-0190(96)00130-5.
- [6] B. Ploeger, J. W. Wesselink, T. A. C. Willemse, Verification of reactive systems via instantiation of Parameterised Boolean Equation Systems, *Information and Computation* 209 (2011) 637–663. doi:10.1016/j.ic.2010.11.025.
- [7] T. Neele, T. A. C. Willemse, J. F. Groote, Finding Compact Proofs for Infinite-Data Parameterised Boolean Equation Systems, *Science of Computer Programming* 188 (2020) 102389. doi:10.1016/j.scico.2019.102389.
- [8] J. J. A. Keiren, W. Wesselink, T. A. C. Willemse, Liveness Analysis for Parameterised Boolean Equation Systems, in: *ATVA 2014*, volume 8837 of *LNCS*, 2014, pp. 219–234. doi:10.1007/978-3-319-11936-6_16.
- [9] S. Orzan, W. Wesselink, T. A. C. Willemse, Static Analysis Techniques for Parameterised Boolean Equation Systems, in: *TACAS 2009*, volume 5505 of *LNCS*, 2009, pp. 230–245. doi:10.1007/978-3-642-00768-2_22.

- [10] S. Cranen, M. Gazda, W. Wesselink, T. A. C. Willemse, Abstraction in Fixpoint Logic, *ACM Transactions on Computational Logic* 16 (2015) 29:1–29:39. doi:10.1145/2740964.
- [11] J. J. A. Keiren, Advanced Reduction Techniques for Model Checking, Ph.D. thesis, Eindhoven University of Technology, 2013. doi:10.6100/IR757862.
- [12] S. Orzan, T. A. C. Willemse, Invariants for Parameterised Boolean Equation Systems, *Theoretical Computer Science* 411 (2010) 1338–1371. doi:10.1016/j.tcs.2009.11.001.
- [13] A. Mader, Modal μ -calculus, model checking and Gauß elimination, in: *TACAS 1995*, volume 1019 of *LNCS*, 1995, pp. 72–88. doi:10.1007/3-540-60630-0_4.
- [14] W. Zielonka, Infinite games on finitely coloured graphs with applications to automata on infinite trees, *Theoretical Computer Science* 200 (1998) 135–183. doi:10.1016/S0304-3975(98)00009-7.
- [15] O. Bunte, J. F. Groote, J. J. A. Keiren, M. Laveaux, T. Neele, E. P. de Vink, W. Wesselink, A. Wijs, T. A. C. Willemse, The mCRL2 Toolset for Analysing Concurrent Systems: Improvements in Expressivity and Usability, in: *TACAS 2019*, volume 11428 of *LNCS*, 2019, pp. 21–39. doi:10.1007/978-3-030-17465-1_2.
- [16] J. A. Bergstra, J. W. Klop, Verification of an alternating bit protocol by means of process algebra protocol, in: *MMSSS 1985*, volume 215 of *LNCS*, 1986, pp. 9–23. doi:10.1007/3-540-16444-8_1.
- [17] J. Pang, W. Fokkink, R. Hofman, R. Veldema, Model checking a cache coherence protocol of a Java DSM implementation, *Journal of Logic and Algebraic Programming* 71 (2007) 1–43. doi:10.1016/j.jlap.2006.08.007.