Liveness Analysis for Parameterised Boolean Equation Systems

Jeroen J. A. Keiren[1], Wieger Wesselink[2], and Tim A. C. Willemse[2]

[1] VU University Amsterdam, The Netherlands
`j.j.a.keiren@vu.nl`
[2] Eindhoven University of Technology, The Netherlands
`{j.w.wesselink, t.a.c.willemse}@tue.nl`

Abstract. We present a sound static analysis technique for fighting the combinatorial explosion of parameterised Boolean equation systems (PBESs). These essentially are systems of mutually recursive fixed point equations ranging over first-order logic formulae. Our method detects parameters that are not live by analysing a control flow graph of a PBES, and it subsequently eliminates such parameters. We show that a naive approach to constructing a control flow graph, needed for the analysis, may suffer from an exponential blow-up, and we define an approximate analysis that avoids this problem. The effectiveness of our techniques is evaluated using a number of case studies.

1 Introduction

Parameterised Boolean equation systems (PBESs) [7] are systems of fixpoint equations that range over first-order formulae; they are essentially an equational variation of *Least Fixpoint Logic (LFP)*. Fixpoint logics such as PBESs have applications in database theory and computer aided verification. For instance, the CADP [6] and mCRL2 [4] toolsets use PBESs for model checking and equivalence checking and in [2] PBESs are used to solve Datalog queries.

In practice, the predominant problem for PBESs is evaluating (henceforth referred to as *solving*) them so as to answer the decision problem encoded in them. There are a variety of techniques for solving PBESs, see [7], but the most straightforward method is by instantiation to a *Boolean equation system (BES)* [10], and then solving this BES. This process is similar to the explicit generation of a behavioural state space from its symbolic description, and it suffers from a combinatorial explosion that is akin to the state space explosion problem. Combatting this combinatorial explosion is therefore instrumental in speeding up the process of solving the problems encoded by PBESs.

While several static analysis techniques have been described using fixpoint logics, see *e.g.* [3], with the exception of the static analysis techniques for PBESs, described in [12], no such techniques seem to have been employed to simplify expressions in fixpoint logics.

Our main contribution in this paper is a static analysis method for PBESs that significantly improves over the aforementioned techniques for simplifying PBESs. In our method, we construct a *control flow graph* (CFG) for a given PBES and subsequently apply state space reduction techniques [5, 16], combined with liveness analysis techniques from compiler technology [1]. These typically scrutinise syntactic descriptions of behaviour to detect and eliminate variables that at some point become irrelevant (dead, not live) to the behaviour, thereby decreasing the complexity.

The notion of control flow of a PBES is not self-evident: formulae in fixpoint logics (such as PBESs) do not have a notion of a program counter. Our notion of control flow is based on the concept of *control flow parameters* (CFPs), which induce a CFG. Similar notions exist in the context of state space exploration, see *e.g.* [14], but so far, no such concept exists for fixpoint logics.

The size of the CFGs is potentially exponential in the number of CFPs. We therefore also describe a modification of our analysis—in which reductive power is traded against a lower complexity—that does not suffer from this problem. Our static analysis technique allows for solving PBESs using instantiation that hitherto could not be solved this way, either because the underlying BESs would be infinite or they would be extremely large. We show that our methods are sound; *i.e.*, simplifying PBESs using our analyses leads to PBESs with the same solution.

Our static analysis techniques have been implemented in the mCRL2 toolset [4] and applied to a set of model checking and equivalence checking problems. Our experiments show that the implementations outperform existing static analysis techniques for PBESs [12] in terms of reductive power, and that reductions of almost 100% of the size of the underlying BESs can be achieved. Our experiments confirm that the optimised version sometimes achieves slightly less reduction than our non-optimised version, but is faster. Furthermore, in cases where no additional reduction is achieved compared to existing techniques, the overhead is mostly negligible.

Structure of the paper. In Section 2 we give a cursory overview of basic PBES theory and in Section 3, we present an example to illustrate the difficulty of using instantiation to solve a PBES and to sketch our solution. In Section 4 we describe our construction of control flow graphs for PBESs and in Section 5 we describe our live parameter analysis. We present an optimisation of the analysis in Section 6. The approach is evaluated in Section 7, and Section 8 concludes. We refer to [9] for proofs and additional results.

2 Preliminaries

Throughout this paper, we work in a setting of *abstract data types* with non-empty data sorts $D_1, D_2, \ldots$, and operations on these sorts, and a set $\mathcal{D}$ of sorted data variables. We write vectors in boldface, *e.g.* $\boldsymbol{d}$ is used to denote a vector of data variables. We write $\boldsymbol{d}_i$ to denote the i-th element of a vector $\boldsymbol{d}$.

A semantic set $\mathbb{D}$ is associated to every sort D, such that each term of sort D, and all operations on D are mapped to the elements and operations of $\mathbb{D}$ they represent. *Ground terms* are terms that do not contain data variables. For terms that contain data variables, we use an environment δ that maps each variable from $\mathcal{D}$ to a value of the associated type. We assume an interpretation function $\llbracket \_ \rrbracket$ that maps every term t of sort D to the data element $\llbracket t \rrbracket \delta$ it represents, where the extensions of δ to open terms and vectors are standard. Environment updates are denoted $\delta[v/d]$, where $\delta[v/d](d') = v$ if $d' = d$, and $\delta(d')$ otherwise.

We specifically assume the existence of a sort B with elements $true$ and $false$ representing the Booleans $\mathbb{B}$ and a sort $N = \{0, 1, 2, \ldots\}$ representing the natural numbers $\mathbb{N}$. For these sorts, we assume that the usual operators are available and, for readability, these are written the same as their semantic counterparts.

Parameterised Boolean equation systems [11] are sequences of fixed-point equations ranging over *predicate formulae*. The latter are first-order formulae extended with predicate variables, in which the non-logical symbols are taken from the data language.

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

$$\varphi, \psi ::= b \mid X(\boldsymbol{e}) \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \forall d\colon D.\varphi \mid \exists d\colon D.\varphi$$

in which b is a data term of sort B, $X(\boldsymbol{e})$ is a predicate variable instance *(PVI) in which X is a predicate variable of sort $\boldsymbol{D} \to B$, taken from some sufficiently large set $\mathcal{P}$ of predicate variables, and $\mathbf{e}$ is a vector of data terms of sort $\boldsymbol{D}$. The interpretation of a predicate formula φ in the context of a predicate environment $\eta\colon \mathcal{P} \to \mathbb{D} \to \mathbb{B}$ and a data environment δ is denoted as $[\![\varphi]\!]\eta\delta$, where:*

$$[\![b]\!]\eta\delta = \begin{cases} \textit{true} & \textit{if } \delta(b) \\ \textit{false} & \textit{otherwise} \end{cases} \qquad [\![X(\boldsymbol{e})]\!]\eta\delta = \begin{cases} \textit{true} & \textit{if } \eta(X)(\delta(\boldsymbol{e})) \\ \textit{false} & \textit{otherwise} \end{cases}$$

$$[\![\phi \wedge \psi]\!]\eta\delta = [\![\phi]\!]\eta\delta \textit{ and } [\![\psi]\!]\eta\delta \textit{ hold} \qquad [\![\phi \vee \psi]\!]\eta\delta = [\![\phi]\!]\eta\delta \textit{ or } [\![\psi]\!]\eta\delta \textit{ hold}$$

$$[\![\forall d\colon D.\ \phi]\!]\eta\delta = \textit{for all } v \in \mathbb{D},\ [\![\phi]\!]\eta\delta[v/d] \textit{ holds}$$

$$[\![\exists d\colon D.\ \phi]\!]\eta\delta = \textit{for some } v \in \mathbb{D},\ [\![\phi]\!]\eta\delta[v/d] \textit{ holds}$$

We assume the usual precedence rules for the logical operators. *Logical equivalence* between two predicate formulae φ, ψ, denoted $\varphi \equiv \psi$, is defined as $[\![\varphi]\!]\eta\delta = [\![\psi]\!]\eta\delta$ for all η, δ. Freely occurring data variables in φ are denoted by $FV(\varphi)$. We refer to $X(\boldsymbol{e})$ occurring in a predicate formula as a *predicate variable instance* (PVI).

Definition 2. *PBESs are defined by the following grammar:*

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

in which $\emptyset$ denotes the empty equation system; μ and ν are the least and greatest fixed point signs, respectively; X is a sorted predicate variable of sort $\boldsymbol{D} \to B$, $\boldsymbol{d}$ is a vector of formal parameters, and φ is a predicate formula. We henceforth omit a trailing $\emptyset$.

By convention φ_X denotes the right-hand side of the defining equation for X in a PBES $\mathcal{E}$; $\mathsf{par}(X)$ denotes the set of *formal parameters* of X; and we assume that $FV(\varphi_X) \subseteq \mathsf{par}(X)$, and that $\mathsf{par}(X)$ is disjoint from the set of quantified variables. By superscripting a formal parameter with the predicate variable to which it belongs, we distinguish between formal parameters for different predicate variables, *i.e.*, we write d^X when $d \in \mathsf{par}(X)$. We write σ to stand for either μ or ν.

The set of *bound predicate variables* of some PBES $\mathcal{E}$, denoted $\mathsf{bnd}(\mathcal{E})$, is the set of predicate variables occurring at the left-hand sides of the equations in $\mathcal{E}$. Throughout this paper, we deal with PBESs that are both *well-formed*, *i.e.* for every $X \in \mathsf{bnd}(\mathcal{E})$ there is exactly one equation in $\mathcal{E}$, and *closed*, *i.e.* for every $X \in \mathsf{bnd}(\mathcal{E})$, only predicate variables taken from $\mathsf{bnd}(\mathcal{E})$ occur in φ_X.

To each PBES $\mathcal{E}$ we associate a *top assertion*, denoted $\mathbf{init}\ X(\boldsymbol{v})$, where we require $X \in \mathsf{bnd}(\mathcal{E})$. For a parameter $\boldsymbol{d}_m \in \mathsf{par}(X)$ for the top assertion $\mathbf{init}\ X(\boldsymbol{v})$ we define the value $\mathsf{init}(\boldsymbol{d}_m)$ as $\boldsymbol{v}_m$.

We next define a PBES's semantics. Let $\mathbb{B}^{\mathbb{D}}$ denote the set of functions $f\colon \mathbb{D} \to \mathbb{B}$, and define the ordering $\sqsubseteq$ as $f \sqsubseteq g$ iff for all $\boldsymbol{v} \in \mathbb{D}$, $f(\boldsymbol{v})$ implies $g(\boldsymbol{v})$. For a given pair of environments δ, η, a predicate formula φ gives rise to a predicate transformer T on the complete lattice $(\mathbb{B}^{\mathbb{D}}, \sqsubseteq)$ as follows: $T(f) = \lambda \boldsymbol{v} \in \mathbb{D}.[\![\varphi]\!]\eta[f/X]\delta[\boldsymbol{v}/\boldsymbol{d}]$.

Since the predicate transformers defined this way are monotone, their extremal fixed points exist. We denote the least fixed point of a given predicate transformer T by μT, and the greatest fixed point of T is denoted νT.

Definition 3. *The* solution *of an equation system in the context of a predicate environment η and data environment δ is defined inductively as follows:*

$$\begin{aligned}[\![\emptyset]\!]\eta\delta &= \eta\\ [\![(\mu X(\boldsymbol{d}\colon \boldsymbol{D}) = \varphi_X)\mathcal{E}]\!]\eta\delta &= [\![\mathcal{E}]\!]\eta[\mu T/X]\delta\\ [\![(\nu X(\boldsymbol{d}\colon \boldsymbol{D}) = \varphi_X)\mathcal{E}]\!]\eta\delta &= [\![\mathcal{E}]\!]\eta[\nu T/X]\delta\end{aligned}$$

with $T(f) = \lambda \boldsymbol{v} \in \mathbb{D}.[\![\varphi_X]\!]([\![\mathcal{E}]\!]\eta[f/X]\delta)\delta[\boldsymbol{v}/\boldsymbol{d}]$

The solution prioritises the fixed point signs of left-most equations over the fixed point signs of equations that follow, while respecting the equations. Bound predicate variables of closed PBESs have a solution that is independent of the predicate and data environments in which it is evaluated. We therefore omit these environments and write $[\![\mathcal{E}]\!](X)$ instead of $[\![\mathcal{E}]\!]\eta\delta(X)$.

3 A Motivating Example

In practice, solving PBESs proceeds via *instantiating* [13] into *Boolean equation systems (BESs)*, for which solving is decidable. The latter is the fragment of PBESs with equations that range over propositions only, *i.e.*, formulae without data and quantification. Instantiating a PBES to a BES is akin to state space exploration and suffers from a similar combinatorial explosion. Reducing the time spent on it is thus instrumental in speeding up, or even enabling the solving process. We illustrate this using the following (academic) example, which we also use as our running example:

$$\begin{aligned}&\nu X(i,j,k,l\colon N) = (i \neq 1 \vee j \neq 1 \vee X(2,j,k,l+1)) \wedge \forall m\colon N.Z(i,2,m+k,k)\\ &\mu Y(i,j,k,l\colon N) = k = 1 \vee (i = 2 \wedge X(1,j,k,l))\\ &\nu Z(i,j,k,l\colon N) = (k < 10 \vee j = 2) \wedge (j \neq 2 \vee Y(1,1,l,1)) \wedge Y(2,2,1,l)\end{aligned}$$

The presence of PVIs $X(2,j,k,l+1)$ and $Z(i,2,m+k,k)$ in X's equation means the solution to $X(1,1,1,1)$ depends on the solutions to $X(2,1,1,2)$ and $Z(1,2,v+1,1)$, for all values v, see Fig. 1. Instantiation finds these dependencies by simplifying the right-hand side of X when its parameters have been assigned value 1:

$$(1 \neq 1 \vee 1 \neq 1 \vee X(2,1,1,1+1)) \wedge \forall m\colon N.Z(1,2,m+1,1)$$

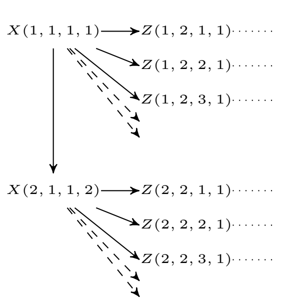


Fig. 1. Dependency graph

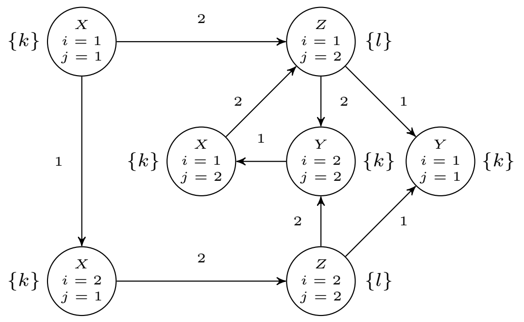


Fig. 2. Control flow graph for the running example

Since for an infinite number of different arguments the solution to Z must be computed, instantiation does not terminate. The problem is with the third parameter (k) of Z. We cannot simply assume that values assigned to the third parameter of Z do not matter; in fact, only when $j = 2$, Z's right-hand side predicate formula does not depend on k's value. This is where our developed method will come into play: it automatically determines that it is sound to replace PVI $Z(i, 2, m + k, k)$ by, *e.g.*, $Z(i, 2, 1, k)$ and to remove the universal quantifier, enabling us to solve $X(1, 1, 1, 1)$ using instantiation.

Our technique uses a *Control Flow Graph* (CFG) underlying the PBES for analysing which parameters of a PBES are *live*. The CFG is a finite abstraction of the dependency graph that would result from instantiating a PBES. For instance, when ignoring the third and fourth parameters in our example PBES, we find that the solution to $X(1, 1, *, *)$ depends on the first PVI, leading to $X(2, 1, *, *)$ and the second PVI in X's equation, leading to $Z(1, 2, *, *)$. In the same way we can determine the dependencies for $Z(1, 2, *, *)$, resulting in the finite structure depicted in Fig. 2. The subsequent liveness analysis annotates each vertex with a label indicating which parameters cannot (cheaply) be excluded from having an impact on the solution to the equation system; these are assumed to be live. Using these labels, we modify the PBES automatically.

Constructing a good CFG is a major difficulty, which we address in Section 4. The liveness analysis and the subsequent modification of the analysed PBES is described in Section 5. Since the CFG constructed in Section 4 can still suffer from a combinatorial explosion, we present an optimisation of our analysis in Section 6.

4 Constructing Control Flow Graphs for PBESs

The vertices in the control flow graph we constructed in the previous section represent the values assigned to a subset of the equations' formal parameters whereas an edge between two vertices captures the dependencies among (partially instantiated) equations. The better the control flow graph approximates the dependency graph resulting from an instantiation, the more precise the resulting liveness analysis.

Since computing a precise control flow graph is expensive, the problem is to compute the graph effectively and to balance precision and cost. To this end, we first identify a set of *control flow parameters*; the values to these parameters will make up the vertices in the control flow graph. While there is some choice for control flow parameters, we require that these are parameters for which we can *statically* determine:

1. the (finite set of) values these parameters can assume,
2. the set of PVIs on which the truth of a right-hand side predicate formula may depend, given a concrete value for each control flow parameter, and
3. the values assigned to the control flow parameters by all PVIs on which the truth of a right-hand side predicate formula may depend.

In addition to these requirements, we impose one other restriction: control flow parameters of one equation must be *mutually independent*; *i.e.*, we have to be able to determine their values independently of each other. Apart from being a natural requirement for a control flow parameter, it enables us to devise optimisations of our liveness analysis.

We now formalise these ideas. First, we characterise three partial functions that together allow to relate values of formal parameters to the dependency of a formula on a given PVI. Our formalisation of these partial functions is based on the following observation: if in a formula φ, we can replace a particular PVI $X(\boldsymbol{e})$ with the subformula $\psi \wedge X(\boldsymbol{e})$ without this affecting the truth value of φ, we know that φ's truth value only depends on $X(\boldsymbol{e})$'s whenever ψ holds. We will choose ψ such that it allows us to pinpoint exactly what value a formal parameter of an equation has (or will be assigned through a PVI). Using these functions, we then identify our control flow parameters by eliminating variables that do not meet all of the aforementioned requirements.

In order to reason about individual PVIs occurring in predicate formulae we introduce the notation necessary to do so. Let $\mathsf{npred}(\varphi)$ denote the number of PVIs occurring in a predicate formula φ. The function $\mathsf{PVI}(\varphi, i)$ is the formula representing the i^{th} PVI in φ, of which $\mathsf{pv}(\varphi, i)$ is the name and $\mathsf{arg}(\varphi, i)$ represents the term that appears as the argument of the instance. In general $\mathsf{arg}(\varphi, i)$ is a vector, of which we denote the j^{th} argument by $\mathsf{arg}_j(\varphi, i)$. Given predicate formula ψ we write $\varphi[i \mapsto \psi]$ to indicate that the PVI at position i is replaced syntactically by ψ in φ.

Definition 4. *Let $s\colon \mathcal{P} \times \mathbb{N} \times \mathbb{N} \to D$, $t\colon \mathcal{P} \times \mathbb{N} \times \mathbb{N} \to D$, and $c\colon \mathcal{P} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be partial functions, where D is the union of all ground terms. The triple (s, t, c) is a* unicity constraint *for PBES $\mathcal{E}$ if for all $X \in \mathsf{bnd}(\mathcal{E})$, $i, j, k \in \mathbb{N}$ and ground terms e:*

- *(source) if $s(X,i,j){=}e$ then $\varphi_X \equiv \varphi_X[i \mapsto (\boldsymbol{d}_j = e \wedge \mathsf{PVI}(\varphi_X, i))]$,*
- *(target) if $t(X,i,j){=}e$ then $\varphi_X \equiv \varphi_X[i \mapsto (\mathsf{arg}_j(\varphi_X, i) = e \wedge \mathsf{PVI}(\varphi_X, i))]$,*
- *(copy) if $c(X,i,j){=}k$ then $\varphi_X \equiv \varphi_X[i \mapsto (\mathsf{arg}_k(\varphi_X, i) = \boldsymbol{d}_j \wedge \mathsf{PVI}(\varphi_X, i))]$.*

Observe that indeed, function s states that, when defined, formal parameter $\boldsymbol{d}_j$ must have value $s(X, i, j)$ for φ_X's truth value to depend on that of $\mathsf{PVI}(\varphi_X, i)$. In the same vein $t(X, i, j)$, if defined, gives the fixed value of the j^{th} formal parameter of $\mathsf{pv}(\varphi_X, i)$. Whenever $c(X, i, j) = k$ the value of variable $\boldsymbol{d}_j$ is transparently copied to position k in the i^{th} predicate variable instance of φ_X. Since s, t and c are partial functions, we do not require them to be defined; we use $\perp$ to indicate this.

Example 1. A unicity constraint (s, t, c) for our running example could be one that assigns $s(X, 1, 2) = 1$, since parameter j^X must be 1 to make X's right-hand side formula depend on PVI $X(2, j, k, l + 1)$. We can set $t(X, 1, 2) = 1$, as one can deduce that parameter j^X is set to 1 by the PVI $X(2, j, k, l + 1)$; furthermore, we can set $c(Z, 1, 4) = 3$, as parameter k^Y is set to l^Z's value by PVI $Y(1, 1, l, 1)$.

From hereon, we assume that $\mathcal{E}$ is an arbitrary PBES with $(\mathsf{source}, \mathsf{target}, \mathsf{copy})$ a unicity constraint we can deduce for it. Notice that for each formal parameter for which either source or target is defined for some PVI, we have a finite set of values that this parameter can assume. However, at this point we do not yet know whether this set of values is exhaustive: it may be that some PVIs may cause the parameter to take on arbitrary values. Below, we will narrow down for which parameters we *can* ensure that the set of values is exhaustive. First, we eliminate formal parameters that do not meet conditions 1–3 for PVIs that induce self-dependencies for an equation.

Definition 5. *A parameter* $\boldsymbol{d}_n \in \mathsf{par}(X)$ *is a* local control flow parameter *(LCFP) if for all* i *such that* $\mathsf{pv}(\varphi_X, i) = X$*, either* $\mathsf{source}(X, i, n)$ *and* $\mathsf{target}(X, i, n)$ *are defined, or* $\mathsf{copy}(X, i, n) = n$*.*

Example 2. Formal parameter l^X in our running example does not meet the conditions of Def. 5 and is therefore not an LCFP. All other parameters in all other equations are still LCFPs since X is the only equation with a self-dependency.

From the formal parameters that are LCFPs, we next eliminate those parameters that do not meet conditions 1–3 for PVIs that induce dependencies among *different* equations.

Definition 6. *A parameter* $\boldsymbol{d}_n \in \mathsf{par}(X)$ *is a* global control flow parameter *(GCFP) if it is an LCFP, and for all* $Y \in \mathsf{bnd}(\mathcal{E}) \setminus \{X\}$ *and all* i *such that* $\mathsf{pv}(\varphi_Y, i) = X$*, either* $\mathsf{target}(Y, i, n)$ *is defined, or* $\mathsf{copy}(Y, i, m) = n$ *for some GCFP* $\boldsymbol{d}_m \in \mathsf{par}(Y)$*.*

The above definition is recursive in nature: if a parameter does not meet the GCFP conditions then this may result in another parameter also not meeting the GCFP conditions. Any set of parameters that meets the GCFP conditions is a good set, but larger sets possibly lead to better information about the control flow in a PBES.

Example 3. Formal parameter k^Z in our running example is not a GCFP since in PVI $Z(i, 2, m + k, k)$ from X's equation, the value assigned to k^Z cannot be determined.

The parameters that meet the GCFP conditions satisfy the conditions 1–3 that we imposed on control flow parameters: they assume a finite set of values, we can deduce which PVIs may affect the truth of a right-hand side predicate formula, and we can deduce how these parameters evolve as a result of all PVIs in a PBES. However, we may still have parameters of a given equation that are mutually dependent. Note that this dependency can only arise as a result of copying parameters: in all other cases, the functions source and target provide the information to deduce concrete values.

Example 4. GCFP k^Y affects GCFP k^X's value through PVI $X(1, j, k, l)$; likewise, k^X affects l^Z's value through PVI $Z(i, 2, m + k, k)$. Through the PVI $Y(2, 2, 1, l)$ in Z's equation, GCFP l^Z affects GCFPs l^Y value. Thus, k^Y affects l^Y's value transitively.

We identify parameters that, through copying, may become mutually dependent. To this end, we use a relation $\sim$, to indicate that GCFPs are *related*. Let $\boldsymbol{d}^{\boldsymbol{X}}_n$ and $\boldsymbol{d}^{\boldsymbol{Y}}_m$ be GCFPs; these are *related*, denoted $\boldsymbol{d}^{\boldsymbol{X}}_n \sim \boldsymbol{d}^{\boldsymbol{Y}}_m$, if $n = \mathsf{copy}(Y, i, m)$ for some i. Next, we characterise when a set of GCFPs does not introduce mutual dependencies.

Definition 7. *Let $\mathcal{C}$ be a set of GCFPs, and let $\sim^*$ denote the reflexive, symmetric and transitive closure of $\sim$ on $\mathcal{C}$. Assume $\approx \subseteq \mathcal{C} \times \mathcal{C}$ is an equivalence relation that subsumes $\sim^*$;* i.e.*, that satisfies $\sim^* \subseteq \approx$. Then the pair $\langle \mathcal{C}, \approx \rangle$ defines a* control structure *if for all $X \in \mathsf{bnd}(\mathcal{E})$ and all $d, d' \in \mathcal{C} \cap \mathsf{par}(X)$, if $d \approx d'$, then $d = d'$.*

We say that a unicity constraint is a *witness* to a control structure $\langle \mathcal{C}, \approx \rangle$ if the latter can be deduced from the unicity constraint through Definitions 5–7. The equivalence $\approx$ in a control structure also serves to identify GCFPs that take on the same role in *different* equations: we say that two parameters $c, c' \in \mathcal{C}$ are *identical* if $c \approx c'$. As a last step, we formally define our notion of a control flow parameter.

Definition 8. *A formal parameter c is a* control flow parameter (CFP) *if there is a control structure $\langle \mathcal{C}, \approx \rangle$ such that $c \in \mathcal{C}$.*

Example 5. There is a unicity constraint that identifies that parameter i^X is copied to i^Z in our running example. Then necessarily $i^Z \sim i^X$ and thus $i^X \approx i^Z$ for a control structure $\langle \mathcal{C}, \approx \rangle$ with $i^X, i^Z \in \mathcal{C}$. However, i^X and i^Y do not have to be related, but we have the option to define $\approx$ so that they are. The structure $\langle \{i^X, j^X, i^Y, j^Y, i^Z, j^Z\}, \approx \rangle$ for which $\approx$ relates all (and only) identically named parameters is a control structure.

Using a control structure $\langle \mathcal{C}, \approx \rangle$, we can ensure that all equations have the same set of CFPs. This can be done by assigning unique names to identical CFPs and by adding CFPs that do not appear in an equation as formal parameters for this equation. Without loss of generality we therefore continue to work under the following assumption.

Assumption 1 *The set of CFPs is the same for every equation in a PBES $\mathcal{E}$;* i.e.*, for all $X, Y \in \mathsf{bnd}(\mathcal{E})$, $d^X \in \mathsf{par}(X)$ is a CFP iff $d^Y \in \mathsf{par}(Y)$ is a CFP, and $d^X \approx d^Y$.*

From hereon, we call any formal parameter that is not a control flow parameter a *data parameter*. We make this distinction explicit by partitioning $\mathcal{D}$ into CFPs $\mathcal{C}$ and data parameters $\mathcal{D}^{DP}$. As a consequence of Assumption 1, we may assume that every PBES we consider has equations with the same sequence of CFPs; *i.e.*, all equations are of the form $\sigma X(\boldsymbol{c} \colon \boldsymbol{C}, \boldsymbol{d^X} : \boldsymbol{D^X}) = \varphi_X(\boldsymbol{c}, \boldsymbol{d^X})$, where $\boldsymbol{c}$ is the (vector of) CFPs, and $\boldsymbol{d^X}$ is the (vector of) data parameters of the equation for X.

Using the CFPs, we next construct a control flow graph. Vertices in this graph represent valuations for the vector of CFPs and the edges capture dependencies on PVIs. The set of potential valuations for the CFPs is bounded by $\mathsf{values}(\boldsymbol{c}_k)$, defined as:

$$\{\mathsf{init}(\boldsymbol{c}_k)\} \cup \bigcup_{i \in \mathbb{N}, X \in \mathsf{bnd}(\mathcal{E})} \{v \in D \mid \mathsf{source}(X, i, k) = v \vee \mathsf{target}(X, i, k) = v\}.$$

We generalise values to the vector $\boldsymbol{c}$ in the obvious way.

Definition 9. *The control flow graph (CFG) of $\mathcal{E}$ is a directed graph $(V, \to)$ with:*

- $V \subseteq \mathsf{bnd}(\mathcal{E}) \times \mathsf{values}(\boldsymbol{c})$.
- $\to \subseteq V \times \mathbb{N} \times V$ *is the least relation for which, whenever $(X, \boldsymbol{v}) \xrightarrow{i} (\mathsf{pv}(\varphi_X, i), \boldsymbol{w})$ then for every k either:*
  - $\mathsf{source}(X, i, k) = \boldsymbol{v}_k$ *and* $\mathsf{target}(X, i, k) = \boldsymbol{w}_k$*, or*

- $\mathsf{source}(X, i, k) = \bot$, $\mathsf{copy}(X, i, k) = k$ *and* $\boldsymbol{v}_k = \boldsymbol{w}_k$, *or*
- $\mathsf{source}(X, i, k) = \bot$, *and* $\mathsf{target}(X, i, k) = \boldsymbol{w}_k$.

We refer to the vertices in the CFG as *locations*. Note that a CFG is finite since the set $\mathsf{values}(\boldsymbol{c})$ is finite. Furthermore, CFGs are complete in the sense that all PVIs on which the truth of some φ_X may depend when $\boldsymbol{c} = \boldsymbol{v}$ are neighbours of location $(X, \boldsymbol{v})$.

Example 6. Using the CFPs identified earlier and an appropriate unicity constraint, we can obtain the CFG depicted in Fig. 2 for our running example.

Implementation. CFGs are defined in terms of CFPs, which in turn are obtained from a unicity constraint. Our definition of a unicity constraint is not constructive. However, a unicity constraint can be derived from *guards* for a PVI. Computing the exact guard, *i.e.* the strongest formula ψ satisfying $\varphi \equiv \varphi[i \mapsto (\psi \wedge \mathsf{PVI}(\varphi, i))]$, is computationally hard. We can efficiently approximate it such that $\varphi \equiv \varphi[i \mapsto (\mathsf{guard}^i(\varphi) \wedge \mathsf{PVI}(\varphi, i))]$; *i.e.*, $\mathsf{PVI}(\varphi, i)$ is relevant to φ's truth value only if $\mathsf{guard}^i(\varphi)$ is satisfiable, as follows:

Definition 10. *Let φ be a predicate formula. We define the* guard *of the i-th PVI in φ, denoted* $\mathsf{guard}^i(\varphi)$, *inductively as follows:*

$$\mathsf{guard}^i(b) = \mathit{false} \qquad \mathsf{guard}^i(Y) = \mathit{true}$$

$$\mathsf{guard}^i(\forall d\colon D.\varphi) = \mathsf{guard}^i(\varphi) \qquad \mathsf{guard}^i(\exists d\colon D.\varphi) = \mathsf{guard}^i(\varphi)$$

$$\mathsf{guard}^i(\varphi \wedge \psi) = \begin{cases} s(\varphi) \wedge \mathsf{guard}^{i-\mathsf{npred}(\varphi)}(\psi) & \textit{if } i > \mathsf{npred}(\varphi) \\ s(\psi) \wedge \mathsf{guard}^i(\varphi) & \textit{if } i \leq \mathsf{npred}(\varphi) \end{cases}$$

$$\mathsf{guard}^i(\varphi \vee \psi) = \begin{cases} s(\neg\varphi) \wedge \mathsf{guard}^{i-\mathsf{npred}(\varphi)}(\psi) & \textit{if } i > \mathsf{npred}(\varphi) \\ s(\neg\psi) \wedge \mathsf{guard}^i(\varphi) & \textit{if } i \leq \mathsf{npred}(\varphi) \end{cases}$$

where $s(\varphi) = \varphi$ *if* $\mathsf{npred}(\varphi) = 0$, *and true otherwise.*

Example 7. In the running example $\mathsf{guard}^1(\varphi_X) = \mathit{true} \wedge \neg(i \neq 1) \wedge \neg(j \neq 1) \wedge \mathit{true}$.

A good heuristic for defining the unicity constraints is looking for positive occurrences of constraints of the form $d = e$ in the guards and using this information to see if the arguments of PVIs reduce to constants.

5 Data Flow Analysis

Our liveness analysis is built on top of CFGs constructed using Def. 9. The analysis proceeds as follows: for each location in the CFG, we first identify the data parameters that may directly affect the truth value of the corresponding predicate formula. Then we inductively identify data parameters that can affect such parameters through PVIs as live as well. Upon termination, each location is labelled by the *live* parameters at that location. The set $\mathsf{sig}(\varphi)$ of parameters that affect the truth value of a predicate formula φ, *i.e.*, those parameters that occur in Boolean data terms, are approximated as follows:

$$\mathsf{sig}(b) = FV(b) \qquad \mathsf{sig}(Y(e)) = \emptyset$$

$$\mathsf{sig}(\varphi \wedge \psi) = \mathsf{sig}(\varphi) \cup \mathsf{sig}(\psi) \qquad \mathsf{sig}(\varphi \vee \psi) = \mathsf{sig}(\varphi) \cup \mathsf{sig}(\psi)$$

$$\mathsf{sig}(\exists d\colon D.\varphi) = \mathsf{sig}(\varphi) \setminus \{d\} \qquad \mathsf{sig}(\forall d\colon D.\varphi) = \mathsf{sig}(\varphi) \setminus \{d\}$$

Observe that $\mathsf{sig}(\varphi)$ is not invariant under logical equivalence. We use this fact to our advantage: we assume the existence of a function simplify for which we require $\mathsf{simplify}(\varphi) \equiv \varphi$, and $\mathsf{sig}(\mathsf{simplify}(\varphi)) \subseteq \mathsf{sig}(\varphi)$. An appropriately chosen function simplify may help to narrow down the parameters that affect the truth value of predicate formulae in our base case; in the worst case the function leaves φ unchanged. Labelling the CFG with live variables is achieved as follows:

Definition 11. *Let $\mathcal{E}$ be a PBES and let $(V, \to)$ be its CFG. The labelling $L\colon V \to \mathbb{P}(\mathcal{D}^{DP})$ is defined as $L(X, \boldsymbol{v}) = \bigcup_{n \in \mathbb{N}} L^n(X, \boldsymbol{v})$, with L^n inductively defined as:*

$$\begin{aligned}
L^0(X, \boldsymbol{v}) &= \mathsf{sig}(\mathsf{simplify}(\varphi_X[\boldsymbol{c} := \boldsymbol{v}])) \\
L^{n+1}(X, \boldsymbol{v}) &= L^n(X, \boldsymbol{v}) \cup \{d \in \mathsf{par}(X) \cap \mathcal{D}^{DP} \mid \exists i \in \mathbb{N}, (Y, \boldsymbol{w}) \in V : \\
&\qquad (X, \boldsymbol{v}) \xrightarrow{i} (Y, \boldsymbol{w}) \wedge \exists \boldsymbol{d}_\ell \in L^n(Y, \boldsymbol{w}) :\ d \in FV(\mathsf{arg}_\ell(\varphi_X, i))\}
\end{aligned}$$

The set $L(X, \boldsymbol{v})$ approximates the set of parameters potentially live at location $(X, \boldsymbol{v})$; all other data parameters are guaranteed to be "dead", *i.e.*, irrelevant.

Example 8. The labelling computed for our running example is depicted in Fig. 2. One can cheaply establish that $k^Z \notin L^0(Z, 1, 2)$ since assigning value 2 to j^Z in Z's right-hand side effectively allows to reduce subformula $(k < 10 \vee j = 2)$ to $true$. We have $l \in L^1(Z, 1, 2)$ since we have $k^Y \in L^0(Y, 1, 1)$.

A parameter d that is not live at a location can be assigned a fixed default value. To this end the corresponding data argument of the PVIs that lead to that location are replaced by a default value $\mathsf{init}(d)$. This is achieved by function Reset, defined below:

Definition 12. *Let $\mathcal{E}$ be a PBES, let $(V, \to)$ be its CFG, with labelling L. The PBES $\mathsf{Reset}_L(\mathcal{E})$ is obtained from $\mathcal{E}$ by replacing every PVI $X(\boldsymbol{e}, \boldsymbol{e'})$ in every φ_X of $\mathcal{E}$ by the formula $\bigwedge_{\boldsymbol{v} \in \mathsf{values}(\boldsymbol{c})} (\boldsymbol{v} \neq \boldsymbol{e} \vee X(\boldsymbol{e}, \mathsf{Reset}_L^{(X,\boldsymbol{v})}(\boldsymbol{e'})))$. The function $\mathsf{Reset}_L^{(X,\boldsymbol{v})}(\boldsymbol{e'})$ is defined positionally as follows:*

$$\textit{if } \boldsymbol{d}_i \in L(X, \boldsymbol{v}) \textit{ we set } \mathsf{Reset}_L^{(X,\boldsymbol{v})}(\boldsymbol{e'})_i = \boldsymbol{e'}_i, \ \textit{else } \mathsf{Reset}_L^{(X,\boldsymbol{v})}(\boldsymbol{e'})_i = \mathsf{init}(\boldsymbol{d}_i).$$

Resetting dead parameters preserves the solution of the PBES, as we claim below.

Theorem 1. *Let $\mathcal{E}$ be a PBES, and L a labelling. For all predicate variables X, and ground terms $\boldsymbol{v}$ and $\boldsymbol{w}$: $[\![\mathcal{E}]\!](X([\![\boldsymbol{v}]\!], [\![\boldsymbol{w}]\!])) = [\![\mathsf{Reset}_L(\mathcal{E})]\!](X([\![\boldsymbol{v}]\!], [\![\boldsymbol{w}]\!]))$.*

Proof sketch. We define a relation R^L such that $(X, [\![\boldsymbol{v}]\!], [\![\boldsymbol{w}]\!]) R^L (Y, [\![\boldsymbol{v'}]\!], [\![\boldsymbol{w'}]\!])$ if and only if $X = Y$, $[\![\boldsymbol{v}]\!] = [\![\boldsymbol{v'}]\!]$, and $\forall \boldsymbol{d}_k \in L(X, \boldsymbol{v}) : [\![\boldsymbol{w}_k]\!] = [\![\boldsymbol{w'}_k]\!]$. This relation is a *consistent correlation* [15]; the result then follows. See [9] for a detailed proof. □

As a consequence of the above theorem, instantiation of a PBES may become feasible where this was not the case for the original PBES. This is nicely illustrated by our running example, which now indeed can be instantiated to a BES.

Example 9. Observe that parameter k^Z is not labelled in any of the Z locations. This means that X's right-hand side essentially changes to:

$$\begin{gathered}
(i \neq 1 \vee j \neq 1 \vee X(2, j, k, l+1)) \wedge \\
\forall m\colon N.(i \neq 1 \vee Z(i, 2, 1, k)) \wedge \forall m\colon N.(i \neq 2 \vee Z(i, 2, 1, k))
\end{gathered}$$

Since variable m no longer occurs in the above formula, the quantifier can be eliminated. Applying the reset function on the entire PBES leads to a PBES that we *can* instantiate to a BES (in contrast to the original PBES), allowing us to compute that the solution to $X(1,1,1,1)$ is $true$. This BES has only 7 equations.

6 Optimisation

Constructing a CFG can suffer from a combinatorial explosion; *e.g.*, the size of the CFG underlying the following PBES is exponential in the number of detected CFPs.

$$\begin{array}{rcl} \nu X(i_1,\dots,i_n\colon B) & = & (i_1 \wedge X(\mathit{false},\dots,i_n)) \vee (\neg i_1 \wedge X(\mathit{true},\dots,i_n)) \vee \\ & & \cdots \vee (i_n \wedge X(i_1,\dots,\mathit{false})) \vee (\neg i_n \wedge X(i_1,\dots,\mathit{true})) \end{array}$$

In this section we develop an alternative to the analysis of the previous section which mitigates the combinatorial explosion but still yields sound results. The correctness of our alternative is based on the following proposition, which states that resetting using any labelling that approximates that of Def. 11 is sound.

Proposition 1. *Let, for given PBES $\mathcal{E}$, $(V,\to)$ be a CFG with labelling L, and let L' be a labelling such that $L(X,\boldsymbol{v}) \subseteq L'(X,\boldsymbol{v})$ for all $(X,\boldsymbol{v})$. Then for all $X, \boldsymbol{v}$ and $\boldsymbol{w}$: $[\![\mathcal{E}]\!](X([\![\boldsymbol{v}]\!],[\![\boldsymbol{w}]\!])) = [\![\mathsf{Reset}_{L'}(\mathcal{E})]\!](X([\![\boldsymbol{v}]\!],[\![\boldsymbol{w}]\!]))$*

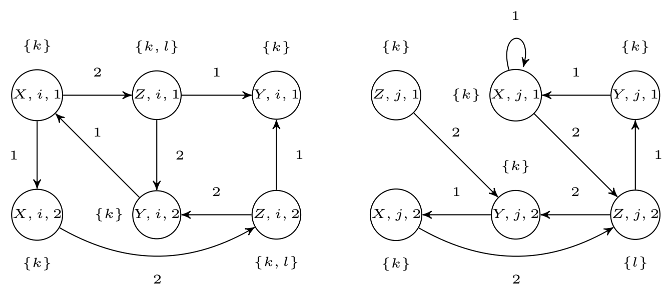
The idea is to analyse a CFG consisting of disjoint subgraphs for each individual CFP, where each subgraph captures which PVIs are under the control of a CFP: only if the CFP can confirm whether a predicate formula potentially depends on a PVI, there will be an edge in the graph. As before, let $\mathcal{E}$ be an arbitrary but fixed PBES, $(\mathsf{source},\mathsf{target},\mathsf{copy})$ a unicity constraint derived from $\mathcal{E}$, and $\boldsymbol{c}$ a vector of CFPs.

Definition 13. *The* local *control flow graph (LCFG) is a graph $(V^l, \hookrightarrow)$ with:*

- $V^l = \{(X,n,v) \mid X \in \mathsf{bnd}(\mathcal{E}) \wedge n \leq |\boldsymbol{c}| \wedge v \in \mathsf{values}(\boldsymbol{c}_n)\}$, *and*
- $\hookrightarrow \subseteq V^l \times \mathbb{N} \times V^l$ *is the least relation satisfying* $(X,n,v) \overset{i}{\hookrightarrow} (\mathsf{pv}(\varphi_X,i),n,w)$ *if:*
  - $\mathsf{source}(X,i,n) = v$ *and* $\mathsf{target}(X,i,n) = w$*, or*
  - $\mathsf{source}(X,i,n) = \bot$, $\mathsf{pv}(\varphi_X,i) \neq X$ *and* $\mathsf{target}(X,i,n) = w$*, or*
  - $\mathsf{source}(X,i,n) = \bot$, $\mathsf{pv}(\varphi_X,i) \neq X$ *and* $\mathsf{copy}(X,i,n) = n$ *and* $v = w$.

We write $(X,n,v)\overset{i}{\hookrightarrow}$ if there exists some (Y,m,w) such that $(X,n,v)\overset{i}{\hookrightarrow}(Y,m,w)$. Note that the size of an LCFG is $\mathcal{O}(|\mathsf{bnd}(\mathcal{E})| \times |\boldsymbol{c}| \times \max\{|\mathsf{values}(\boldsymbol{c}_k)| \mid 0 \leq k \leq |\boldsymbol{c}|\})$.

Example 10. For our running example, we obtain the following LCFG.



We next describe how to label the LCFG in such a way that the labelling meets the condition of Proposition 1, ensuring soundness of our liveness analysis. The idea of using LCFGs is that in practice, the use and alteration of a data parameter is entirely determined by a single CFP, and that only on "synchronisation points" of two CFPs (when the values of the two CFPs are such that they both confirm that a formula may depend on the same PVI) there is exchange of information in the data parameters.

We first formalise when a data parameter is involved in a recursion (*i.e.*, when the parameter may affect whether a formula depends on a PVI, or when a PVI may modify the data parameter through a self-dependency or uses it to change another parameter). Let $X \in \mathsf{bnd}(\mathcal{E})$ be an arbitrary bound predicate variable in the PBES $\mathcal{E}$.

Definition 14. *Denote* $\mathsf{PVI}(\varphi_X, i)$ *by* $Y(\boldsymbol{e})$*. Parameter* $\boldsymbol{d}_j \in \mathsf{par}(X)$ *is:*
- used for $Y(\boldsymbol{e})$ *if* $\boldsymbol{d}_j \in FV(\mathsf{guard}^i(\varphi_X))$*;*
- used in $Y(\boldsymbol{e})$ *if for some* k*, we have* $\boldsymbol{d}_j \in FV(\boldsymbol{e}_k)$*, (*$k \neq j$ *if* $X = Y$*) ;*
- changed *by* $Y(\boldsymbol{e})$ *if both* $X = Y$ *and* $\boldsymbol{d}_j \neq \boldsymbol{e}_j$*.*

Example 11. In the running example, k is used for $\mathsf{PVI}(\varphi_Y, 1)$. Parameter l is used in $\mathsf{PVI}(\varphi_Z, 1)$ and it is changed by $\mathsf{PVI}(\varphi_X, 1)$.

The PVIs using or modifying some data parameter constitute the parameter's dataflow. A data parameter *belongs to* a CFP if its complete dataflow is controlled by that CFP.

Definition 15. *CFP* $\boldsymbol{c}_j$ rules $\mathsf{PVI}(\varphi_X, i)$ *if* $(X, j, v) \overset{i}{\hookrightarrow}$ *for some* v*. Let* $d \in \mathsf{par}(X) \cap \mathcal{D}^{DP}$ *be a data parameter;* d belongs to $\boldsymbol{c}_j$ *if and only if:*
- *whenever* d *is used for* or *in* $\mathsf{PVI}(\varphi_X, i)$*,* $\boldsymbol{c}_j$ *rules* $\mathsf{PVI}(\varphi_X, i)$*, and*
- *whenever* d *is changed by* $\mathsf{PVI}(\varphi_X, i)$*,* $\boldsymbol{c}_j$ *rules* $\mathsf{PVI}(\varphi_X, i)$*.*

The set of data parameters that belong to $\boldsymbol{c}_j$ *is denoted by* $\mathsf{belongs}(\boldsymbol{c}_j)$*.*

Example 12. In all equations in our running example, k and l both belong to both i and j. Consider, *e.g.*, $\mathsf{PVI}(\varphi_X, 2)$, for which k is used; this PVI is ruled by i, which is witnessed by the edge $(X, i, 2) \overset{2}{\hookrightarrow} (Z, i, 2)$.

By adding dummy CFPs that can only take on one value, we can ensure that every data parameter belongs to at least one CFP. For simplicity and without loss of generality, we can therefore continue to work under the following assumption.

Assumption 2 *Each data parameter in an equation belongs to at least one CFP.*

We next describe how to conduct the liveness analysis using the LCFG. Every live data parameter is only labelled in those subgraphs corresponding to the CFPs to which it belongs. The labelling itself is constructed in much the same way as was done in the previous section. Our base case labels a vertex (X, n, v) with those parameters that belong to the CFP and that are significant in φ_X when $\boldsymbol{c}_n$ has value v. The backwards reachability now distinguishes two cases, based on whether the influence on live variables is internal to the CFP or via an external CFP.

Definition 16. *Let $(V^l, \hookrightarrow)$ be a LCFG for PBES $\mathcal{E}$. The labelling $L_l \colon V^l \to \mathbb{P}(\mathcal{D}^{DP})$ is defined as $L_l(X, n, v) = \bigcup_{k \in \mathbb{N}} L_l^k(X, n, v)$, with L_l^k inductively defined as:*

$$\begin{aligned}
L_l^0(X,n,v) &= \{d \in \mathsf{belongs}(\boldsymbol{c}_n) \mid d \in \mathsf{sig}(\mathsf{simplify}(\varphi_X[\boldsymbol{c}_n := v]))\}\\
L_l^{k+1}(X,n,v) &= L_l^k(X,n,v)\\
&\quad \cup\{d \in \mathsf{belongs}(\boldsymbol{c}_n) \mid \exists i, w \text{ such that } \exists \boldsymbol{d}_\ell^{\boldsymbol{Y}} \in L_l^k(Y,n,w) :\\
&\qquad (X,n,v) \overset{i}{\hookrightarrow} (Y,n,w) \wedge d \in FV(\mathsf{arg}_\ell(\varphi_X, i))\}\\
&\quad \cup\{d \in \mathsf{belongs}(\boldsymbol{c}_n) \mid \exists i, m, v', w' \text{ such that } (X,n,v) \overset{i}{\hookrightarrow}\\
&\qquad \wedge\ \exists \boldsymbol{d}_\ell^{\boldsymbol{Y}} \in L_l^k(Y,m,w') : \boldsymbol{d}_\ell^{\boldsymbol{Y}} \notin \mathsf{belongs}(\boldsymbol{c}_n)\\
&\qquad \wedge\ (X,m,v') \overset{i}{\hookrightarrow} (Y,m,w') \wedge d \in FV(\mathsf{arg}_\ell(\varphi_X, i))\}
\end{aligned}$$

Example 13. In the LCFG for the running example, initially k occurs in the labelling of all Y and Z nodes, except for $(Z, j, 2)$, since when $j = 2$, $k < 10 \vee j = 2$ is satisfied regardless of the value of k. The final labelling is shown in the graph in Example 10.

On top of this labelling we define the induced labelling $L_l(X, \boldsymbol{v})$, defined as $d \in L_l(X, \boldsymbol{v})$ iff for all k for which $d \in \mathsf{belongs}(\boldsymbol{c}_k)$ we have $d \in L_l(X, k, \boldsymbol{v}_k)$. This labelling over-approximates the labelling of Def. 11; *i.e.*, we have $L(X, \boldsymbol{v}) \subseteq L_l(X, \boldsymbol{v})$ for all $(X, \boldsymbol{v})$. The induced labelling L_l can remain implicit; in an implementation, the labelling constructed by Def. 16 can be used directly, sidestepping a combinatorial explosion. Combined with Prop. 1, this leads to the following theorem.

Theorem 2. *We have $[\![\mathcal{E}]\!](X([\![\boldsymbol{v}]\!], [\![\boldsymbol{w}]\!])) = [\![\mathsf{Reset}_{L_l}(\mathcal{E})]\!](X([\![\boldsymbol{v}]\!], [\![\boldsymbol{w}]\!]))$ for all predicate variables X and ground terms $\boldsymbol{v}$ and $\boldsymbol{w}$.*

Example 14. Using the labelling from Example 13, we obtain $L_l(Z, v, 2) = \{l\}$ for $v \in \{1, 2\}$, and $L_l(X, \boldsymbol{v}) = \{k\}$ for all other $X, \boldsymbol{v}$. Observe that, for the reachable part shown in Figure 2, this coincides with the labelling obtained using the global algorithm. For the example, this analysis thus yields the same reduction as the analysis in Section 5.

7 Case Studies

We implemented our techniques in the tool `pbesstategraph` of the mCRL2 toolset [4]. Here, we report on the tool's effectiveness in simplifying the PBESs originating from model checking problems and behavioural equivalence checking problems: we compare sizes of the BESs underlying the original PBESs to those for the PBESs obtained after running the tool `pbesparelm` (implementing the techniques from [12]) and those for the PBESs obtained after running our tool. Furthermore, we compare the total times needed for reducing the PBES, instantiating it into a BES, and solving this BES.

Our cases are taken from the literature. We here present a selection of the results. For the model checking problems, we considered the *Onebit* protocol, which is a complex sliding window protocol, and Hesselink's handshake register [8]. Both protocols are parametric in the set of values that can be read and written. A selection of properties of varying complexity and varying nesting degree, expressed in the data-enhanced modal μ-calculus are checked.[3] For the behavioural equivalence checking problems, we

[3] The formulae are contained in [9]; here we use textual characterisations instead.

Table 1. Sizes of the BESs underlying (1) the original PBESs, and the reduced PBESs using (2) `pbesparelm`, (3) `pbesstategraph` (global) and (4) `pbesstategraph` (local). For the original PBES, we report the number of generated BES equations, and the time required for generating and solving the resulting BES. For the other PBESs, we state the total reduction in percentages (*i.e.*, $100 * (|original| - |reduced|)/|original|$), and the reduction of the times (in percentages, computed in the same way), where for times we additionally include the `pbesstategraph/parelm` running times. Verdict $\surd$ indicates the problem has solution $true$; $\times$ indicates it is $false$.

		Sizes				Times				Verdict
	$\lvert D\rvert$	Original	`parelm`	`st.graph (global)`	`st.graph (local)`	Original	`parelm`	`st.graph (global)`	`st.graph (local)`	
Model Checking Problems										
No deadlock										
Onebit	2	81,921	86%	89%	89%	15.7	90%	85%	90%	$\surd$
	4	742,401	98%	99%	99%	188.5	99%	99%	99%	$\surd$
Hesselink	2	540,737	100%	100%	100%	64.9	99%	95%	99%	$\surd$
	3	13,834,801	100%	100%	100%	2776.3	100%	100%	100%	$\surd$
No spontaneous generation of messages										
Onebit	2	185,089	83%	88%	88%	36.4	87%	85%	88%	$\surd$
	4	5,588,481	98%	99%	99%	1178.4	99%	99%	99%	$\surd$
Messages that are read are inevitably sent										
Onebit	2	153,985	63%	73%	73%	30.8	70%	62%	73%	$\times$
	4	1,549,057	88%	92%	92%	369.6	89%	90%	92%	$\times$
Messages can overtake one another										
Onebit	2	164,353	63%	73%	70%	36.4	70%	67%	79%	$\times$
	4	1,735,681	88%	92%	90%	332.0	88%	88%	90%	$\times$
Values written to the register can be read										
Hesselink	2	1,093,761	1%	92%	92%	132.8	-3%	90%	91%	$\surd$
	3	27,876,961	1%	98%	98%	5362.9	25%	98%	99%	$\surd$
Equivalence Checking Problems										
Branching bisimulation equivalence										
ABP-CABP	2	31,265	0%	3%	0%	3.9	-4%	-1880%	-167%	$\surd$
	4	73,665	0%	5%	0%	8.7	-7%	-1410%	-72%	$\surd$
Buf-Onebit	2	844,033	16%	23%	23%	112.1	30%	28%	31%	$\surd$
	4	8,754,689	32%	44%	44%	1344.6	35%	44%	37%	$\surd$
Hesselink I-S	2	21,062,529	0%	93%	93%	4133.6	0%	74%	91%	$\times$
Weak bisimulation equivalence										
ABP-CABP	2	50,713	2%	6%	2%	5.3	2%	-1338%	-136%	$\surd$
	4	117,337	3%	10%	3%	13.0	4%	-862%	-75%	$\surd$
Buf-Onebit	2	966,897	27%	33%	33%	111.6	20%	29%	28%	$\surd$
	4	9,868,225	41%	51%	51%	1531.1	34%	49%	52%	$\surd$
Hesselink I-S	2	29,868,273	4%	93%	93%	5171.7	7%	79%	94%	$\times$

considered a number of communication protocols such as the *Alternating Bit Protocol* (ABP), the *Concurrent Alternating Bit Protocol* (CABP), a two-place buffer (Buf) and the aforementioned Onebit protocol. Moreover, we compare an implementation of Hesselink's register to a specification of the protocol that is correct with respect to trace equivalence (but for which currently no PBES encoding exists) but not with respect to the two types of behavioural equivalence checking problems we consider here: branching bisimilarity and weak bisimilarity.

The experiments were performed on a 64-bit Linux machine with kernel version 2.6.27, consisting of 14 Intel® Xeon© E5520 Processors running at 2.27GHz, and 1TB of shared main memory. In this system 7 servers are aggregated to appear as a single machine using vSMP software (each node has 2 CPUs and 144GB of main memory). In our experiments we run up to 24 tools simultaneously, but within each tool no multi-core features are used. We used revision 12637 of the mCRL2 toolset.[4]

The results are reported in Table 1; higher percentages mean better reductions/-smaller runtimes. The experiments confirm our technique can achieve as much as an additional reduction of about 97% over `pbesparelm`, see the model checking and equivalence problems for Hesselink's register. Compared to the sizes of the BESs underlying the original PBESs, the reductions can be immense. Furthermore, reducing the PBES using the local stategraph algorithm, instantiating, and subsequently solving it is typically faster than using the global stategraph algorithm, even when the reduction achieved by the first is less. For the equivalence checking cases, when no reduction is achieved the local version of stategraph sometimes results in substantially larger running times than parelm, which in turn already adds an overhead compared to the original; however, for the cases in which this happens the original running time is around or below 10 seconds, so the observed increase may be due to inaccuracies in measuring.

8 Conclusions and Future Work

We described a static analysis technique for PBESs that uses a notion of control flow to determine when data parameters become irrelevant. Using this information, the PBES can be simplified, leading to smaller underlying BESs. Our static analysis technique enables the solving of PBESs using instantiation that so far could not be solved this way as shown by our running example. Compared to existing techniques, our new static analysis technique can lead to additional reductions of up-to 97% in practical cases, as illustrated by our experiments. Furthermore, if a reduction can be achieved the technique can significantly speed up instantiation and solving, and in case no reduction is possible, it typically does not negatively impact the total running time.

Several techniques described in this paper can be used to enhance existing reduction techniques for PBESs. For instance, our notion of a *guard* of a predicate variable instance in a PBES can be put to use to cheaply improve on the heuristics for constant elimination [12]. Moreover, we believe that our (re)construction of control flow graphs from PBESs can be used to automatically generate invariants for PBESs. The theory on invariants for PBESs is well-established, but still lacks proper tool support.

Acknowledgments

The authors would like to thank Wan Fokkink and Sjoerd Cranen for remarks on earlier drafts of this paper that greatly helped improving the presentation of the material.

[4] The complete scripts for our test setup are available at `https://github.com/jkeiren/pbesstategraph-experiments`.

References

1. A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.
2. M. Alpuente, M. A. Feliú, C. Joubert, and A. Villanueva. Datalog-based program analysis with bes and rwl. In *Datalog*, volume 6702 of *LNCS*, pages 1–20. Springer, 2011.
3. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, POPL '77, page 238252, New York, NY, USA, 1977. ACM.
4. S. Cranen, J. F. Groote, J. J. A. Keiren, F. P. M. Stappers, E. P. de Vink, W. Wesselink, and T. A. C. Willemse. An overview of the mCRL2 toolset and its recent advances. In *TACAS*, volume 7795 of *LNCS*, pages 199–213. Springer, 2013.
5. J.-C. Fernandez, M. Bozga, and L. Ghirvu. State space reduction based on live variables analysis. *Science of Computer Programming*, 47(2-3):203–220, 2003.
6. H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2010: A toolbox for the construction and analysis of distributed processes. In Parosh Abdulla and K Leino, editors, *TACAS*, volume 6605 of *LNCS*, page 372387. Springer, 2011.
7. J. F. Groote and T. A. C. Willemse. Parameterised boolean equation systems. *Theoretical Computer Science*, 343(3):332–369, 2005.
8. W. H. Hesselink. Invariants for the construction of a handshake register. *Information Processing Letters*, 68:173–177, 1998.
9. J. J. A. Keiren, J. W. Wesselink, and T. A. C. Willemse. Improved static analysis of parameterised boolean equation systems using control flow reconstruction, 2013. arXiv:1304.6482 [cs.LO].
10. A. Mader. *Verification of Modal Properties Using Boolean Equation Systems*. PhD thesis, Technische Universität München, 1997.
11. R. Mateescu. *Vérification des propriétés temporelles des programmes parallèles*. PhD thesis, Institut National Polytechnique de Grenoble, 1998.
12. S. Orzan, J. W. Wesselink, and T. A. C. Willemse. Static Analysis Techniques for Parameterised Boolean Equation Systems. In *TACAS'09*, volume 5505 of *LNCS*, pages 230–245. Springer, 2009.
13. B. Ploeger, W. Wesselink, and T.A.C. Willemse. Verification of reactive systems via instantiation of parameterised Boolean equation systems. *Information and Computation*, 209(4):637–663, 2011.
14. J. C. van de Pol and M. Timmer. State Space Reduction of Linear Processes. In *ATVA*, volume 5799 of *LNCS*, pages 54–68. Springer, 2009.
15. T. A. C. Willemse. Consistent Correlations for Parameterised Boolean Equation Systems with Applications in Correctness Proofs for Manipulations. In *CONCUR 2010*, volume 6269 of *LNCS*, pages 584–598. Springer, 2010.
16. K. Yorav and O. Grumberg. Static Analysis for State-Space Reductions Preserving Temporal Logics. *Formal Methods in System Design*, 25(1):67–96, 2004.