Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 343 (2005) 332–369

Theoretical Computer Science

www.elsevier.com/locate/tcs

Parameterised boolean equation systems

Jan Friso Groote[a,*], Tim A.C. Willemse[a,b]

[a]*Department of Mathematics and Computer Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands*

[b]*Faculty of Science, Mathematics and Computing Science, University of Nijmegen, P.O. Box 9010, 6500 GL Nijmegen, The Netherlands*

Abstract

Boolean equation system are a useful tool for verifying formulas from modal μ-calculus on transition systems (see [Mader, Lecture Notes in Computer Science, Vol. 1019, 1995, pp. 72–88] for an excellent treatment). We are interested in an extension of boolean equation systems with data. This allows to formulate and prove a substantially wider range of properties on much larger and even infinite state systems. In previous works [Groote and Mateescu, Lecture Notes in Computer Science, Vol. 1548, 1999, pp. 74–90; Groote and Willemse, Sci. Comput. Program., 2005] it has been outlined how to transform a modal formula and a process, both containing data, to a so-called parameterised boolean equation system, or equation system for short. In this article we focus on techniques to solve such equation systems.

We introduce a new equivalence between equation systems, because existing equivalences are not compositional. We present techniques similar to Gauß elimination as outlined in [Mader, Lecture Notes in Computer Science, Vol. 1019, 1995, pp. 72–88] that allow to solve each equation system provided a single equation can be solved. We give several techniques for solving single equations, such as approximation (known), patterns (new) and invariants (new). Finally, we provide several small but illustrative examples of verifications of modal μ-calculus formulas on concrete processes to show the use of the techniques.
© 2005 Elsevier B.V. All rights reserved.

Keywords: First order modal μ-calculus; Parameterised boolean equation systems; Model checking; Infinite state systems

* Corresponding author.
E-mail address: J.F.Groote@tue.nl (J.F. Groote).

0304-3975/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2005.06.016

1. Introduction

Boolean Equation Systems (BESs) [20,21,25] are systems of the form $(\sigma_1 X_1 = f_1) \ldots (\sigma_N X_N = f_N)$, where σ_i is either a least fixpoint symbol μ or a greatest fixpoint symbol ν and f_i is a propositional formula. These systems can be seen as generalisations of nested and alternating fixpoint expressions, interpreted over a Boolean lattice.

BESs have been studied in detail by Vergauwen and Lewi [25], and Mader [20,21] in the context of model checking modal μ-calculus formulae. In [21], Mader shows that the model checking problem can be solved by solving BESs. Furthermore, she provides a complete proof system for solving BESs by means of algebraic manipulations.

Parameterised boolean equation systems (PBESs) (also known as *First-Order* Boolean Equation Systems) [11,15,26] are sequences of equations of the form $\sigma X(d_1{:}D_1, \ldots, d_n{:}D_n) = \varphi$, where σ is either a least or a greatest fixpoint symbol, d_i is a data variable of sort D_i and φ is a predicate formula. The sort $D_1 \times \cdots \times D_n$ is referred to as the *parameter-space* of a parameterised boolean equation.

PBESs form an extension of plain BESs. Groote and Mateescu [11] introduced these PBESs as an intermediate formalism for model checking processes with (arbitrary) data. Extending on the results of Mader [20,21], they showed that their model checking problem could be translated to the problem of solving PBESs. In [11], they provided four proof rules for approximating the solution of single parameterised equations: two for the least fixpoint and two for the greatest fixpoint. Furthermore, as a proof of concept, we showed in [15,26] that PBESs can be solved automatically by means of a technique that combines the essentials of Gauß-elimination [20,21], and approximation (see e.g. [10]).

While the automated approach has proved successful for several practical applications, it also illustrates the undecidability of model checking when no restrictions on the involved data-types are made, by occasionally requiring transfinite approximations of fixpoint expressions (i.e., in such cases, approximation procedures do not terminate). The emphasis on automation set a scene where possible remedies for such situations were hard to find.

Inspired by this latter observation, we take a different approach altogether in this paper, and focus on algebraic techniques that help in solving PBESs by hand. While this may seem a step back to some, being able to solve PBESs by hand provides a better understanding of the techniques that are involved. We intentionally proved many properties about systems by hand, some of which can be found in the second part of this paper, with as primary goal to build up experience and skill. As expected this led to effective techniques to manually solve parameterised boolean equation systems which are reported in the first part of this paper. Although it is not the focus of this paper, we expect that these techniques will also have a positive impact on the mechanised and automatic verification of modal formulas on processes in a setting with data.

The approach we describe in this paper is similar in spirit to the algebraic approach for solving BESs, taken by Mader [21]. We separate the problems of solving PBESs as a whole, and parameterised boolean equations in isolation. Central to our approach is the notion of a *system equivalence* that allows us to reason compositionally about PBESs. While in [21], also a system equivalence is introduced for BESs, it turns out that this equivalence is not compositional. We illustrate this fact by a simple example in Section 3. Together with system equivalence we introduce system ordering which on several occasions turns out to be an indispensable tool.

Based on our new notion of system equivalence, we present an overall and complete technique, allowing to solve all PBESs using syntactic manipulations only, provided the means to solve a single parameterised boolean equation in isolation are available (Section 4.1).

In Section 4.2 we investigate various techniques for solving a single parameterised boolean equation. These include a theorem allowing logical reasoning using predicate calculus and a result allowing to transfer results obtained using parameterised boolean equations to predicate logic. We proceed by restating results on approximation from [11] in terms of the new system equivalence.

Some of the parameterised boolean equation systems that we encountered were not easily solved using for instance approximation. But we noticed that many of these had a very similar pattern. For some of the most general patterns we could give a standard solution. We present this result in Section 4.2.3. We, however, believe that we have only scratched this topic on the surface. We expect a situation comparable to solving differential equations, where identifying and solving differential equations of a particular form has become a field of its own. There have been a number of typical parameterised boolean equations that we have not been able to solve and that deserve a separate investigation.

While invariants are an effective tool in diverse areas, such as process algebras [3] and program analysis [9], they have not yet been connected to BESs and PBESs. So, we set out to find their counterpart in parameterised boolean equations. We provide a definition and two theorems to ease their use in concrete situations. Our notion of an invariant in equation systems plays a very helpful role in many of the examples in Section 5 and so we believe that it will become a similarly effective tool as invariants are elsewhere.

The structure of this paper is as follows. Section 2 introduces the terminology used throughout this paper, together with a short overview of PBESs, their semantics and several smaller results. In Section 3 an equivalence for PBESs is introduced and compared against the equivalence for BESs that can be found in the literature. Section 4 then focuses on solving PBESs globally and parameterised boolean equations in isolation. As an illustration of these techniques, we apply these to several smaller examples in Section 5. Concluding remarks are presented in Section 6.

An extended abstract of this paper appeared as [16], and presents some of the main results of our investigations into PBESs. In addition to the results described in [16], this paper presents lemmata and theorems that are at the basis of those results and provides detailed proofs for them. This provides a better understanding of the presented techniques and foundations. We provide a more extensive treatment of approximation, invariants and on the use of predicate calculus for PBESs. In addition to the three examples that appeared in [16], this paper includes extra examples to illustrate techniques that where not applicable to the examples of [16].

2. Definition of a parameterised boolean equation system

We are interested in solving sequences of fixpoint equations where the equations have the form

$$\mu X(d_1{:}D_1, \ldots, d_n{:}D_n) = \varphi,$$

where μ indicates a minimal fixpoint, or

$$\nu X(d_1{:}D_1, \dots, d_n{:}D_n) = \varphi,$$

where ν indicates that this is a maximal fixpoint equation.

Each equation has a predicate variable X (from a set $\mathcal{X}$ of variables) at its left-hand side that depends on zero or more data variables $d_1, \dots, d_n$ of sorts $D_1, \dots, D_n$. For simplicity and without loss of generality, we restrict ourselves to a single variable at the left-hand side in all our theoretical considerations. We treat data in an abstract way. So, we assume that there are nonempty data sorts, generally written using letters D, E, F, that include the sort $\mathbb{B}$ of booleans containing $\bot$ and $\top$, representing *false* and *true*, respectively. We have a set $\mathcal{D}$ of data variables, with typical elements $d, d_1, \dots,$ and we assume that there is some data language that is sufficiently rich to denote all relevant data terms, such as for instance $3 + d_1 \leqslant d_2$. For a closed term e, we assume an interpretation function $[\![e]\!]$ that maps e to the data element it represents. For open terms we use a *data environment* ε that maps each variable from $\mathcal{D}$ to a data value of the right sort. The interpretation of an open term e of sort $\mathbb{B}$, denoted as $[\![e]\!]\varepsilon$ is given by $[\![\varepsilon(e)]\!]$ where ε is extended to terms in the standard way.

The right-hand side of each equation is a *predicate formula* containing data terms, boolean connectives, quantifiers over (possibly infinite) data domains and data and predicate variables. Predicate formulae φ are defined by the following grammar:

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

where b is a data term of sort $\mathbb{B}$, X is a predicate variable, d is a data variable of sort D and e is a data term. Note that negation does not occur in predicate formulae, except as an operator in data terms.

In the sequel it turns out to be necessary to lift predicate formulas to functions from data to formulas. We use conventional typed lambda calculus to denote such functions. For example $\lambda d{:}D.\varphi$ denotes a function from elements from data sort D to predicates. Sometimes, the lambda is omitted if that leads to a more compact notation. For instance $\lambda d{:}D.X(d)$ is generally simply written as X.

Predicate formulae are interpreted in a context of a data environment ε and a *predicate environment* $\eta{:}\mathcal{X} \to (D \to \mathbb{B})$. The semantics of predicate formulae is defined below. For an arbitrary environment θ (be it a data environment or predicate environment), we write $\theta[v/d]$ for the environment θ in which the variable d has been assigned the value v. For a predicate formula φ, a predicate environment η and a data environment ε, we write $\varphi(\eta\varepsilon)$, denoting the formula φ in which all free predicate variables X have received the value $\eta(X)$ and all free data variables d have received the value $\varepsilon(d)$. Environments are applied to functions, where bound variables are respected.

Definition 1 (*Semantics of predicate formulae*). Let ε be a data environment and $\eta{:}\mathcal{X} \to (D \to \mathbb{B})$ be a predicate environment. The *interpretation* $[\![\varphi]\!]\eta\varepsilon$ maps a predicate formula φ to “true” or “false” and is inductively defined as follows:

$$
\begin{aligned}
&[\![\top]\!]\eta\varepsilon && \stackrel{\text{def}}{=} \text{true},\\
&[\![\bot]\!]\eta\varepsilon && \stackrel{\text{def}}{=} \text{false},\\
&[\![b]\!]\eta\varepsilon && \stackrel{\text{def}}{=} [\![b]\!]\varepsilon,\\
&[\![X(e)]\!]\eta\varepsilon, && \stackrel{\text{def}}{=} \eta(X)([\![e]\!]\varepsilon),\\
&[\![\varphi_1 \wedge \varphi_2]\!]\eta\varepsilon && \stackrel{\text{def}}{=} [\![\varphi_1]\!]\eta\varepsilon \text{ and } [\![\varphi_2]\!]\eta\varepsilon,\\
&[\![\varphi_1 \vee \varphi_2]\!]\eta\varepsilon && \stackrel{\text{def}}{=} [\![\varphi_1]\!]\eta\varepsilon \text{ or } [\![\varphi_2]\!]\eta\varepsilon,\\
&[\![\forall d{:}D.\varphi]\!]\eta\varepsilon && \stackrel{\text{def}}{=} \begin{cases} \text{true, if for all } v{:}D \text{ it holds that } [\![\varphi]\!]\eta(\varepsilon[v/d]),\\ \text{false, otherwise,}\end{cases}\\
&[\![\exists d{:}D.\varphi]\!]\eta\varepsilon && \stackrel{\text{def}}{=} \begin{cases} \text{true, if there exists a } v{:}D \text{ such that } [\![\varphi]\!]\eta(\varepsilon[v/d]),\\ \text{false, otherwise.}\end{cases}
\end{aligned}
$$

Consider for an arbitrary data sort D, all (total) functions $f{:}D\to\mathbb{B}$. The set of all such functions is denoted $[D\to\mathbb{B}]$. The ordering $\sqsubseteq$ on $[D\to\mathbb{B}]$ is defined as $f \sqsubseteq g$ iff for all $d{:}D$, we have $f(d)$ implies $g(d)$. The set $([D\to\mathbb{B}], \sqsubseteq)$ is a complete lattice. For a subset A of $[D\to\mathbb{B}]$, we write $(\bigwedge A)$ for the *infimum* of the set A and $(\bigvee A)$ for the *supremum* of the set A.

We denote the set of all predicate environments by $[\mathcal{X}\to(D\to\mathbb{B})]$. The ordering $\leqslant$ on $[\mathcal{X}\to(D\to\mathbb{B})]$ is defined as $\eta\leqslant\eta'$ iff for all $X\in\mathcal{X}$, we have $\eta(X) \sqsubseteq \eta'(X)$. The set $([\mathcal{X}\to(D\to\mathbb{B})], \leqslant)$ is also a complete lattice.

Definition 2 (*Parameterised boolean equation system*). A *parameterised boolean equation system* is inductively defined as follows: the empty parameterised boolean equation system is denoted ε, and for a parameterised boolean equation system $\mathcal{E}$, also $(\sigma X(d{:}D) = \varphi)\mathcal{E}$ is a parameterised boolean equation system where $\sigma\in\{\mu, \nu\}$ is a fixpoint symbol and φ a predicate formula.

In the remainder of this article, we abbreviate parameterised boolean equation system with *equation system* if no confusion can arise. The set of *binding predicate variables* in an equation system $\mathcal{E}$, denoted by $\mathrm{bnd}(\mathcal{E})$, is defined as $\mathrm{bnd}(\varepsilon) \stackrel{\text{def}}{=} \emptyset$ and $\mathrm{bnd}((\sigma X(d{:}D) = \varphi)\mathcal{E}) \stackrel{\text{def}}{=} \mathrm{bnd}(\mathcal{E}) \cup \{X\}$, i.e. a binding variable is a variable that occurs at the left-hand side of an equation. An equation system $\mathcal{E}$ is said to be *well-formed* iff all binding predicate variables of $\mathcal{E}$ are unique. Thus, $(\nu X = \top)(\mu X = \bot)$ is not a well-formed equation system. We only consider well-formed equation systems in this paper. We say an equation system $\mathcal{E}$ is *closed* whenever all predicate variables occurring at the right-hand side of the equations in $\mathcal{E}$ (collected in the set $\mathrm{occ}(\mathcal{E})$) are binding variables, i.e. $\mathrm{occ}(\mathcal{E}) \subseteq \mathrm{bnd}(\mathcal{E})$; if an equation system $\mathcal{E}$ is not closed, we say $\mathcal{E}$ is *open*. We say an equation $\sigma X(d{:}D) = \varphi$ is *solved* if φ contains no predicate variables. Likewise, an equation system $\mathcal{E}$ is *solved* iff all its constituting equations are solved. We say that a parameterised boolean equation system is *solved in X* if the predicate variable X does not occur in any right-hand side. The *solution* of an equation system is defined in the context of a predicate environment η and a data environment ε:

Definition 3 (*Solution of an equation system*). The *solution* of an equation system $\mathcal{E}$ in the context of a predicate environment η and a data environment ε is inductively defined as follows (cf. Definition 3.3 of [21]):

$$\begin{array}{ll} [\epsilon]\eta\varepsilon & \stackrel{\text{def}}{=} \eta, \\ [(\sigma X(d{:}D) = \varphi)\mathcal{E}]\eta\varepsilon & \stackrel{\text{def}}{=} [\mathcal{E}](\eta[\sigma X(d{:}D).\varphi([\mathcal{E}]\eta\varepsilon)/X]), \end{array}$$

where $\sigma X(d{:}D).\varphi([\mathcal{E}]\eta\varepsilon)$ is defined as

$$\begin{array}{ll} \mu X(d{:}D).\varphi([\mathcal{E}]\eta\varepsilon) & \stackrel{\text{def}}{=} \bigwedge\{\psi{:}D\to\mathbb{B} \mid \lambda v{:}D.[\![\varphi]\!]([\mathcal{E}]\eta[\psi/X]\varepsilon[v/d])\varepsilon[v/d] \sqsubseteq \psi\}, \\ \nu X(d{:}D).\varphi([\mathcal{E}]\eta\varepsilon) & \stackrel{\text{def}}{=} \bigvee\{\psi{:}D\to\mathbb{B} \mid \psi \sqsubseteq \lambda v{:}D.[\![\varphi]\!]([\mathcal{E}]\eta[\psi/X]\varepsilon[v/d])\varepsilon[v/d]\}. \end{array}$$

As an illustration consider the equation system $(\nu X = Y)(\mu Y = X)$. For a given predicate environment η, its solutions are $\eta[\top/X][\top/Y]$. Note that the solution for $(\mu Y = X)(\nu X = Y)$ is $\eta[\bot/X][\bot/Y]$. This illustrates that the sequence in which the equations occur is of importance.

In the remainder of this paper, we consider only parameterised boolean equation systems for which all data variables that occur at the right-hand side of an equation, are bound at the left-hand side of this equation. For this class of parameterised boolean equation systems, we have the following result:

Lemma 4. *Let η be a predicate environment and let $\varepsilon, \varepsilon'$ be data environments. Let $\mathcal{E}$ be a parameterised boolean equation system for which all data variables occurring at the right-hand side of an equation are bound in the left-hand side. Then $[\mathcal{E}]\eta\varepsilon = [\mathcal{E}]\eta\varepsilon'$.*

From hereon, we use the empty data environment for denoting the solution of an equation system and we generally omit it.

Equation systems are monotone operators on the set of all predicate environments.

Lemma 5. *Let η, η' be predicate environments and $\mathcal{E}$ an arbitrary equation system. Then $\eta \leqslant \eta'$ implies $[\mathcal{E}]\eta \leqslant [\mathcal{E}]\eta'$.*

Proof. By induction on the structure of $\mathcal{E}$. $\square$

In general, the solution of an equation system depends largely on the context in which it is computed (i.e. the predicate environment η). However, for closed equation systems, we have the following theorem.

Theorem 6. *Let $\mathcal{E}$ be a closed equation system. Then for all predicate environments η and η', and all binding variables $X \in \mathrm{bnd}(\mathcal{E})$,*

$$[\mathcal{E}]\eta(X) = [\mathcal{E}]\eta'(X).$$

The following lemma and corollary say that closed equation systems can be solved independently.

Lemma 7. *Let $\mathcal{E}$ and $\mathcal{F}$ be equation systems for which* $(\mathrm{occ}(\mathcal{E}) \cup \mathrm{bnd}(\mathcal{E})) \cap \mathrm{bnd}(\mathcal{F}) = \emptyset$, *and let η be an arbitrary environment. Then*

$$[\mathcal{E}\mathcal{F}]\eta = [\mathcal{F}]([\mathcal{E}]\eta).$$

Proof. We use induction on $\mathcal{E}$.

- Suppose $\mathcal{E}$ is empty. Then we must show that $[\mathcal{F}]\eta = [\mathcal{F}]\eta$, which trivially holds by reflexivity.
- Suppose $\mathcal{E}$ equals $(\sigma X(d{:}D){=}\varphi)\mathcal{E}'$. So, we find that $[(\sigma X(d{:}D){=}\varphi)\mathcal{E}'\mathcal{F}]\eta$ equals by definition $[\mathcal{E}'\mathcal{F}](\eta[\sigma X(d{:}D).\varphi([\mathcal{E}'\mathcal{F}]\eta)/X])$. This equals using the induction hypothesis
$$[\mathcal{F}]([\mathcal{E}']\eta[\sigma X(d{:}D).\varphi([\mathcal{F}]([\mathcal{E}']\eta))/X]). \tag{1}$$
From the assumption, it follows that $\mathrm{bnd}(\mathcal{F}) \cap \mathrm{occ}(\varphi) = \emptyset$. Therefore we have $\varphi([\mathcal{F}]([\mathcal{E}']\eta)) = \varphi([\mathcal{E}']\eta)$. Using this fact and Definition 3, Expression (1) can be shown to be equal to $[\mathcal{F}]([(\sigma X(d{:}D){=}\varphi)\mathcal{E}']\eta)$ as had to be shown. □

Corollary 8. *Let $\mathcal{E}$ be a closed equation system and $\mathcal{F}$ be an equation system for which* $\mathrm{bnd}(\mathcal{E}) \cap \mathrm{bnd}(\mathcal{F}) = \emptyset$, *and let η be an arbitrary environment. Then*

$$[\mathcal{E}\mathcal{F}]\eta = [\mathcal{F}]([\mathcal{E}]\eta).$$

Due to the complex nature of the solution to an equation system (especially the treelike recursion where $\mathcal{E}$ occurs twice in the right-hand side in Definition 3 is tricky), it is not straightforward to solve an equation system. In the subsequent sections, we present lemmas and theorems that help to solve equation systems algebraically.

A well known approach to 'calculate' the solution for a fixpoint equation is by using a transfinite approximation.

Lemma 9. *Let $F = \sigma X(d{:}D).\varphi(\eta\varepsilon)$ with η a predicate environment and ε a data environment. The transfinite approximations X_α of F are defined by*:

$$\begin{array}{lll} & \textit{for } \sigma = \mu & \textit{for } \sigma = \nu, \\ \alpha{=}\beta{+}1 \textit{ is a successor ordinal} & X_{\beta+1} = \varphi[X_\beta/X] & X_{\beta+1} = \varphi[X_\beta/X], \\ \alpha \textit{ is a limit ordinal} & X_\alpha = \bigvee\limits_{\beta<\alpha} X_\beta & X_\alpha = \bigwedge\limits_{\beta<\alpha} X_\beta, \end{array}$$

then $\sigma X(d{:}D).\varphi(\eta\varepsilon) = \lambda v{:}D.[\![X_\alpha]\!]\eta\varepsilon[v/d]$ for some sufficiently large α, where the interpretation of the infinitary disjunction operator $[\![\bigvee_{\beta<\alpha} X_\beta]\!]\eta\varepsilon$ is defined as $\bigvee_{\beta<\alpha}[\![X_\beta]\!]\eta\varepsilon$. The interpretation of the infinitary conjunction operator is similar.

The following result is also useful, as it says that fixpoints can be solved stepwise. This means that the solution of an equation can partly be substituted without altering the solution of the equation.

Lemma 10. *Let $\varphi(X, Y)$ be a predicate formula in which the predicate variables X and Y may occur. Let η be some predicate environment and $F = \sigma X(d{:}D).\varphi(X, X)(\eta)$ and $G = \sigma X(d{:}D).\varphi(X, Y)(\eta[F/Y])$. Then $F = G$.*

Proof. We treat the case where $\sigma = \mu$. The case where $\sigma = \nu$ is fully dual and is therefore omitted. Obviously, F is a solution for X in the second fixpoint. So we find that G is smaller than F. Substituting G for X in the first equation yields $\varphi(X, X)(\eta[G/X])$, which by monotonicity is smaller than $\varphi(X, Y)(\eta[G/X][F/Y])$ which equals G. So, G is a pre-fixpoint of the first equation, which implies that F is smaller than G, showing $F = G$. $\square$

3. Equivalence of parameterised boolean equation systems

Boolean equation systems (BESs) have been studied in great detail [21]. BESs are instances of our parameterised boolean equation systems, i.e. the proposition variables in a BES do not carry data parameters. We introduce two notions of equivalence. The first equivalence is based on the equivalence between BESs, and can be found in the literature [21]. We argue that this equivalence is not suitable and introduce an equivalence that is slightly finer.

Definition 11 (*Standard system equivalence and system ordering*). Let $\mathcal{E}, \mathcal{E}'$ be equation systems. We write $\mathcal{E} \lessdot \mathcal{E}'$ iff for all predicate environments η it holds that $[\mathcal{E}]\eta \leqslant [\mathcal{E}']\eta$. We write $\mathcal{E} \sim \mathcal{E}'$ iff both $\mathcal{E} \lessdot \mathcal{E}'$ and $\mathcal{E}' \lessdot \mathcal{E}$. The relation $\lessdot$ is referred to as the *standard (equation) system ordering*, whereas the relation $\sim$ is referred to as the *standard* (*equation*) *system equivalence*.

Lemma 12. *The relation $\lessdot$ is reflexive, anti-symmetric and transitive. The relation $\sim$ is an equivalence relation.*

Proof. Follows immediately from the definition of $\lessdot$ and $\sim$. $\square$

The standard system equivalence $\sim$ does not allow for compositional reasoning. Consider the two open BESs $\mu X = Y$ and $\nu X = Y$. It is easy to see that $\mu X = Y \sim \nu X = Y$, since both have the same solutions for all predicate environments. However, this does not imply that the two BESs are equivalent in all contexts, since the predicate variable Y can interfere. For example, if we add the equation $\nu Y = X$ to the two BESs, the resulting BESs are different, i.e. we have $(\mu X = Y)(\nu Y = X) \not\sim (\nu X = Y)(\nu Y = X)$, since the solution to the first BES is $X = Y = \perp$, whereas the solution to the second BES is $X = Y = \top$. To mend this situation, we redefine the standard system equivalence and the standard system ordering. Throughout this paper we use this new notion and not the one from [21].

Definition 13 (*System equivalence and system ordering*). Let $\mathcal{E}, \mathcal{E}'$ be equation systems. We write $\mathcal{E} \Rrightarrow \mathcal{E}'$ iff for all predicate environments η and all equation systems $\mathcal{F}$ with $\mathrm{bnd}(\mathcal{F}) \cap (\mathrm{bnd}(\mathcal{E}) \cup \mathrm{bnd}(\mathcal{E}')) = \emptyset$, it holds that $[\mathcal{E}\mathcal{F}]\eta \leqslant [\mathcal{E}'\mathcal{F}]\eta$. We write $\mathcal{E} \equiv \mathcal{E}'$ iff both

$\mathcal{E} \Rrightarrow \mathcal{E}'$ and $\mathcal{E}' \Rrightarrow \mathcal{E}$. The relation $\Rrightarrow$ is referred to as the (*equation*) *system ordering*, whereas the relation $\equiv$ is referred to as (*equation*) *system equivalence*.

Lemma 14. *The relation $\Rrightarrow$ is reflexive, anti-symmetric and transitive. The relation $\equiv$ is an equivalence relation.*

Proof. The proof that $\equiv$ is an equivalence relation follows by definition from the fact that $\Rrightarrow$ is reflexive, anti-symmetric and transitive. Hence, we concentrate on proving these latter properties.

(1) We first show that $\Rrightarrow$ is reflexive. Let $\mathcal{E}, \mathcal{F}$ be arbitrary equation systems, s.t. $\mathrm{bnd}(\mathcal{F}) \cap \mathrm{bnd}(\mathcal{E}) = \emptyset$ and let η be an arbitrary environment. Then, by definition, we have $[\mathcal{E}\mathcal{F}]\eta \leqslant [\mathcal{E}\mathcal{F}]\eta$, i.e. $\mathcal{E} \Rrightarrow \mathcal{E}$.

(2) For anti-symmetry, we reason as follows. Let $\mathcal{E}, \mathcal{E}', \mathcal{F}$ be arbitrary equation systems, s.t. $\mathrm{bnd}(\mathcal{F}) \cap (\mathrm{bnd}(\mathcal{E}) \cup \mathrm{bnd}(\mathcal{E}')) = \emptyset$, and let η be an arbitrary environment. Suppose we have $\mathcal{E} \Rrightarrow \mathcal{E}'$. Hence, by definition $[\mathcal{E}\mathcal{F}]\eta \leqslant [\mathcal{E}'\mathcal{F}]\eta$ and $[\mathcal{E}'\mathcal{F}]\eta \leqslant [\mathcal{E}\mathcal{F}]\eta$. Then by anti-symmetry of $\leqslant$, we have $[\mathcal{E}\mathcal{F}]\eta = [\mathcal{E}'\mathcal{F}]\eta$, i.e. $\mathcal{E} \equiv \mathcal{E}'$.

(3) Finally, we show that $\Rrightarrow$ is transitive. Let $\mathcal{E}, \mathcal{E}', \mathcal{E}''$ be arbitrary equation systems for which $\mathcal{E} \Rrightarrow \mathcal{E}'$ and $\mathcal{E}' \Rrightarrow \mathcal{E}''$ hold. Let $\mathcal{F}$ be an equation system, s.t. $\mathrm{bnd}(\mathcal{F}) \cap (\mathrm{bnd}(\mathcal{E}) \cup \mathrm{bnd}(\mathcal{E}'')) = \emptyset$ and let η be an arbitrary environment. We distinguish two cases:
   (a) Suppose $\mathrm{bnd}(\mathcal{F}) \cap \mathrm{bnd}(\mathcal{E}') \neq \emptyset$. We show that this premise leads to a contradiction. Let $X \in \mathrm{bnd}(\mathcal{F}) \cap \mathrm{bnd}(\mathcal{E}')$, and let $\mathcal{F}'$ be an arbitrary equation system, s.t. $\mathrm{bnd}(\mathcal{F}') \cap (\mathrm{bnd}(\mathcal{E}) \cup \mathrm{bnd}(\mathcal{E}') \cup \mathrm{bnd}(\mathcal{E}'')) = \emptyset$. Then by assumption, we have $[\mathcal{E}\mathcal{F}']\eta \leqslant [\mathcal{E}'\mathcal{F}']\eta$ for all environments η, implying $[\mathcal{E}\mathcal{F}']\eta(X) \sqsubseteq [\mathcal{E}'\mathcal{F}']\eta(X)$. This can only be the case when $[\mathcal{E}'\mathcal{F}']\eta(X) = \top$ for all η, since X does not occur in $\mathcal{E}\mathcal{F}'$. Likewise, we have $[\mathcal{E}'\mathcal{F}']\eta \leqslant [\mathcal{E}''\mathcal{F}']\eta$ for all η, implying $[\mathcal{E}'\mathcal{F}']\eta(X) \sqsubseteq [\mathcal{E}''\mathcal{F}']\eta(X)$. This can only be the case when $[\mathcal{E}'\mathcal{F}']\eta(X) = \bot$ for all η, since X does not occur in $\mathcal{E}''\mathcal{F}'$. But we cannot at the same time have $[\mathcal{E}'\mathcal{F}']\eta(X) = \top$ and $[\mathcal{E}'\mathcal{F}']\eta(X) = \bot$ for all η, hence, we have a contradiction.
   (b) So we may assume that $\mathrm{bnd}(\mathcal{F}) \cap \mathrm{bnd}(\mathcal{E}') = \emptyset$. Then from $[\mathcal{E}\mathcal{F}]\eta \leqslant [\mathcal{E}'\mathcal{F}]\eta$ and $[\mathcal{E}'\mathcal{F}]\eta \leqslant [\mathcal{E}''\mathcal{F}]\eta$, we arrive at $[\mathcal{E}\mathcal{F}]\eta \leqslant [\mathcal{E}''\mathcal{F}]\eta$. Hence, we have $\mathcal{E} \Rrightarrow \mathcal{E}''$, concluding the proof of transitivity.

The system ordering we defined is (unlike the standard system ordering) robust when composing equation systems from smaller equation systems (see Theorem 15). This means that if we have the means to solve equations in isolation, we can use this solved equation for solving equations in a larger context. □

Theorem 15 (*Compositionality of equation systems*). *Let $\mathcal{E}$, $\mathcal{E}'$ and $\mathcal{F}$ be equation systems for which* $\mathrm{bnd}(\mathcal{F}) \cap (\mathrm{bnd}(\mathcal{E}) \cup \mathrm{bnd}(\mathcal{E}')) = \emptyset$. *Then*
(1) $\mathcal{E} \Rrightarrow \mathcal{E}' \Rightarrow \mathcal{F}\mathcal{E} \Rrightarrow \mathcal{F}\mathcal{E}'$,
(2) $\mathcal{E} \Rrightarrow \mathcal{E}' \Rightarrow \mathcal{E}\mathcal{F} \Rrightarrow \mathcal{E}'\mathcal{F}$.

Proof. The second property follows immediately from the definition of $\Rrightarrow$. Thus, we concentrate on the first property. We use induction on the length of $\mathcal{F}$.

(1) Assume $\mathcal{F}$ is the empty equation system. We must show that $\mathcal{E} \Rrightarrow \mathcal{E}'$, but this holds by assumption,

(2) Let η be a predicate environment. Assume $\mathcal{F}$ is of the form $(\sigma X(d{:}D)=\varphi)\mathcal{F}'$. By definition, $[(\sigma X(d{:}D)=\varphi)\mathcal{F}'\mathcal{E}]\eta$ equals $[\mathcal{F}'\mathcal{E}]\eta[\sigma X(d{:}D).\varphi([\mathcal{F}'\mathcal{E}]\eta)/X]$. Using the induction hypothesis and the monotonicity of equation systems over environments, this is at most

$$[\mathcal{F}'\mathcal{E}]\eta[\sigma X(d{:}D).\varphi([\mathcal{F}'\mathcal{E}']\eta)/X].$$

Using the induction hypothesis once more, this in turn is at most

$$[\mathcal{F}'\mathcal{E}']\eta[\sigma X(d{:}D).\varphi([\mathcal{F}'\mathcal{E}']\eta)/X].$$

By definition, this is equivalent to $[(\sigma X(d{:}D)=\varphi)\mathcal{F}'\mathcal{E}']\eta$. Thus

$$(\sigma X(d{:}D)=\varphi)\mathcal{F}'\mathcal{E}\Rrightarrow(\sigma X(d{:}D)=\varphi)\mathcal{F}'\mathcal{E}'.$$

The previous result immediately carries over to system equivalence. □

Corollary 16. *For all equation systems $\mathcal{E}$, $\mathcal{E}'$, $\mathcal{F}$, for which* $\mathrm{bnd}(\mathcal{F})\cap(\mathrm{bnd}(\mathcal{E})\cup\mathrm{bnd}(\mathcal{E}')) = \emptyset$, *we have*
(1) $\mathcal{E} \equiv \mathcal{E}' \Rightarrow \mathcal{F}\mathcal{E} \equiv \mathcal{F}\mathcal{E}'$,
(2) $\mathcal{E} \equiv \mathcal{E}' \Rightarrow \mathcal{E}\mathcal{F} \equiv \mathcal{E}'\mathcal{F}$.

In fact, the standard system equivalence and ordering are very much related to the system equivalence and ordering, as defined in Definition 13. For closed equation systems the two notions coincide.

Lemma 17. *Let $\mathcal{E}$ and $\mathcal{E}'$ be closed equation systems. Then $\mathcal{E} \Rrightarrow \mathcal{E}'$ iff $\mathcal{E} \lll \mathcal{E}'$.*

Proof. The implication from left to right holds by definition. Thus, we focus on the implication from right to left. Let $\mathcal{F}$ be an equation system such that $\mathrm{bnd}(\mathcal{F})\cap(\mathrm{bnd}(\mathcal{E})\cup\mathrm{bnd}(\mathcal{E}')) = \emptyset$. Let η be an arbitrary environment. Since equation systems are monotonic operators, $[\mathcal{E}]\eta \leqslant [\mathcal{E}']\eta$ implies $[\mathcal{F}]([\mathcal{E}]\eta) \leqslant [\mathcal{F}]([\mathcal{E}']\eta)$. Since $\mathcal{E}$ and $\mathcal{E}'$ are closed, this is equivalent to $[\mathcal{E}\mathcal{F}]\eta \leqslant [\mathcal{E}'\mathcal{F}]\eta$ (see Corollary 8). Since this holds for arbitrary $\mathcal{F}$ and η, we also have $\mathcal{E} \Rrightarrow \mathcal{E}'$. □

4. Solving parameterised boolean equation systems

In Section 4.1, we identify several rules for calculating with equation systems as a whole and we present a completeness result that says that if single equations can be solved in one variable a complete parameterised boolean equation system can be solved. In Section 4.2, we present several techniques that can be applied to solve equations for a single variable.

4.1. Global techniques for solving parameterised boolean equation systems

The focus in this section is on algebraic techniques for solving equation systems as a whole. The first lemma also appeared in [21] as Lemma 6.3 using a slightly different phrasing. It allows to substitute the right-hand side of an equation for the left-hand side in all the equations preceding it. In [21], this step formed an essential part of the so-called *Gauß elimination* procedure to solve boolean equation systems.

Lemma 18 (*Substitution*). *Let $\mathcal{E}$ be an equation system for which $X, Y \notin \mathrm{bnd}(\mathcal{E})$, then*:

$$(\sigma X(d{:}D) = \varphi)\mathcal{E}(\sigma' Y(e{:}E) = \psi) \equiv (\sigma X(d{:}D) = \varphi[\psi/Y])\mathcal{E}(\sigma' Y(e{:}E) = \psi).$$

Proof. Let $\mathcal{F}$ be an arbitrary equation system and η be an environment. We reason as follows. By Definition 3, it suffices to show that:

$$\begin{aligned}&[\mathcal{E}(\sigma' Y(e{:}E){=}\psi)\mathcal{F}]\eta[\sigma X(d{:}D).\varphi([\mathcal{E}(\sigma' Y(e{:}E) = \psi)\mathcal{F}]\eta)/X]\\ &\quad = [\mathcal{E}(\sigma' Y(e{:}E){=}\psi)\mathcal{F}]\eta[\sigma X(d{:}D).\varphi[\psi/Y]([\mathcal{E}(\sigma' Y(e{:}E) = \psi)\mathcal{F}]\eta)/X].\end{aligned}$$

This follows directly from the following observation:

$$\varphi([\mathcal{E}(\sigma' Y(e{:}E) = \psi)\mathcal{F}]\eta) = \varphi[\psi/Y]([\mathcal{E}(\sigma' Y(e{:}E) = \psi)\mathcal{F}]\eta). \tag{2}$$

We show this by induction on the length of $\mathcal{E}$. If $\mathcal{E}$ is empty (2) can be shown as follows:

$$\begin{aligned}&\varphi([(\sigma' Y(e{:}E) = \psi)\mathcal{F}]\eta)\\ &\quad = \varphi([\mathcal{F}]\eta[\sigma' Y(e{:}E).\psi([\mathcal{F}]\eta)/Y])\\ &\quad = \varphi[\psi/Y]([\mathcal{F}]\eta[\sigma' Y(e{:}E).\psi([\mathcal{F}]\eta)/Y])\\ &\quad = \varphi[\psi/Y]([(\sigma' Y(e{:}E) = \psi)\mathcal{F}]\eta).\end{aligned}$$

The one but last step follows as $\sigma' Y(e{:}E).\psi([\mathcal{F}]\eta)$ is a fixpoint for the equation for Y. If $\mathcal{E}$ consists of $(\sigma'' Z(f{:}F) = \chi)\mathcal{E}'$, then we derive

$$\begin{aligned}&\varphi([(\sigma'' Z(f{:}F) = \chi)\mathcal{E}'(\sigma' Y(e{:}E) = \psi)\mathcal{F}]\eta)\\ &\quad = \varphi([\mathcal{E}'(\sigma' Y(e{:}E) = \psi)\mathcal{F}](\eta[\sigma'' Z(f{:}F).\chi([\mathcal{E}'(\sigma' Y(e{:}E) = \psi)\mathcal{F}]\eta)/Z]))\\ &\quad \overset{\text{i.h.}}{=} \varphi[\psi/Y]([\mathcal{E}'(\sigma' Y(e{:}E) = \psi)\mathcal{F}](\eta[\sigma'' Z(f{:}F).\chi([\mathcal{E}'(\sigma' Y(e{:}E){=}\psi)\mathcal{F}]\eta)/Z]))\\ &\quad = \varphi[\psi/Y]([(\sigma'' Z(f{:}F) = \chi)\mathcal{E}'(\sigma' Y(e{:}E) = \psi)\mathcal{F}]\eta).\end{aligned}$$

This finishes this proof. □

The sequence in which equations in a parameterised boolean equation system occur is important. It is only allowed to change this order under very particular circumstances.

Lemma 19 (*Migration*). *Let $\sigma X(d{:}D) = \varphi$ be a solved equation, i.e.* $\mathrm{occ}(\varphi) = \emptyset$, *and $\mathcal{E}$ an equation system, such that $X \notin \mathrm{bnd}(\mathcal{E})$, then*:

$$(\sigma X(d{:}D) = \varphi)\mathcal{E} \equiv \mathcal{E}(\sigma X(d{:}D) = \varphi).$$

Proof. By induction on the size of $\mathcal{E}$.

(1) Assume $\mathcal{E}$ is the empty equation system. Then we must show $(\sigma X(d{:}D) = \varphi) \equiv (\sigma X(d{:}D) = \varphi)$, which holds by reflexivity of $\equiv$.

(2) Assume $\mathcal{E}$ has the form $(\sigma' Y(e{:}E) = \psi)\mathcal{E}'$. Let $\mathcal{F}$ be an arbitrary equation system and η an arbitrary environment. We calculate as follows. Given that φ contains no predicate

variables, we have

$$\begin{aligned}&[(\sigma X(d{:}D) = \varphi)(\sigma' Y(e{:}E) = \psi)\mathcal{E}'\mathcal{F}]\eta \\ &\quad = [\mathcal{E}'\mathcal{F}]\eta[\varphi/X][(\sigma' Y(e{:}E).\psi([\mathcal{E}'\mathcal{F}]\eta[\varphi/X]))/Y].\end{aligned}$$

We have $\eta[\varphi/X] = \eta[(\sigma X(d{:}D).\varphi([\mathcal{E}'\mathcal{F}]\eta))/X]$, since φ contains no predicate variables. Then, by definition, we have

$$\begin{aligned}&[\mathcal{E}'\mathcal{F}]\eta[(\sigma X(d{:}D).\varphi([\mathcal{E}'\mathcal{F}]\eta))/X] \\ &\quad [(\sigma' Y(e{:}E).\psi([\mathcal{E}'\mathcal{F}]\eta[(\sigma X(d{:}D).\varphi([\mathcal{E}'\mathcal{F}]\eta))/X]))/Y] \\ &\qquad = [(\sigma X(d{:}D) = \varphi)\mathcal{E}'\mathcal{F}]\eta[(\sigma' Y(e{:}E).\psi([(\sigma X(d{:}D) = \varphi)\mathcal{E}'\mathcal{F}]\eta))/Y].\end{aligned}$$

Now, applying the induction hypothesis twice, we have

$$\begin{aligned}&[\mathcal{E}'(\sigma X(d{:}D) = \varphi)\mathcal{F}]\eta[(\sigma' Y(e{:}E)\psi([\mathcal{E}'(\sigma X(d{:}D) = \varphi)\mathcal{F}]\eta))/Y] \\ &\quad = [(\sigma' Y(e{:}E) = \psi)\mathcal{E}'(\sigma X(d{:}D) = \varphi)\mathcal{F}]\eta.\end{aligned}$$

This concludes the proof. □

The following theorem states that we have all the requirements to solve an equation system if we can solve a single equation.

Theorem 20 (*Global completeness*). *Assume we can derive for arbitrary equations* $(\sigma X(d{:}D){=}\varphi) \equiv (\sigma X(d{:}D){=}\psi)$, *such that* X *does not occur in* ψ. *Then all* closed *equation systems can be rewritten to* solved *equation systems using the rules of migration and substitution.*

Proof. Consider a closed equation system $\mathcal{E}$ being equal to

$$(\sigma_1 X_1(d_1{:}D_1){=}\varphi_1)\ldots(\sigma_n X_n(d_n{:}D_n){=}\varphi_n).$$

We prove the theorem in two stages. First we transform $\mathcal{E}$ to an equivalent equation system $\mathcal{E}'$ for which X_i $(1 \leqslant i \leqslant n)$ does not occur in any φ_j for $j \leqslant i$. We call this requirement 1. Suppose requirement 1 does not hold. Consider the largest i such that X_i occurs in some φ_j for $j \leqslant i$. If X_i occurs in φ_i, then by assumption we can replace φ_i by ψ in $\mathcal{E}$ where X_i does not occur in ψ maintaining system equivalence. Using Lemma 18 (substitution) we can remove all occurrences of X_i in φ_j for $j < i$. By repeatedly applying this step we have obtained our desired equation system satisfying requirement 1.

Now, we transform $\mathcal{E}'$ such that for all i, X_i does not occur in any of the φ_j for $j > i$, too. We call this requirement 2. Note that any closed equation system satisfying requirement 1 and 2 is solved. Consider the first equation $\sigma_i X_i(d_i{:}D_i){=}\varphi_i$ not satisfying requirement 2. Observe that φ_i does not contain any predicate variable. So, we can move this equation to the last position of the equation system using Lemma 19 (migration). Using Lemma 18 we can substitute φ_i for X_i in all other equations. By Lemma 19 we can move this equation back to its original place. Observe that the newly obtained parameterised boolean equation system satisfies requirements 1 and 2 for i and is equivalent to the old equation system. Repeatedly applying this step yields an equation system completely satisfying requirements 1 and 2. As already observed above, the equation system is thereby solved, proving this theorem. □

The following lemma is convenient to reorder the equations in equation systems, but it is not needed for completeness. A similar lemma already appeared in [21, Lemma 3.21] for the standard system equivalence. It carries over to our notion of system equivalence.

Lemma 21 (*Switching*). *Let $\sigma X(d{:}D)=\varphi$ and $\sigma Y(e{:}E)=\psi$ be equations with the same fixpoint symbol σ. Then, the following equality holds*:

$$(\sigma X(d{:}D)=\varphi)(\sigma Y(e{:}E)=\psi) \equiv (\sigma Y(e{:}E)=\psi)(\sigma X(d{:}D)=\varphi).$$

Proof. Follows from Bekič's [2] theorem for elimination of simultaneous fixpoints and Definition 3. □

In [21, Lemma 3.22], the author (and also we in [17]) claims the following property, which turns out not to hold, as we will shortly show.

Let φ and ψ be predicate formulae for which Y does not occur in φ and X does not occur in ψ, then

$$(\sigma X(d{:}D)=\varphi)(\sigma' Y(e{:}E)=\psi) \equiv (\sigma' Y(e{:}E)=\psi)(\sigma X(d{:}D)=\varphi).$$

A simple counter example to the above claim is as follows. Consider the two open equation systems $(\nu X = Z)(\mu Y = W)$ and $(\mu Y = W)(\nu X = Z)$. We find that $(\nu X = Z)(\mu Y = W) \not\equiv (\mu Y = W)(\nu X = Z)$. To see this, consider the (right) closure of the above equation systems with the equation system $(\mu Z = Y)(\mu W = X)$. Then we find that the solution to the first equation system is $X = Y = Z = W = \top$, whereas the solution to the second equation system is $X = Y = Z = W = \bot$.

To further stress the difference between the standard system equivalence and our notion of system equivalence, we find that the above property does hold when we use the standard system equivalence, as the following lemma shows.

Lemma 22 (*Independence*). *Let φ and ψ be predicate formulae for which Y does not occur in φ and X does not occur in ψ, then*

$$(\sigma X(d{:}D)=\varphi)(\sigma' Y(e{:}E)=\psi) \sim (\sigma' Y(e{:}E)=\psi)(\sigma X(d{:}D)=\varphi).$$

Proof. Let η be an arbitrary environment. By definition,

$$[(\sigma X(d{:}D)=\varphi)(\sigma' Y(e{:}E)=\psi)]\eta$$

is equivalent to

$$\eta[\sigma X(d{:}D).\varphi(\eta[\sigma' Y(e{:}E).\psi(\eta)/Y])/X]$$
$$[\sigma' Y(e{:}E).\psi(\eta[\sigma X(d{:}D).\varphi(\eta[\sigma' Y(e{:}E).\psi(\eta)/Y])/X])/Y].$$

Now, since Y does not occur in φ and X does not occur in ψ, this equals

$$\eta[\sigma X(d{:}D)\varphi(\eta)/X][\sigma' Y(e{:}E)\psi(\eta)/Y].$$

Following the above steps in reverse order, we find that this is equivalent to

$$\eta[\sigma' Y(e{:}E).\psi(\eta[\sigma X(d{:}D).\varphi(\eta)/X])/Y]$$
$$[\sigma X(d{:}D).\varphi(\eta[\sigma' Y(e{:}E).\psi(\eta[\sigma X(d{:}D).\varphi(\eta)/X])/Y])/X].$$

By definition, this is equivalent to

$$[(\sigma' Y(e{:}E){=}\psi)(\sigma X(d{:}D){=}\varphi)]\eta,$$

which concludes the proof. □

In some cases only an approximation of a solution can be found for a particular equation, for instance $\sigma X(d{:}D){=}\varphi \Rrightarrow \sigma X(d{:}D){=}\psi$. The following two theorems indicate that such an approximation can still be used to derive the equivalence between two equation systems. First we provide a lemma needed to facilitate the proof.

Lemma 23. *Let φ, ψ and χ be predicate formulae such that the variable $X \notin \mathrm{occ}(\psi)$. Let $\mathcal{F}$ be an equation system containing an equation of the form $\sigma X(d{:}D){=}\varphi$ and let η be a predicate environment. If*
(1) *$(\sigma X(d{:}D){=}\psi)\Rrightarrow(\sigma X(d{:}D){=}\varphi)$ and*
(2) *χ and $\chi[\lambda d{:}D.(\psi \wedge X(d))/X]$ are logically equivalent*
then

$$\sigma' Y(e{:}E).\chi([\mathcal{F}]\eta) = \sigma' Y(e{:}E).\chi[\lambda d{:}D.\psi/X]([\mathcal{F}]\eta).$$

Proof. The first condition says $(\sigma X(d{:}D){=}\psi) \Rrightarrow (\sigma X(d{:}D){=}\varphi)$, which we rewrite to a form that can subsequently be used. So, the condition is equivalent to for all equation systems $\mathcal{G}$ and predicate environments η:

$$[(\sigma X(d{:}D){=}\psi)\mathcal{G}]\eta \leqslant [(\sigma X(d{:}D){=}\varphi)\mathcal{G}]\eta,$$

which by definition is equivalent to

$$[\mathcal{G}]\eta[\sigma X(d{:}D).\psi([\mathcal{G}]\eta)/X] \leqslant [\mathcal{G}]\eta[\sigma X(d{:}D).\varphi([\mathcal{G}]\eta)/X].$$

By applying both sides on X one can see that this yields

$$\sigma X(d{:}D).\psi([\mathcal{G}]\eta) \sqsubseteq \sigma X(d{:}D).\varphi([\mathcal{G}]\eta)$$

and as $X \notin \mathrm{occ}(\psi)$ this is equivalent to

$$\lambda d{:}D.\psi([\mathcal{G}]\eta) \sqsubseteq \sigma X(d{:}D).\varphi([\mathcal{G}]\eta).$$

So, in other words, the expressions

$$\psi([\mathcal{G}]\eta) \quad \text{and} \quad \psi([\mathcal{G}]\eta) \wedge \sigma X(d{:}D).\varphi([\mathcal{G}]\eta)(d) \tag{3}$$

are logically equivalent for all $d{:}D$ and all $\mathcal{G}$.

Now we turn to the proof of this lemma. Recall that $\mathcal{F}$ is an equation system containing an equation of the form $\sigma X(d{:}D) = \varphi$. We use induction on the size of $\mathcal{F}$. If $\mathcal{F}$ is empty, the theorem holds because the premise that $\sigma X(d{:}D){=}\varphi$ is in $\mathcal{F}$, is clearly invalid.

So, assume $\mathcal{F}$ is not empty. We distinguish the following two cases:

- $\mathcal{F}$ has the form $(\sigma X(d{:}D){=}\varphi)\mathcal{F}'$. Hence,

$$\begin{aligned}
&\sigma' Y(e{:}E).\chi([\mathcal{F}]\eta)\\
&\quad= \sigma' Y(e{:}E).\chi([(\sigma X(d{:}D){=}\varphi)\mathcal{F}']\eta)\\
&\quad= \sigma' Y(e{:}E).\chi([\mathcal{F}']\eta[\sigma X(d{:}D).\varphi([\mathcal{F}']\eta)/X])\\
&\quad=^1 \sigma' Y(e{:}E).\chi[\lambda d{:}D.\psi \wedge X(d)/X]([\mathcal{F}']\eta[\sigma X(d{:}D).\varphi([\mathcal{F}']\eta)/X])\\
&\quad= \sigma' Y(e{:}E).\chi[\lambda d{:}D.\psi{\wedge}\sigma X(d{:}D).\varphi([\mathcal{F}']\eta)(d)/X]([\mathcal{F}']\eta[\sigma X(d{:}D).\varphi([\mathcal{F}']\eta)/X])\\
&\quad=^2 \sigma' Y(e{:}E).\chi[\lambda d{:}D.\psi/X]([\mathcal{F}']\eta[\sigma X(d{:}D).\varphi([\mathcal{F}']\eta)/X])\\
&\quad= \sigma' Y(e{:}E).\chi[\lambda d{:}D.\psi/X]([(\sigma X(d{:}D){=}\varphi)\mathcal{F}']\eta).
\end{aligned}$$

  At $=^1$ we use the second condition and at $=^2$ we use (3) and $X \notin \mathrm{occ}(\psi)$.

- $\mathcal{F}$ has the form $(\sigma'' Z(f{:}F){=}\xi)\mathcal{F}'$ with $Z \neq X$ and $\sigma X(d{:}D) = \varphi$ in $\mathcal{F}'$. So, we get

$$\begin{aligned}
&\sigma' Y(e{:}E).\chi([\mathcal{F}]\eta)\\
&\quad= \sigma' Y(e{:}E).\chi([(\sigma'' Z(f{:}F){=}\xi)\mathcal{F}']\eta)\\
&\quad= \sigma' Y(e{:}E).\chi([\mathcal{F}']\eta[\sigma'' Z(f{:}F).\xi([\mathcal{F}']\eta)/X])\\
&\quad\overset{\text{i.h.}}{=} \sigma' Y(e{:}E).\chi[\psi/X]([\mathcal{F}']\eta[\sigma'' Z(f{:}F).\xi([\mathcal{F}']\eta)/X])\\
&\quad\overset{\text{i.h.}}{=} \sigma' Y(e{:}E).\chi[\psi/X]([(\sigma'' Z(f{:}F){=}\xi)\mathcal{F}']\eta)\\
&\quad= \sigma' Y(e{:}E).\chi[\psi/X]([\mathcal{F}]\eta),
\end{aligned}$$

which finishes the proof. □

Theorem 24. *Let $\mathcal{E}$ be an equation system and let φ, ψ and χ be predicate formulae such that the variable $X \notin \mathrm{occ}(\psi)$. If*
(1) *$\sigma X(d{:}D) = \psi \Rrightarrow \sigma X(d{:}D) = \varphi$ and*
(2) *χ and $\chi[\lambda d{:}D.(\psi \wedge X(d))/X]$ are logically equivalent*
then

$$\begin{aligned}
&(\sigma' Y(e{:}E) = \chi)\mathcal{E}(\sigma X(d{:}D) = \varphi)\\
&\quad\equiv (\sigma' Y(e{:}E) = \chi[\lambda d{:}D.\psi/X])\mathcal{E}(\sigma X(d{:}D) = \varphi).
\end{aligned}$$

Proof. Using the definition we must show for all equation systems $\mathcal{F}$ and predicate environments η:

$$\begin{aligned}
&[(\sigma' Y(e{:}E) = \chi)\mathcal{E}(\sigma X(d{:}D) = \varphi)\mathcal{F}]\eta\\
&\quad= [(\sigma' Y(e{:}E) = \chi[\lambda d{:}D.\psi/X])\mathcal{E}(\sigma X(d{:}D) = \varphi)\mathcal{F}]\eta.
\end{aligned}$$

By definition this is equivalent to

$$\begin{aligned}
&[\mathcal{E}(\sigma X(d{:}D) = \varphi)\mathcal{F}]\eta[\sigma' Y(e{:}E).\chi([\mathcal{E}(\sigma X(d{:}D) = \varphi)\mathcal{F}]\eta)/Y]\\
&\quad[\mathcal{E}(\sigma X(d{:}D) = \varphi)\mathcal{F}]\eta[\sigma' Y(e{:}E).\chi[\lambda d{:}D.\psi/X]([\mathcal{E}(\sigma X(d{:}D) = \varphi)\mathcal{F}]\eta)/Y],
\end{aligned}$$

which is a direct consequence of Lemma 23. □

Below we state the dual of the previous theorem without proof.

Theorem 25. *Let $\mathcal{E}$ be an equation system and let φ, ψ and χ be predicate formulae such that the variable $X \notin \mathrm{occ}(\psi)$. If*
(1) $\sigma X(d{:}D) = \varphi \Rrightarrow \sigma X(d{:}D) = \psi$ *and*
(2) χ *and* $\chi[\lambda d{:}D.(\psi \vee X(d))/X]$ *are logically equivalent*
then

$$(\sigma' Y(e{:}E) = \chi)\mathcal{E}(\sigma X(d{:}D) = \varphi)$$
$$\equiv (\sigma' Y(e{:}E) = \chi[\lambda d{:}D.\psi/X])\mathcal{E}(\sigma X(d{:}D) = \varphi).$$

4.2. Techniques for finding local solutions

In Theorem 20 it has been shown that we can solve a parameterised boolean equation system, if we can solve each equation of the form $\sigma X(d{:}D) = \varphi$ in X, i.e. if we can find an equivalent equation in which X does not occur in the right-hand side. In this section, we focus on techniques to find such equations.

We do not strive for completeness in any formal sense here. Our focus in this paper is to yield a set of rules that allows effective manual verification, and we have shown efficacy by applying our rules to numerous examples some of which are presented in Section 5. General incompleteness results indicate that completeness can only be achieved under particular circumstances. For instance, it is possible to prove completeness using infinitary logics (see e.g. [19]). But such means are unwieldy for practical purposes and generally only satisfy a general desire for completeness results. Completeness can also be achieved for restricted data types. This is useful as such exercises can reveal new verification rules and techniques. Albeit interesting, we do not treat such questions in this paper and postpone these to further investigations in the field.

4.2.1. Predicate calculus

A self evident way of solving a single equation is by applying the standard rules of predicate calculus. In order to use these, we first define logical implication for our setting.

Definition 26 (*Logical implication and logical equivalence*). Let φ, φ' be arbitrary predicate formulae. We write $\varphi \rightarrow \varphi'$, representing *logical implication* which is defined as $[\![\varphi]\!]\eta\varepsilon$ implies $[\![\varphi']\!]\eta\varepsilon$ for all data environments ε and predicate environments η. We write $\varphi \leftrightarrow \varphi'$ as a shorthand for $\varphi \rightarrow \varphi'$ and $\varphi' \rightarrow \varphi$.

Note that in this definition we used a data environment, which is only important if free data variables occur in formulae. In line with the rest of this paper, we omit the data environment elsewhere.

Lemma 27. *The relation $\rightarrow$ is reflexive, anti-symmetric and transitive. The relation $\leftrightarrow$ is an equivalence relation.*

Well-known rules from predicate logic such as given in Table 1, allow symbolic manipulations for transforming and rewriting predicate formulae to simpler predicate formulae. These rules are valid for the implication arrow as defined in Definition 26. The following lemma and corollary express how implications derivable using the rules in Table 1 can be

Table 1
Transformation rules for predicate formulae; χ, φ and ψ are predicate formulae

$\varphi \wedge \varphi \leftrightarrow \varphi$	$\varphi \vee \varphi \leftrightarrow \varphi$
$\varphi \wedge \psi \leftrightarrow \psi \wedge \varphi$	$\varphi \vee \psi \leftrightarrow \psi \vee \varphi$
$\varphi \wedge (\psi \wedge \chi) \leftrightarrow (\varphi \wedge \psi) \wedge \chi$	$\varphi \vee (\psi \vee \chi) \leftrightarrow (\varphi \vee \psi) \vee \chi$
$\varphi \wedge (\psi \vee \chi) \leftrightarrow (\varphi \wedge \psi) \vee (\varphi \wedge \chi)$	$\varphi \vee (\psi \wedge \chi) \leftrightarrow (\varphi \vee \psi) \wedge (\varphi \vee \chi)$
$\varphi \wedge \psi \rightarrow \varphi$	$\varphi \rightarrow \varphi \vee \psi$
$\varphi \wedge (\varphi \vee \psi) \leftrightarrow \varphi$	$\varphi \vee (\varphi \wedge \psi) \leftrightarrow \varphi$
$\varphi \vee \bot \leftrightarrow \varphi$	$\varphi \wedge \top \leftrightarrow \varphi$
$\varphi \wedge \bot \leftrightarrow \bot$	$\varphi \vee \top \leftrightarrow \top$
$\forall d{:}D.\varphi \rightarrow \varphi$	$\varphi \rightarrow \exists d{:}D.\varphi$
$\forall d{:}D.(\varphi \wedge \psi) \leftrightarrow \forall d{:}D.\varphi \wedge \forall d{:}D.\psi$	$\exists d{:}D.(\varphi \vee \psi) \leftrightarrow \exists d{:}D.\varphi \vee \exists d{:}D.\psi$

used in equation systems. We found that it is not always easy to solve equations directly. But by weakening or strengthening the equations a little using for instance Lemma 28, we can replace an equation by an approximate, which can be easier to solve and which is sufficient for the purposes at hand.

Lemma 28 (*Monotonicity of predicate formulae*). *Let φ and ψ be predicate formulae such that $\varphi \rightarrow \psi$. Then $(\sigma X(d{:}D) = \varphi) \Rrightarrow (\sigma X(d{:}D) = \psi)$.*

Proof. As $\varphi \rightarrow \psi$, $[\![\varphi]\!]\eta\varepsilon$ implies $[\![\psi]\!]\eta\varepsilon$ for any predicate environment η and data environment ε. So, by monotonicity, $\sigma X(d{:}D).\varphi([\mathcal{F}]\eta) \sqsubseteq \sigma X(d{:}D).\psi([\mathcal{F}]\eta)$. Again using monotonicity, we find that

$$[\mathcal{F}]\eta[\sigma X(d{:}D).\varphi([\mathcal{F}]\eta)/X] \leqslant [\mathcal{F}]\eta[\sigma X(d{:}D).\psi([\mathcal{F}]\eta)/X].$$

This is exactly equivalent to what we have to prove. $\square$

From Lemma 28, the following consequence is immediate.

Corollary 29. *Let φ and ψ be arbitrary predicate formulae for which $\varphi \leftrightarrow \psi$. We find that $(\sigma X(d{:}D) = \varphi) \equiv (\sigma X(d{:}D) = \psi)$.*

The route from equation systems to formulae only works in restricted cases.

Lemma 30. *Let φ and ψ be arbitrary predicate formulae such that $X \notin \mathrm{occ}(\varphi) \cup \mathrm{occ}(\psi)$. If $\sigma X(d{:}D) = \varphi \Rrightarrow \sigma X(d{:}D) = \psi$ then $\varphi \rightarrow \psi$ or in other words $\varphi \leftrightarrow \varphi \wedge \psi$ or $\psi \leftrightarrow \varphi \vee \psi$.*

Proof. By assumption we have

$$\sigma X(d{:}D) = \varphi \Rrightarrow \sigma X(d{:}D) = \psi.$$

So, by definition, for all $\mathcal{F}$ and η we find:

$$[(\sigma X(d{:}D) = \varphi)\mathcal{F}]\eta \leqslant [(\sigma X(d{:}D) = \psi)\mathcal{F}]\eta.$$

Again by definition

$$[\mathcal{F}]\eta[\sigma X(d{:}D).\varphi([\mathcal{F}]\eta)/X] \leqslant [\mathcal{F}]\eta[\sigma X(d{:}D).\psi([\mathcal{F}]\eta)/X].$$

If we apply left- and right-hand side to X and by taking $\mathcal{F}$ empty, we may conclude

$$\sigma X(d{:}D).\varphi(\eta) \sqsubseteq \sigma X(d{:}D).\psi(\eta).$$

As X does not occur in φ and ψ, we find that the fixpoints equal $\lambda d{:}D.[\![\varphi]\!]\eta$ and $\lambda d{:}D.[\![\psi]\!]\eta$. So, for all $d{:}D$:

$$[\![\varphi]\!]\eta \text{ implies } [\![\psi]\!]\eta.$$

This is by Definition 26 equal to $\varphi \to \psi$.

Note that the following rephrasing of the theorem is *not* true if $X \in \mathrm{occ}(\varphi) \cup \mathrm{occ}(\psi)$.

$$\begin{array}{l} \quad (\nu X(d{:}D) = \varphi) \Rrightarrow (\nu X(d{:}D) = \psi) \\ \text{implies} \\ \quad (\nu X(d{:}D) = \varphi) \equiv (\nu X(d{:}D) = \varphi \wedge \psi). \end{array} \tag{4}$$

A simple counter example is the following. Take d and D equal to n and $\mathbb{N}$ and consider $\varphi = n{<}1$ and $\psi = X(n+1)$. We find that obviously

$$(\nu X(n{:}\mathbb{N}) = n < 1) \Rrightarrow (\nu X(n{:}\mathbb{N}) = X(n+1)),$$

as the solution for the right-hand side is $X(n) = \top$. But it does not hold that

$$(\nu X(n{:}\mathbb{N}) = n < 1) \equiv (\nu X(n{:}\mathbb{N}) = X(n+1) \wedge n{<}1)$$

as the right-hand side has solution $X(n) = \bot$ which clearly does not match the solution of the left-hand side. There are other counter examples showing that (4) does not hold when ν is replaced by μ, and/or $\wedge$ is replaced by $\vee$. □

4.2.2. *Iterative approximation*

A straightforward (but usually laborious) method for solving an equation $\sigma X(d{:}D) = \varphi$ in X is by means of an iterative approximation of the fixpoint solution of X, which is possible as we are dealing with a monotonic lattice. One starts with an initial solution S_0 for X being either $\lambda d{:}D.\bot$ (for $\sigma = \mu$) or $\lambda d{:}D.\top$ (for $\sigma = \nu$). Then the approximate solutions of the form $\lambda d{:}D.S_{n+1} = \varphi[S_n/X]$ are calculated repeatedly. A *stable approximant* is an approximant that is logically equivalent to its next approximation. Such stable approximants are in fact the fixpoint solution to the equation. But in general this procedure does not terminate, since the lattice $(D, \sqsubseteq)$ can have infinite ascending chains. However, using the equation system ordering approximants that are not stable can still be of use in solving equation systems. This is another motivation for defining an ordering on equation systems.

Definition 31. Let φ, ψ be predicate formulae and X a predicate variable. We inductively define $\psi[\varphi/X]^k$, where k is of sort $\mathbb{N}$.

(1) $\psi[\varphi/X]^0 \overset{\text{def}}{=} \varphi$, and

(2) $\psi[\varphi/X]^{k+1} \overset{\text{def}}{=} \psi[(\psi[\varphi/X]^k)/X]$.

Thus, $\psi[\varphi/X]^k$ represents the result of recursively substituting φ for X in ψ. Note that for any $k{:}\mathbb{N}$, and all predicate formulae ψ, φ, the expression $\psi[\varphi/X]^k$ is a predicate formula. Below we state that $\varphi[\bot/X]^k$ and $\varphi[\top/X]^k$ are approximations of the solution of an equation and that a stable approximant is *the* solution to an equation.

Lemma 32 (*Approximants as (pre-)solutions*). *Let φ be a predicate formula and $k{:}\mathbb{N}$ be an arbitrary natural number. Then*
(1) $(\mu X(d{:}D) = \varphi[\bot/X]^k) \Rrightarrow (\mu X(d{:}D) = \varphi)$.
(2) $(\nu X(d{:}D) = \varphi) \Rrightarrow (\nu X(d{:}D) = \varphi[\top/X]^k)$.

Proof. Follows from the fact that pre-solutions imply/are implied by the solution of the equation system and Lemma 28. □

Lemma 33 (*Stable approximants as solutions*). *Let φ be a predicate formula and $k{:}\mathbb{N}$ be a natural number. Then*
(1) *If $\varphi[\bot/X]^k \leftrightarrow \varphi[\bot/X]^{k+1}$ then $(\mu X(d{:}D) = \varphi[\bot/X]^k) \equiv (\mu X(d{:}D) = \varphi)$.*
(2) *If $\varphi[\top/X]^k \leftrightarrow \varphi[\top/X]^{k+1}$ then $(\nu X(d{:}D) = \varphi) \equiv (\nu X(d{:}D) = \varphi[\top/X]^k)$.*

A less mechanic but often more efficient version of Lemmata 32 and 33 is Lemma 34. In the setting of parameterised boolean equation systems this lemma first appeared in [11]. It allows one to “guess” an approximate solution to an equation. Only a relatively simple (inductive) check is needed to establish that this solution indeed approximates the exact solution of the fixpoint equation.

Lemma 34 (*Groote and Mateescu*). *Let φ, ψ be predicate formulae where $k{:}\mathbb{N}$ is possibly a free variable in φ and X a free variable in ψ. Then*:
(1) *If for all k, $\varphi(k) \rightarrow \psi[\bot/X]^k$, then $(\mu X(d{:}D) = \exists k{:}\mathbb{N}.\varphi(k)) \Rrightarrow (\mu X(d{:}D) = \psi)$.*
(2) *If $\psi[\varphi/X] \rightarrow \varphi$, then $(\mu X(d{:}D) = \psi) \Rrightarrow (\mu X(d{:}D) = \varphi)$.*
(3) *If for all k, $\psi[\top/X]^k \rightarrow \varphi(k)$, then $(\nu X(d{:}D) = \psi) \Rrightarrow (\nu X(d{:}D) = \forall k{:}\mathbb{N}.\varphi(k))$.*
(4) *If $\varphi \rightarrow \psi[\varphi/X]$, then $(\nu X(d{:}D) = \varphi) \Rrightarrow (\nu X(d{:}D) = \psi)$.*

Proof. Along the lines of [11]. □

The first rule in Lemma 34 captures the fact that for a least fixpoint, a carefully chosen formula is a smaller solution to an equation when it is always at most the kth approximant. The second rule describes the case when we have a solution to an equation (which is not necessarily the least solution). The third and fourth rules are the dual counterparts of the rules for the greatest fixpoint.

4.2.3. Patterns for equation systems

The techniques for finding the solution to equation systems we described in the previous section are not always efficient or easy to apply. For instance, iterative approximation is not always applicable, as the following example shows.

Example 35. Consider the following greatest fixpoint equation: $\nu X(i{:}\mathbb{N}) = i \leqslant N \wedge X(i+1)$, where N is some arbitrary natural number. By approximating, we obtain infinitely many approximants, without ever reaching the solution. Obviously, the solution to this equation should be $\forall j{:}\mathbb{N}.i + j \leqslant N$, which can be further reduced to $\bot$.

In order to be able to solve such an equation effectively, we need to resort to a different method altogether. We study equations of a certain generic form, and provide generic solutions to these equations. Equations, such as the one from the above example, can then be recognised to be of a certain form, and be solved, simply by looking them up. We refer to these abstract equations as *patterns*. Note that identifying 'patterns' is very common in mathematics, for instance when solving differential equations.

The first pattern is obtained by generalising the equation in the example given above. Note that the solutions for the minimal and maximal fixpoint equations are dual. Let $f{:}D \to D$ be an arbitrary, total function. We assume the existence of a function $f{:}\mathbb{N} \times D \to D$, written as $f^n(d)$, with the property that $f^0(d) = d$ and $f^{n+1}(d) = f(f^n(d))$.

Theorem 36. *Let $\sigma X(d{:}D) = \varphi(d) \wedge (\psi(d) \vee X(f(d)))$ be an equation, where $f{:}D \to D$ is an arbitrary total function and X does not occur in φ and ψ.*
(1) *The solution to X for $\sigma = \nu$ is*
$\forall j{:}\mathbb{N}.((\forall i{:}\mathbb{N}.i < j \to \neg\psi(f^i(d))) \to \varphi(f^j(d)))$,
(2) *The solution to X for $\sigma = \mu$ is*:
$\exists i{:}\mathbb{N}.\psi(f^i(d)) \wedge \forall j{:}\mathbb{N}.(j \leqslant i \to \varphi(f^j(d)))$.

Proof. We first deal with $\sigma = \nu$. We prove this theorem by directly, but transfinitely, calculating the fixpoint (Lemma 9). The finite solutions are given by the following formula:

$$X_n(d{:}D) = \bigwedge_{j=0}^{n-1} \left(\left(\bigwedge_{i=0}^{j-1} \neg\psi(f^i(d)) \right) \to \varphi(f^j(d)) \right).$$

It is easy to show that X_n is the nth approximation of X using induction on n. The next approximation $X_\omega(d)$ is equal to the maximal solution and given by

$$\begin{aligned} X_\omega(d{:}D) &= \forall n{:}\mathbb{N}.X_n(d) \\ &= \forall n{:}\mathbb{N}. \textstyle\bigwedge_{j=0}^{n-1} \left(\left(\bigwedge_{i=0}^{j-1} \neg\psi(f^i(d)) \right) \to \varphi(f^j(d)) \right) \\ &= \forall j{:}\mathbb{N}. \left(\left(\textstyle\bigwedge_{i=0}^{j-1} \neg\psi(f^i(d)) \right) \to \varphi(f^j(d)) \right) \\ &= \forall j{:}\mathbb{N}.(\forall i{:}\mathbb{N}.i < j \to \neg\psi(f^i(d))) \to \varphi(f^j(d)). \end{aligned}$$

It only remains to be shown that the solution is stable, which can be seen as follows:

$$\begin{aligned} &\varphi(d) \wedge (\psi(d) \vee X_\omega(f(d))) \\ &\quad = \varphi(d) \wedge (\psi(d) \vee \forall j{:}\mathbb{N}.(\forall i{:}\mathbb{N}.i < j \to \neg\psi(f^{i+1}(d))) \to \varphi(f^{j+1}(d))) \\ &\quad = \varphi(d) \wedge (\neg\psi(d) \to (\forall j{:}\mathbb{N}.j > 0 \to (\forall i{:}\mathbb{N}.1 \leqslant i < j \to \neg\psi(f^i(d))) \to \varphi(f^j(d)))) \\ &\quad = \forall j{:}\mathbb{N}.((\forall i{:}\mathbb{N}.i < j \to \neg\psi(f^i(d))) \to \varphi(f^j(d))) \\ &\quad = X_\omega(d). \end{aligned}$$

The proof for $\sigma = \mu$ follows the same lines. The finitary approximations are given by

$$X_n(d{:}D) = \bigvee_{i=0}^{n-1} (\psi(f^i(d)) \wedge \bigwedge_{j=0}^{i} \varphi(f^j(d))).$$

The first infinitary approximation is calculated as follows

$$\begin{aligned}
X_\omega(d{:}D) &= \exists\, n{:}\mathbb{N}.X_n(d)\\
&= \exists\, n{:}\mathbb{N}.\ \textstyle\bigvee_{i=0}^{n-1}(\psi(f^i(d)) \wedge \bigwedge_{j=0}^{i} \varphi(f^j(d)))\\
&= \exists\, i{:}\mathbb{N}.(\psi(f^i(d)) \wedge \textstyle\bigwedge_{j=0}^{i} \varphi(f^j(d)))\\
&= \exists\, i{:}\mathbb{N}.(\psi(f^i(d)) \wedge \forall j{:}\mathbb{N}.(j \leqslant i \to \varphi(f^j(d)))).
\end{aligned}$$

Showing that $X_\omega(d)$ is stable goes in the following way:

$$\begin{aligned}
&\varphi(d) \wedge (\psi(d) \vee X_\omega(f(d)))\\
&\quad = \varphi(d) \wedge (\psi(d) \vee \exists i{:}\mathbb{N}.(\psi(f^{i+1}(d))) \wedge \forall j{:}\mathbb{N}.(j \leqslant i \to \varphi(f^{j+1}(d))))\\
&\quad = (\psi(d) \wedge \varphi(d)) \vee \exists i{:}\mathbb{N}.(\psi(f^{i+1}(d)) \wedge \forall j{:}\mathbb{N}.(j \leqslant i{+}1 \to \varphi(f^j(d))))\\
&\quad = \exists i{:}\mathbb{N}.(\psi(f^i(d)) \wedge \forall j{:}\mathbb{N}.(j \leqslant i \to \varphi(f^j(d))))\\
&\quad = X_\omega(d).
\end{aligned}$$

The first pattern above immediately provides us with the solution to the equation of Example 35, by taking the function $f{:}\mathbb{N}\to\mathbb{N}$, defined as $f(i) = i + 1$, and defining the predicate $\varphi(i) = i \leqslant N$ and $\psi(i) = \bot$.

When more than one occurrence of X occurs in the right-hand side of the pattern in Theorem 36 we have a straightforward generalisation for which we can find a solution in a similar vein.

In this case we assume that functions $f_i{:}D \to D$ for $i < N$ for some given N are given. We let $g : \mathbb{N} \to \{0, \ldots, N-1\}$ be an arbitrary function. We assume the existence of functions $f(g, j, d)$ with the property that $f(g, 0, d) = d$ and $f(g, j + 1, d) = f_{g(j)}(f(g, j, d))$. $\square$

Theorem 37. *Let $N{:}\mathbb{N}$ be some arbitrary natural number and let*

$$\sigma X(d{:}D) = \varphi(d) \wedge \bigwedge_{i=0}^{N-1} (\psi_i(d) \vee X(f_i(d)))$$

be an equation, where $f_i{:}D\to D$ are arbitrary total functions and X does not occur in φ and ψ_i.

(1) *The solution to X for $\sigma = \nu$ is*
$\forall j{:}\mathbb{N}.\forall g{:}\mathbb{N}\to\{0, \ldots, N-1\}.((\forall i{:}\mathbb{N}.i<j\to\neg\psi_{g(i)}(f(g, i, d)))\to\varphi(f(g, j, d)))$,

(2) *The solution to X for $\sigma = \mu$ is*
$\exists j{:}\mathbb{N}.\exists g{:}\mathbb{N}\to\{0, \ldots, N-1\}.((\forall i{:}\mathbb{N}.i<j\to\neg\psi_{g(i)}(f(g, i, d))) \wedge \varphi(f(g, j, d)))$,

Proof. We exactly follow the structure of the proofs of Theorem 36 and we provide only the proof for $\sigma = \nu$ here. First we define the finitary approximations:

$$X_n(d) = \forall g{:}\mathbb{N}\to\{0, \ldots, N-1\}.\bigwedge_{j=0}^{n-1}\left(\left(\bigwedge_{k=0}^{j-1}\neg\psi_{g(i)}(f(g, k, d))\right)\to\varphi(f(g, j, d))\right).$$

In order to see that $X_n(d)$ is the nth approximation observe that

$$X_0(d) = \top$$

and

$$\begin{aligned}
&\varphi(d) \wedge \textstyle\bigwedge_{i=0}^{N-1}(\psi_i(d) \vee X_n(f_i(d)))\\
&\quad = \varphi(d) \wedge \textstyle\bigwedge_{i=0}^{N-1}\Big(\psi_i(d)\vee\\
&\qquad\qquad \forall g.\textstyle\bigwedge_{j=0}^{n-1}\Big(\Big(\bigwedge_{k=0}^{j-1}\neg\psi_{g(k)}(f(g,i,f_i(d)))\Big)\rightarrow\varphi(f(g,j,f_i(d)))\Big)\Big)\\
&\quad = \forall g.\bigwedge_{i=0}^{N-1}\varphi(d) \wedge \Big(\psi_i(d)\vee\\
&\qquad\qquad \textstyle\bigwedge_{j=0}^{n-1}\Big(\Big(\bigwedge_{k=0}^{j-1}\neg\psi_{g(k)}(f(g,k,f_i(d))\Big)\rightarrow\varphi(f(g,j,f_i(d)))\Big)\Big)\\
&\quad =^{*} \forall h.\varphi(d) \wedge \Big(\psi_{g(k)}(d)\vee\\
&\qquad\qquad \textstyle\bigwedge_{j=0}^{n-1}\Big(\Big(\bigwedge_{k=0}^{j-1}\neg\psi_{h(k+1)}(f(h,k{+}1,d))\Big)\rightarrow\varphi(f(n,j{+}1,d))\Big)\Big)\\
&\quad = \forall h.\textstyle\bigwedge_{j=0}^{n-1}\varphi(d) \wedge \Big(\psi_{h(0)}(d)\vee\\
&\qquad\qquad \Big(\Big(\textstyle\bigwedge_{k=0}^{j-1}\neg\psi_{h(k+1)}(f(h,k{+}1,d))\Big)\rightarrow\varphi(f(h,j{+}1,d))\Big)\Big)\Big)\\
&\quad = \forall h.\textstyle\bigwedge_{j=0}^{n-1}\varphi(d) \wedge \Big(\Big(\bigwedge_{k=0}^{j}\neg\psi_{h(k)}(f(h,k,d))\Big)\rightarrow\varphi(f(h,j{+}1,d))\Big)\\
&\quad = \forall h.\textstyle\bigwedge_{j=0}^{n-1}\varphi(d) \wedge \Big(\Big(\bigwedge_{k=0}^{j}\neg\psi_{h(k)}(f(h,k,d))\Big)\rightarrow\varphi(f(h,j{+}1,d))\Big)\\
&\quad = \forall h.\textstyle\bigwedge_{j=1}^{n}\varphi(d) \wedge \Big(\Big(\bigwedge_{k=0}^{j-1}\neg\psi_{h(k)}(f(h,k,d))\Big)\rightarrow\varphi(f(h,j,d))\Big)\\
&\quad = \forall h.\textstyle\bigwedge_{j=0}^{n}\Big(\Big(\bigwedge_{k=0}^{j-1}\neg\psi_{h(k)}(f(h,k,d))\Big)\rightarrow\varphi(f(h,j,d))\Big)\\
&\quad = X_{n+1}(d),
\end{aligned}$$

where at $*$ we introduce $h{:}\mathbb{N}\rightarrow\{0,\ldots,N-1\}$ such that $i = h(0)$ and $g(l) = h(l+1)$ for all l. The universally bound function g above and below has type $g{:}\mathbb{N}\rightarrow\{0,\ldots,N-1\}$.

Next we calculate the first infinitary approximation, which happens to be equal to the solution of the equation.

$$\begin{aligned}
X_\omega(d) &= \forall n{:}\mathbb{N}.X_n(d)\\
&= \forall n{:}\mathbb{N}.\forall g.\textstyle\bigwedge_{j=0}^{n-1}\Big(\Big(\bigwedge_{k=0}^{j-1}\neg\psi_{g(k)}(f(g,k,d))\Big)\rightarrow\varphi(f(g,j,d))\Big)\\
&= \forall j{:}\mathbb{N}.\forall g.((\forall k{:}\mathbb{N}.(k<j\rightarrow\neg\psi_{g(k)}(f(g,k,d))))\rightarrow\varphi(f(g,j,d))).
\end{aligned}$$

Finally, we show that the first infinitary approximation is stable, which proves that it is indeed the maximal fixpoint solution for this equation.

$$\begin{aligned}
&\varphi(d) \wedge \textstyle\bigwedge_{i=0}^{N-1}(\psi_i(d) \vee X_\omega(f_i(d)))\\
&\quad = \varphi(d) \wedge \textstyle\bigwedge_{i=0}^{N-1}(\psi_i(d) \vee\\
&\qquad\qquad \forall j{:}\mathbb{N}.\forall g.((\forall k{:}\mathbb{N}.(k<j\rightarrow\neg\psi_{g(k)}(f(g,k,f(d)))))\rightarrow\varphi(f(g,j,f_i(d)))))\\
&\quad = \forall g.\textstyle\bigwedge_{i=0}^{N-1}\varphi(d) \wedge (\psi_i(d) \vee
\end{aligned}$$

$$
\begin{aligned}
&\qquad \forall j{:}\mathbb{N}.((\forall k{:}\mathbb{N}.(k<j\rightarrow\neg\psi_{g(k)}(f(g,k,f_i(d)))))\rightarrow\varphi(f(g,j,f_i(d)))))\\
&=^{\star\star} \forall h.\varphi(d)\wedge(\psi_{h(0)}(d)\vee\\
&\qquad \forall j{:}\mathbb{N}.((\forall k{:}\mathbb{N}.(k<j\rightarrow\neg\psi_{h(k+1)}(f(g,k+1,d))))\rightarrow\varphi(f(h,j+1,d))))\\
&=\forall h.\forall j{:}\mathbb{N}.\varphi(d)\wedge((\forall k{:}\mathbb{N}.(k<j+1\rightarrow\neg\psi_{h(k)}(f(g,k,d))))\rightarrow\varphi(f(h,j+1,d)))\\
&=\forall h.\forall j{:}\mathbb{N}.(\forall k{:}\mathbb{N}.(k<j\rightarrow\neg\psi_{h(k)}(f(g,k,d))))\rightarrow\varphi(f(h,j,d))\\
&=X_\omega(d).
\end{aligned}
$$

See for the $\star\star$ the remark marked with the $*$ above. □

The patterns that we considered in this section are inspired by the examples in Section 5. We expect that these will be encountered very often when solving parameterised boolean equation systems that will occur when proving the validity of modal formulas on large examples. We actually think that it will be fruitful to build a library of patterns and include these in tools that automatically solve boolean parameterised boolean equation systems. This has for instance been done in computer algebra systems with mathematical formulae.

However, finding and in particular solving these patterns might turn out to be difficult. A pattern that we encountered but were not able to solve thus far is the following:

$$\sigma X(d{:}D)=\varphi(d)\wedge\forall e{:}E.\psi(d,e)\vee X(f(d,e)))$$

for arbitrary data sort E. Actually—and we pose this as a very interesting open question—it might very well be possible to devise a method to solve all single fixed point equations of the form $\sigma X(d{:}D)=\varphi$ by replacing φ by a first order formula in which X does not occur. Using Gauß elimination, this would yield a complete method that allows to transform each parameterised boolean equation system to a first order formula. Solving the equation system would then be equivalent to determine whether the formula is a tautology. The advantage of this transformation is that it moves the relatively unknown field of model checking (with data) and equation systems to the well studied field of first order logic.

4.2.4. Invariants

Invariants characterise 'the reachable parameter space' of a parameterised boolean equation. As in the verification of programs they can be used to prove properties that only hold within the reachable state space. Within parameterised boolean equation systems they can be used to simplify equations with a particular parameter instantiation.

A formal definition of an invariant is given below. In our setting the definition looks uncommon, but still expresses what is ordinarily understood as an invariant. Note that our invariants only have the transfer property, and do not involve an initial state.

Definition 38 (*Invariant*). Let $\sigma X(d{:}D)=\varphi$ be an equation and let $I{:}D\rightarrow\mathbb{B}$ be a predicate formula in which no predicate variable occurs. Then, I is an invariant of X iff

$$(I\wedge\varphi)\leftrightarrow(I\wedge\varphi[(\lambda e{:}D.I[e/d]\wedge X(e))/X]).$$

Basically, a predicate formula is an invariant iff, for that part of the parameter space of the equation for which the invariant holds, the solution is not changed by adding the invariant.

Note that in general this affects the solution of the equation, as the solution with and without the invariant only coincide in those situations for which the invariant holds. Nevertheless, invariants can be used for simplifying an equation system by calculating with the equation system in which the invariant is used, as expressed by the following theorem. First an auxiliary lemma is proven, and subsequently the invariance rule is given that indicates how invariants can be used.

Lemma 39. *Let $\sigma X(d{:}D) = \varphi$ and $\sigma Y(d{:}D) = I(d) \wedge \varphi[Y/X]$ be equations such that $Y \notin \mathrm{occ}(\varphi)$ and let $I{:}D \to \mathbb{B}$ be an invariant of X. For all $d{:}D$ for which $I(d)$ is valid, it holds that*

$$(\sigma X(d{:}D).\varphi(\eta))(d) = (\sigma Y(d{:}D).(I(d) \wedge \varphi[Y/X])(\eta))(d).$$

Proof. We prove this lemma by a transfinite approximation (see Lemma 9). So, we let X_α and Y_α be the αth approximation for X and Y respectively, where α is an ordinal, and we show that $I(d)$ implies

$$X_\alpha(d) = Y_\alpha(d).$$

We find:

- For $\alpha = 0$, we must distinguish between $\sigma = \nu$ and $\sigma = \mu$. If $\sigma = \nu$ it holds that $X_0(d) = Y_0(d) = \top$. For $\sigma = \mu$ we find that $X_0(d) = Y_0(d) = \bot$.
- For $\alpha = \beta + 1$ a successor ordinal we find under the assumption that $I(d)$ holds:

$$\begin{array}{rcl} Y_{\beta+1}(d) & = & \varphi(Y_\beta(d)) \\ & \overset{\text{invariant}}{=} & \varphi(I(d) \wedge Y_\beta(d)) \\ & \overset{\text{i.h.}}{=} & \varphi(I(d) \wedge X_\beta(d)) \\ & \overset{\text{invariant}}{=} & \varphi(X_\beta(d)) \\ & = & X_{\beta+1}(d). \end{array}$$

- For α a limit ordinal and $\sigma = \mu$ we find

$$Y_\alpha(d) = \bigvee_{\beta<\alpha} Y_\beta(d) \overset{\text{i.h.}}{=} \bigvee_{\beta<\alpha} X_\beta(d) = X_\alpha(d).$$

  The case with $\sigma = \nu$ is dual.

So, we have shown that $X_\alpha(d) = Y_\alpha(d)$. Now, as we know that X_α and Y_α are the minimal/maximal solutions for a sufficiently large α, the lemma follows. □

Theorem 40 (*Invariance rule*). *Let $\sigma X(d{:}D) = \varphi$ be an equation such that $Y \notin \mathrm{occ}(\varphi)$ for some predicate variable Y and let $I{:}D \to \mathbb{B}$ be an invariant of X. Then*

$$\begin{array}{l} \big(\sigma X(d{:}D) = \varphi\big)\big(\sigma Y(d{:}D) = I(d) \wedge \varphi[Y/X]\big) \\ \quad \equiv \big(\sigma X(d{:}D) = (I(d) \wedge Y(d)) \vee (\neg I(d) \wedge \varphi)\big)\big(\sigma Y(d{:}D) = I(d) \wedge \varphi[Y/X]\big). \end{array}$$

Proof. We write $\mathcal{B}$ for $\sigma Y(d{:}D) = I(d) \wedge \varphi[Y/X]$. According to Definition 13 we must show for all η and $\mathcal{F}$:

$$[(\sigma X(d{:}D) = \varphi)\mathcal{B}\mathcal{F}]\eta = [(\sigma X(d{:}D) = (I(d) \wedge Y(d)) \vee (\neg I(d) \wedge \varphi))\mathcal{B}\mathcal{F}]\eta.$$

By Definition 3 this is equivalent to:

$$\begin{aligned}&[\mathcal{BF}]\eta[\sigma X(d{:}D).\varphi([\mathcal{BF}]\eta)/X]\\ &\quad = [\mathcal{BF}]\eta[\sigma X(d{:}D).((I(d)\wedge Y(d))\vee(\neg I(d)\wedge\varphi))([\mathcal{BF}]\eta)/X].\end{aligned}$$

This in turn follows from

$$\sigma X(d{:}D).\varphi([\mathcal{BF}]\eta) = \sigma X(d{:}D).((I(d)\wedge Y(d))\vee(\neg I(d)\wedge\varphi))([\mathcal{BF}]\eta).$$

Distribution of the substitution leads to

$$\sigma X(d{:}D).\varphi([\mathcal{BF}]\eta) = \sigma X(d{:}D).((I(d)\wedge(Y(d)([\mathcal{BF}]\eta)))\vee\neg(I(d)\wedge(\varphi([\mathcal{BF}]\eta)))).$$

Because Y does not occur in φ, this subsequently reduces to

$$\sigma X(d{:}D).\varphi([\mathcal{F}]\eta) = \sigma X(d{:}D).((I(d)\wedge(Y(d)([\mathcal{BF}]\eta)))\vee(\neg I(d)\wedge(\varphi([\mathcal{F}]\eta)))). \tag{5}$$

It holds that $Y(d)([\mathcal{BF}]\eta)$ equals $(\sigma Y(d{:}D).(I(d)\wedge\varphi[Y/X])([\mathcal{F}]\eta))(d)$. Using Lemma 39 this is equal to $I(d)\wedge(\sigma X(d{:}D).\varphi([\mathcal{F}]\eta))(d)$. So, Eq. (5) is equal to

$$\begin{aligned}&\sigma X(d{:}D).\varphi([\mathcal{F}]\eta)\\ &\quad = \sigma X(d{:}D).(I(d)\wedge(\sigma X(d{:}D).\varphi([\mathcal{F}]\eta))(d))\vee\neg I(d)\wedge(\varphi([\mathcal{F}]\eta))).\end{aligned}$$

Now observe that the right-hand side of this equation equals the left-hand side, except that the solution for X has been partially substituted. Using Lemma 10 both sides are equal. □

A disadvantage of the previous theorem is that it requires an extra equation. Therefore, we provide a theorem below that allows the use of an invariant without this additional equation. But first an auxiliary lemma is given:

Lemma 41. *Let $\sigma X(d{:}D)=\varphi$ and $\sigma' Y(e{:}E)=\psi$ be equations and let $I{:}D\to\mathbb{B}$ be an invariant of X. Let $\mathcal{K}$ be a parameterised boolean equation system. If for some predicate formula χ with $X\notin \mathrm{occ}(\chi)$*
(1) $(\sigma X(d{:}D)=\varphi\wedge I(d))\equiv(\sigma X(d{:}D)=\chi)$,
(2) $(\sigma' Y(e{:}E)=\psi)\equiv(\sigma' Y(e{:}E)=\psi[\lambda d{:}D.I(d)\wedge X(d)/X])$ *and*
(3) $\sigma X(d{:}D)=\varphi$ *is in* $\mathcal{K}$,
then

$$\sigma' Y(e{:}E).\psi[\mathcal{K}]\eta = \sigma' Y(e{:}E).\psi[\lambda d{:}D.\chi/X][\mathcal{K}]\eta.$$

Proof. This lemma is proven with induction on the length of $\mathcal{K}$. If $\mathcal{K}$ is empty, $\sigma X(d{:}D)=\varphi$ cannot occur in $\mathcal{K}$ and the lemma holds as Condition 3 is invalid.

If $\mathcal{K}$ is not empty, we distinguish two cases:
(1) $\mathcal{K}$ equals $(\sigma X(d{:}D)=\varphi)\mathcal{K}'$. So, we must show:

$$\begin{aligned}&\sigma' Y(e{:}E).\psi[(\sigma X(d{:}D)=\varphi)\mathcal{K}']\eta\\ &\quad = \sigma' Y(e{:}E).\psi[\mathcal{K}']\eta[\sigma X(d{:}D).\varphi([\mathcal{K}']\eta)/X]\\ &\quad \overset{\star}{=} \sigma' Y(e{:}E).\psi[\lambda d{:}D.I(d)\wedge\sigma X(d{:}D).\varphi([\mathcal{K}']\eta)(d)/X][\mathcal{K}']\eta[\sigma X(d{:}D).\varphi([\mathcal{K}']\eta)/X]\end{aligned}$$

$$
\begin{aligned}
&=^{\star\star} \sigma' Y(d{:}D).\psi[\sigma X(d{:}D).(I(d) \wedge \varphi)([\mathcal{K}']\eta)/X][\mathcal{K}']\eta[\sigma X(d{:}D).\varphi([\mathcal{K}']\eta)/X] \\
&=^{\star\star\star} \sigma' Y(e{:}E).\psi[\sigma X(d{:}D).\chi([\mathcal{K}']\eta)/X][\mathcal{K}']\eta[\sigma X(d{:}D).\varphi([\mathcal{K}']\eta)/X] \\
&= \sigma' Y(e{:}E).\psi[\chi/X][\mathcal{K}']\eta[\sigma X(d{:}D).\varphi([\mathcal{K}']\eta)/X] \\
&= \sigma' Y(e{:}E).\psi[\chi/X]([(\sigma X(d{:}D) = \varphi)\mathcal{K}']\eta)
\end{aligned}
$$

At $\star$ Condition 2 is used. At $\star\star$ Lemma 39 is used. At $\star\star\star$ Condition 1 is used.

(2) $\mathcal{K}$ equals $(\sigma'' Z(f{:}F) = \xi)\mathcal{K}'$ with $Z \neq X$. We find

$$
\begin{aligned}
&\sigma' Y(e{:}E).\psi[(\sigma'' Z(f{:}F) = \xi)\mathcal{K}']\eta \\
&\quad = \sigma' Y(e{:}E).\psi[\mathcal{K}']\eta[\sigma'' Z(f{:}F).\xi([\mathcal{K}']\eta)/Z] \\
&\quad =^{\star} \sigma' Y(e{:}E).\psi[\chi/X][\mathcal{K}']\eta[\sigma'' Z(f{:}F).\xi([\mathcal{K}']\eta)/Z] \\
&\quad = \sigma' Y(e{:}E).\psi[\chi/X][(\sigma'' Z(f{:}F) = \xi)\mathcal{K}']\eta.
\end{aligned}
$$

At $\star$ we use the induction hypothesis.

The theorem below says that if χ is a solution for the equation $\sigma X(d{:}D) = \varphi$ under invariant I (Condition 1) and X is used in an equation $\sigma' Y(e{:}E) = \psi$ in a situation where I implies X (Condition 2), then we may substitute solution χ for X in ψ. □

Theorem 42. *Let $\sigma X(d{:}D) = \varphi$ and $\sigma' Y(e{:}E) = \psi$ be equations and let $I{:}D \to \mathbb{B}$ be an invariant of X. Let $\mathcal{E}$ be a parameterised boolean equation system such that $\{X, Y\} \nsubseteq \mathrm{bnd}(\mathcal{E})$. If for some predicate formula χ such that $X \notin \mathrm{occ}(\chi)$*
(1) $(\sigma X(d{:}D) = \varphi \wedge I(d)) \equiv (\sigma X(d{:}D) = \chi)$ and
(2) $(\sigma' Y(e{:}E) = \psi) \equiv (\sigma' Y(e{:}E) = \psi[\lambda d{:}D.I(d) \wedge X(d)/X])$.
then

$$
\begin{aligned}
&(\sigma' Y(e{:}E) = \psi)\mathcal{E}(\sigma X(d{:}D) = \varphi) \\
&\quad \equiv (\sigma' Y(e{:}E) = \psi[\lambda d{:}D.\chi/X])\mathcal{E}(\sigma X(d{:}D) = \varphi).
\end{aligned}
$$

Proof. By definition we must show for all $\mathcal{F}$ and η that

$$
\begin{aligned}
&[(\sigma' Y(e{:}E) = \psi)\mathcal{E}(\sigma X(d{:}D) = \varphi)\mathcal{F}]\eta \\
&\quad = ((\sigma' Y(e{:}E) = \psi[\lambda d{:}D.\chi/X])\mathcal{E}(\sigma X(d{:}D) = \varphi)\mathcal{F}]\eta.
\end{aligned}
$$

Abbreviate $\mathcal{E}(\sigma X(d{:}D) = \varphi)\mathcal{F}$ with $\mathcal{K}$. We can rewrite the previous equation to

$$[\mathcal{K}]\eta[\sigma' Y(e{:}E).\psi[\mathcal{K}]\eta/Y] = [\mathcal{K}]\eta[\sigma' Y(e{:}E).\psi[\lambda d{:}D.\chi/X][\mathcal{K}]\eta/Y],$$

which follows from

$$\sigma' Y(e{:}E).\psi[\mathcal{K}]\eta = \sigma' Y(e{:}E).\psi[\lambda d{:}D.\chi/X][\mathcal{K}]\eta,$$

which matches the conclusion of Lemma 41. □

5. Applications

In this section, we study properties of several small but characteristic reactive systems. Note that, although the systems that we study are small in size, their behaviours are in many cases quite complex.

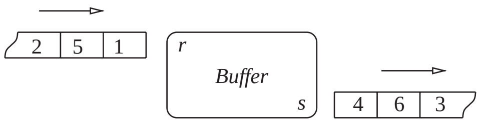

Fig. 1. A one-place buffer system.

We study the systems by proving the validity of certain modal formulas governing their behaviour. We translate the process descriptions and the formulas to parameterised boolean equation systems that are subsequently solved. For a detailed account on how these equations can be derived from a process and a formula, we refer to [11,15,26]. For the remainder of this paper, we assume the reader is familiar with the use of the specification language μCRL[13,14], and the use of the *first-order modal* μ*-calculus* with data [11,15] to specify logical properties of systems. We use natural numbers as the main data type in the examples as natural numbers are very common. More complex data types can be used similarly.

5.1. A one-place buffer

The first system we study is a one-place buffer. We study two properties that are not commonly studied on buffers, namely that if the input stream of the buffer consists of identical values, the output stream also consists of identical values and if the input stream is increasing, then the output stream is also increasing. These properties need data in modal logic to be expressed.

The buffer is represented by the μCRL process *Buffer* (see below). It reads natural numbers one-by-one from an infinite stream using action *r*, and it outputs a stream of data using action *s* (see Fig. 1).

$$\mathbf{proc}\ \mathit{Buffer}(b{:}\mathbb{B}, n{:}\mathbb{N}) = \sum_{m:\mathbb{N}} r(m) \cdot \mathit{Buffer}(\bot, m) \lhd b \rhd s(n) \mathit{Buffer}(\top, n),$$

where the initial state is $\mathit{Buffer}(\top, n)$ for an arbitrary $n \in \mathbb{N}$.

5.1.1. A constant input stream

The first property we set out to investigate is the following: provided that the input is a stream of the form k^{ω}, for some natural number *k*, then the output is also of the form k^{ω}. In other words, the buffer does not perform any transformations on its input when this is a constant input stream.

The property requires keeping track of the value that appears in the input stream. It is expressed by the following formula. We use fixpoint variables with a tilde ($\tilde{X}$) to stress the difference with variables in equation systems:

$$\forall k{:}\mathbb{N}.(\nu \tilde{X}.\forall l{:}\mathbb{N}.[r(l)](l{=}k \to \tilde{X}) \land [s(l)](l = k \land \tilde{X})).$$

The property and the process can in the standard way be translated to an equation system. The property holds if $\forall k{:}\mathbb{N}.X(b,n,k)$ holds where X is given by

$$\begin{aligned}\nu X(b{:}\mathbb{B}, n, k{:}\mathbb{N}) = \forall l{:}\mathbb{N}.\ &(\forall m{:}\mathbb{N}.(b \wedge m{=}l \rightarrow (l{=}k \rightarrow X(\bot, m, k)))\wedge \\ &(\neg b \wedge l{=}n \rightarrow (l = k \wedge X(\top, n, k)))).\end{aligned}$$

We can eliminate the quantifiers by substitution. We get using Corollary 29:

$$\nu X(b{:}\mathbb{B}, n, k{:}\mathbb{N}) = (b \rightarrow X(\bot, k, k)) \wedge (\neg b \rightarrow (n = k \wedge X(\top, n, k))).$$

This equation can be solved using a simple approximation, where X_i denotes the ith approximation.

$$\begin{aligned}X_0(b,n,k) &= \top, \\ X_1(b,n,k) &= \neg b \rightarrow n{=}k, \\ X_2(b,n,k) &= (b \rightarrow (\neg\bot \rightarrow k{=}k)) \wedge (\neg b \rightarrow (n{=}k \wedge (\neg\top \rightarrow n{=}k))) \\ &= \neg b \rightarrow n = k.\end{aligned}$$

As $X_1(b,n,k)$ is stable, we found the solution. So, a buffer preserves a constant input stream if $\forall k{:}\mathbb{N}.(\neg b \rightarrow n = k)$, which is equivalent to b, which is indeed what could be expected.

5.1.2. An ascending input stream

The second property we study is the following. If the input stream is ascending, is the produced stream also ascending? This property can be expressed using two variables to remember the last read input and the last produced output. It is formalised by the following modal formula:

$$(\nu\tilde{X}(in, out{:}\mathbb{N}).\forall l{:}\mathbb{N}.([r(l)](l \geqslant in \rightarrow \tilde{X}(l, out)) \wedge [s(l)](l \geqslant out \wedge \tilde{X}(in, l))))(0,0).$$

The ascending stream property holds on the process *Buffer* if $X(b,n,0,0)$ holds where X is given by:

$$\begin{aligned}\nu X(b{:}\mathbb{B}, n, in, out{:}\mathbb{N}) = \forall l{:}\mathbb{N}.\ &(\forall m{:}\mathbb{N}.(b \wedge l{=}m \rightarrow (l \geqslant in \rightarrow X(\bot, m, l, out)))\wedge \\ &(\neg b \wedge l{=}n \rightarrow (l \geqslant out \wedge X(\top, n, in, l))))).\end{aligned}$$

The right-hand side of this fixpoint equation can be simplified using laws of predicate logic. So, with Corollary 29 we find:

$$\begin{aligned}\nu X(b{:}\mathbb{B}, n, in, out{:}\mathbb{N}) = \forall l{:}\mathbb{N}.&(b \rightarrow (l \geqslant in \rightarrow X(\bot, l, l, out)))\wedge \\ &(\neg b \rightarrow (n \geqslant out \wedge X(\top, n, in, n))).\end{aligned}$$

The approximation of this equation is straightforward:

$$\begin{aligned}X_0(b,n,in,out) &= \top, \\ X_1(b,n,in,out) &= \neg b \rightarrow n \geqslant out, \\ X_2(b,n,in,out) &= \forall l{:}\mathbb{N}.(b \rightarrow (l \geqslant in \rightarrow l \geqslant out)) \wedge (\neg b \rightarrow n \geqslant out), \\ &= (b \rightarrow in \geqslant out) \wedge (\neg b \rightarrow n \geqslant out) \\ X_3(b,n,in,out) &= \forall l{:}\mathbb{N}.(b \rightarrow (l \geqslant in \rightarrow l \geqslant out)) \wedge (\neg b \rightarrow (n \geqslant out \wedge in \geqslant n)) \\ &= (b \rightarrow in \geqslant out) \wedge (\neg b \rightarrow in \geqslant n \wedge n \geqslant out), \\ X_4(b,n,in,out) &= \forall l{:}\mathbb{N}.(b \rightarrow (l \geqslant in \rightarrow l \geqslant l \wedge l \geqslant out)) \wedge (\neg b \rightarrow in \geqslant n \wedge n \geqslant out)) \\ &= (b \rightarrow in \geqslant out) \wedge (\neg b \rightarrow in \geqslant n \wedge n \geqslant out).\end{aligned}$$

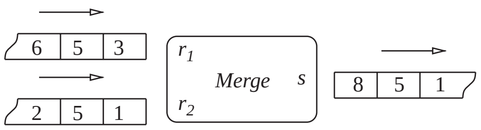

Fig. 2. Combining two input streams into a single output stream.

Note that $X_3(b, n, in, out)$ is stable. Therefore it is the solution of the fixpoint equation. So, the ascending chain property holds if $X(b, n, 0, 0)$ is valid. By substituting the solution of X, this boils down to $\neg b \rightarrow n{=}0$.

5.2. Merging infinite streams

Combining several input streams into a single stream is a technique that is found frequently in streaming media applications. The way streams are combined depends on a particular application. Here, we study a small system that reads data from two (infinite) input streams, one-by-one, and produces a new output stream that is locally ascending, see Fig. 2. Our particular merge system is described by the four process equations below. The initial process is *Merge*. It reads data from stream i via action r_i, where $i \in \{1, 2\}$, and the output is produced via action s:

$$\begin{aligned}
&\mathit{Merge} = \textstyle\sum_{m:\mathbb{N}}(r_1(m) \cdot \mathit{Merge}_1(m) + r_2(m) \cdot \mathit{Merge}_2(m)),\\
&\mathit{Merge}_1(n{:}\mathbb{N}) = \textstyle\sum_{m:\mathbb{N}} r_2(m) \cdot \mathit{Merge}_3(n, m),\\
&\mathit{Merge}_2(m{:}\mathbb{N}) = \textstyle\sum_{n:\mathbb{N}} r_1(n) \cdot \mathit{Merge}_3(n, m),\\
&\mathit{Merge}_3(n, m{:}\mathbb{N}) = s(n) \cdot \mathit{Merge}_2(m) \lhd n \leqslant m \rhd s(m) \cdot \mathit{Merge}_1(n).
\end{aligned}$$

To illustrate its behaviour, consider the input streams as depicted in Fig. 2, and ignore the output stream that is depicted. On this input stream, it first reads the values 3 and 1 in random order, via actions r_1 and r_2, respectively. Since $1 \leqslant 3$, the value 1 is produced as output via action s and value 2 is read from input stream 2, and produced as output, since $2 \leqslant 3$. Subsequently, value 5 is read from stream 2 and value 3 is produced as output, after which value 5 is read from input stream 1. Now, the merge process decides non-deterministically from which of the two streams it reads next, and it outputs value 5.

Clearly, on ascending input streams, the merge system should produce an ascending output. This is expressed by the following formula:

$$\begin{aligned}
(\nu \tilde{X}(in_1, in_2, out{:}\mathbb{N}).\forall l{:}\mathbb{N}.\ (&[r_1(l)](l \geqslant in_1 \rightarrow \tilde{X}(l, in_2, out))\wedge\\
&[r_2(l)](l \geqslant in_2 \rightarrow \tilde{X}(in_1, l, out))\wedge\\
&[s(l)](l \geqslant out \wedge \tilde{X}(in_1, in_2, l))))(0, 0, 0).
\end{aligned}$$

Note that the process *Merge* must first be converted to linear form if we are to verify this property. This is fairly straightforwardly achieved by introducing an additional parameter $\sigma{:}\mathbb{N}$. Process Merge_i is represented by $\sigma = i$, whereas $\sigma = 0$ represents process *Merge* itself. Combining the resulting linear process specification with the above formula according to the translation of [11,15,26] and after applying some simplifications, we obtain the following equation:

$$\begin{aligned}
&\nu X(\sigma, n, m, in_1, in_2, out{:}\mathbb{N})\\
&\quad = (\sigma = 0 \rightarrow (\forall l{:}\mathbb{N}. l \geqslant in_1 \rightarrow X(1, l, m, l, in_2, out))) \wedge\\
&\qquad (\sigma = 0 \rightarrow (\forall l{:}\mathbb{N}. l \geqslant in_2 \rightarrow X(2, n, l, in_1, l, out))) \wedge\\
&\qquad (\sigma = 1 \rightarrow (\forall l{:}\mathbb{N}. l \geqslant in_2 \rightarrow X(3, n, l, in_1, l, out))) \wedge\\
&\qquad (\sigma = 2 \rightarrow (\forall l{:}\mathbb{N}. l \geqslant in_1 \rightarrow X(3, l, m, l, in_2, out))) \wedge\\
&\qquad (\sigma = 3 \wedge n \leqslant m) \rightarrow (n \geqslant out \wedge X(2, n, m, in_1, in_2, n)) \wedge\\
&\qquad (\sigma = 3 \wedge m \leqslant n) \rightarrow (m \geqslant out \wedge X(1, n, m, in_1, in_2, m)),
\end{aligned}$$

where the ascending input/output property holds if $X(\sigma, n, m, 0, 0, 0)$ holds.

A closer inspection of the equation reveals a striking similarity in the use of the variables n and in_1, and, likewise, in the variables m and in_2. This is in fact no coincidence. In the linear process, representing process *Merge*, the variables n and m register the last read values of streams 1 and 2, respectively. The variables in_1 and in_2, appearing in the modal formula have a similar purpose. This redundancy is identified by the invariant $(n = in_1) \wedge (m = in_2)$. Furthermore, the variable *out* satisfies the invariant $out \leqslant \mathsf{min}(in_1, in_2)$. It is straightforward to verify that both properties are invariants in the sense of Definition 38. Thus, rather than immediately solving this equation, it pays to solve the equation with the invariant.

$$\begin{aligned}
&\nu X_I(\sigma, n, m, in_1, in_2, out{:}\mathbb{N})\\
&\quad = (n{=}in_1 \wedge m{=}in_2 \wedge out \leqslant \mathsf{min}(in_1, in_2)) \wedge\\
&\qquad (\sigma = 0 \rightarrow (\forall l{:}\mathbb{N}. l \geqslant in_1 \rightarrow X_I(1, l, m, l, in_2, out))) \wedge\\
&\qquad (\sigma = 0 \rightarrow (\forall l{:}\mathbb{N}. l \geqslant in_2 \rightarrow X_I(2, n, l, in_1, l, out))) \wedge\\
&\qquad (\sigma = 1 \rightarrow (\forall l{:}\mathbb{N}. l \geqslant in_2 \rightarrow X_I(3, n, l, in_1, l, out))) \wedge\\
&\qquad (\sigma = 2 \rightarrow (\forall l{:}\mathbb{N}. l \geqslant in_1 \rightarrow X_I(3, l, m, l, in_2, out))) \wedge\\
&\qquad (\sigma = 3 \wedge n \leqslant m) \rightarrow (n \geqslant out \wedge X_I(2, n, m, in_1, in_2, n)) \wedge\\
&\qquad (\sigma = 3 \wedge m \leqslant n) \rightarrow (m \geqslant out \wedge X_I(1, n, m, in_1, in_2, m)).
\end{aligned}$$

It is straightforward to approximate this equation.

$$\begin{aligned}
&X_0(\sigma, n, m, in_1, in_2, out) = \top,\\
&X_1(\sigma, n, m, in_1, in_2, out) = n{=}in_1 \wedge m{=}in_2 \wedge out \leqslant \mathsf{min}(in_1, in_2).
\end{aligned}$$

The approximation X_1 is stable and hence it is the solution for X_I.

Now we cannot use this solution to construct a solution for $X(\sigma, n, m, 0, 0, 0)$, simply because it does not satisfy the invariant. However, if we consider $X(\sigma, 0, 0, 0, 0, 0)$, then using Theorem 42 we can use the solution for X_I as the solution for X. More concretely, $X(\sigma, 0, 0, 0, 0, 0)$ is always true.

Approximating the fixpoint equation for X directly does not terminate as quickly and is awkward due to universal quantifier that remains present in the approximations.

5.3. An identity tag generator

Many applications depend on a mechanism that produces identity tags for objects. Illustrative examples of such tags are the identity numbers on passports, phone-numbers,

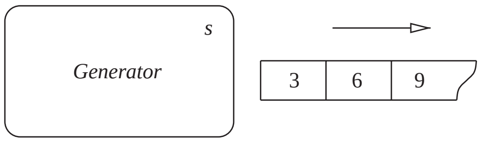

Fig. 3. Identity tag generator.

but also IP-addresses and message-header tags in e-mails. In essence, the mechanism for producing identity tags is a process that writes an infinite stream of identities. We represent these identities by means of natural numbers, see Fig. 3.

The process *Generator* is a generic process that generates identity tags according to some predefined function that is passed as a parameter to process *Generator*. The generator is initialised with the value *i*:

$$\textbf{proc } \mathit{Generator}(f{:}\mathbb{N}\to\mathbb{N}, i{:}\mathbb{N}) = s(i)\cdot \mathit{Generator}(f, f(i)).$$

Thus, by executing process $\mathit{Generator}(\mathit{succ}, 0)$, where *succ* is the successor function for natural numbers, we can generate the natural numbers. Most applications, using the generator, rely on the generator to produce unique tags. Thus, any two outputs of the system should be different. This is expressed by the following modal formula. It says that always in the future whenever a tag *m* is generated, every tag *n* generated later is not equal to *m*.

$$\nu\tilde{X}.([\top]\tilde{X} \wedge \forall m{:}\mathbb{N}.[s(m)]\nu\tilde{Y}.([\top]\tilde{Y} \wedge \forall n{:}\mathbb{N}.[s(n)]m \neq n)).$$

An alternative but more complex approach would be to store all outputs in a set and check that each tag being generated does not occur in the set. The fact that this is not needed in the above modal formula is due to the greatest fixpoint operators, which reasons about infinite runs of a system. Verifying this modal formula on process *Generator* allows us to find the conditions on the generator function that ensures all produced tags are unique. In order to do so, we need to solve the following equation system:

$$\begin{aligned}
\nu X(f{:}\mathbb{N}\to\mathbb{N}, i{:}\mathbb{N}) &= X(f, f(i)) \wedge \forall m{:}\mathbb{N}.(m = i) \to Y(f, f(i), m),\\
\nu Y(f{:}\mathbb{N}\to\mathbb{N}, i, m{:}\mathbb{N}) &= Y(f, f(i), m) \wedge \forall n{:}\mathbb{N}.(n = i) \to m \neq n.
\end{aligned}$$

Obviously, all universal quantifiers can be removed in the equations above. Thus, we can rewrite this equation system to the following equivalent equation system:

$$\begin{aligned}
\nu X(f{:}\mathbb{N}\to\mathbb{N}, i{:}\mathbb{N}) &= X(f, f(i)) \wedge Y(f, f(i), i),\\
\nu Y(f{:}\mathbb{N}\to\mathbb{N}, i, m{:}\mathbb{N}) &= Y(f, f(i), m) \wedge m \neq i.
\end{aligned}$$

These equations are both of the form of the pattern of Theorem 36. Hence, the solution to *Y* is $\forall j{:}\mathbb{N}. f^{j}(i) \neq m$. The solution to *X* is $\forall j'{:}\mathbb{N}.\forall j{:}\mathbb{N}. f^{j+j'+1}(i) \neq f^{j'}(i)$, which is logically equivalent to $\forall j{:}\mathbb{N}.\forall j'{:}\mathbb{N}. j \neq j' \to f^{j}(i) \neq f^{j'}(i)$. Of course, this is exactly the requirement we expected, but it is nice to see that we can also systematically derive it.

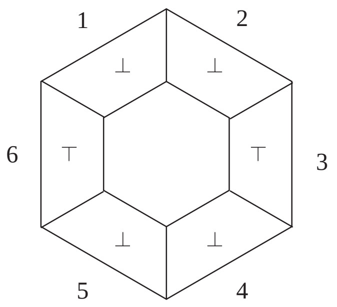

Fig. 4. Token ring system for $N = 6$ with two tokens.

5.4. A token ring

Synchronisation and mutual exclusion between processes in a network can be achieved by passing tokens. By abstracting from the behaviours of these processes, we can study the mechanisms to pass tokens in isolation. Networks using tokens usually have a ring topology and are called *token ring networks*. In Fig. 4, we depict a token ring configuration for two tokens and six processes.

We represent an arbitrary configuration in a token ring of size N by means of subsets of the set $\mathcal{N} = \{0, \ldots, N-1\}$. If there is at least one token at process j, the value j is in this subset. We define the operator $\upharpoonleft_j : 2^{\mathcal{N}} \to 2^{\mathcal{N}}$ as $\upharpoonleft_j(R) \overset{\text{def}}{=} (R \setminus \{j\}) \cup \{(j+1) \,\mathsf{mod}\, N\}$ indicating that tokens move from process j to process $(j+1) \,\mathsf{mod}\, N$. Process *Ring* describes a very simple token passing mechanism in μCRL:

$$\textbf{proc}\ Ring(R{:}2^{\mathcal{N}}) = \sum_{j:\mathbb{N}} token(j) \cdot Ring(\upharpoonleft_j(R)) \lhd j{\in}R \wedge j{<}N \rhd \delta.$$

Basically, process *Ring* executes a $token(j)$ action, for some j, whenever process j passes its token to the next process in the ring. The condition $\lhd j{\in}R \wedge j{<}N \rhd \delta$ says that this can only occur if j is an element of R and j is smaller than N, or in other words if process j has at least one token. One of the characteristics of this token passing mechanism is that it can *delete* tokens. To see that, take the configuration of Fig. 4 where $\top$ indicates the presence of a token. Consider the following sequence of actions: $token(3)\ token(4)\ token(5)$. At this point, there is only one token left in the token ring.

Given the simplicity of this system, it is not hard to see that there will always remain at least one token in the system. In fact, for process $Ring(R)$, the invariant $I(R) \equiv R \neq \emptyset$ can be proven fairly straightforwardly. However, we cannot immediately draw the conclusion that this process is fair in the sense that every process will always eventually hand over a token. This property is formally expressed by the following (first-order) modal μ-calculus formula.

$$\forall k{:}\mathbb{N}.k{<}N \to (\nu\tilde{X}.[\top]\tilde{X} \wedge \mu\tilde{Y}.(\langle token(k)\rangle\top \vee ([\top]\tilde{Y} \wedge \langle\top\rangle\top))).$$

Combining the modal formula with the process expression using the translation given in [11,15,26], we obtain the following equation system for arbitrary $k \in \mathcal{N}$:

$$\begin{aligned}
&\nu X(R{:}2^{\mathcal{N}}, k{:}\mathbb{N}) = \forall j{:}\mathbb{N}.(j \in R \wedge j < N \rightarrow X(\upharpoonleft_j(R), k)) \wedge Y(R, k),\\
&\mu Y(R{:}2^{\mathcal{N}}, k{:}\mathbb{N}) = (\exists j{:}\mathbb{N}. j \in R \wedge j < N \wedge j = k) \vee ((\exists j{:}\mathbb{N}. j \in R \wedge j < N) \wedge \\
&\qquad\qquad (\forall j{:}\mathbb{N}. j \in R \wedge j < N \rightarrow Y(\upharpoonleft_j(R), k))).
\end{aligned} \tag{6}$$

Note that after solving this equation system, the expression $\forall k{:}\mathbb{N}. k < N \rightarrow X(R, k)$ answers whether the token ring is fair. The equation for Y can be rewritten to

$$\begin{aligned}
\mu Y(R{:}2^{\mathcal{N}}, k{:}\mathbb{N}) = {} & ((k \in R \wedge k < N) \vee (\exists j{:}\mathbb{N} j \in R \wedge j < N)) \wedge \\
& \textstyle\bigwedge_{j<N}((k \notin R \vee k \geqslant N) \wedge j \in R \rightarrow Y(\upharpoonleft_j(R), k)).
\end{aligned}$$

Using Theorem 37 this equation can be solved, yielding

$$\begin{aligned}
&\mu Y(R{:}2^{\mathcal{N}}, k{:}\mathbb{N}) \\
&\quad = \exists j{:}\mathbb{N}. \exists g{:}\mathbb{N} \rightarrow \{0, \ldots, N-1\}.((\forall i{:}\mathbb{N}. i < j \rightarrow (k \notin \upharpoonleft(g, i, R) \vee k \geqslant N) \wedge \\
&\qquad g(i) \in \upharpoonleft(g, i, R)) \wedge ((k \in \upharpoonleft(g, j, R) \wedge k < N) \vee (\exists j'{:}\mathbb{N}. j' \in \upharpoonleft(g, j, R) \wedge j' < N))).
\end{aligned}$$

The right-hand side of this equation can be simplified using the rules of predicate calculus. We get (using Corollary 29):

$$\mu Y(R{:}2^{\mathcal{N}}, k{:}\mathbb{N}) = \exists j{:}\mathbb{N}.(j \in R \wedge j < N).$$

The solution for Y can now be substituted in the first equation obtaining:

$$\nu X(R{:}2^{\mathcal{N}}, k{:}\mathbb{N}.){=}\forall j{:}\mathbb{N}(j \in R \wedge j < N \rightarrow X(\upharpoonleft_j(R), k)) \wedge \exists j'{:}\mathbb{N}.(j' \in R \wedge j' < N).$$

We solve this equation by iteration

$$\begin{aligned}
X_0(R, k) &= \top,\\
X_1(R, k) &= \exists j'{:}\mathbb{N}.(j' \in R \wedge j' < N),\\
X_2(R, k) &= \forall j{:}\mathbb{N}.(j \in R \wedge j < N \rightarrow \exists j'{:}\mathbb{N}.(j' \in \upharpoonleft_j(R) \wedge j' < N)) \wedge \\
&\qquad\qquad \exists j''{:}\mathbb{N}.(j'' \in R \wedge j'' < N)\\
&= \exists j{:}\mathbb{N}.(j \in R \wedge j < N).
\end{aligned}$$

Hence, the solution of the system is

$$\begin{aligned}
\nu X(R{:}2^{\mathcal{N}}, k{:}\mathbb{N}) &= \exists j{:}\mathbb{N}.(j \in R \wedge j < N),\\
\mu Y(R{:}2^{\mathcal{N}}, k{:}\mathbb{N}) &= \exists j{:}\mathbb{N}.(j \in R \wedge j < N),
\end{aligned}$$

And so, the token ring is fair if $\forall k{:}\mathbb{N}. k < N \rightarrow \exists j{:}\mathbb{N}.(j \in R \wedge j < N)$. This can be slightly simplified to $N{=}0 \vee \exists j{:}\mathbb{N}.(j \in R \wedge j < N)$.

5.5. A lossy channel

Consider a simple lossy channel that reads information from a stream, and tries to send it to the other side where a message is lost occasionally:

$$\begin{aligned}
&C_\top = \textstyle\sum_{m:\mathbb{N}} r(m) \cdot C_\bot(m),\\
&C_\bot(m{:}\mathbb{N}) = s(m) \cdot C_\top + l \cdot C_\top.
\end{aligned}$$

We wish to verify that when data is not always lost, messages eventually get across. We formulate this using the following modal formula:

$$\nu \tilde{X}.([\top]\tilde{X} \wedge (\mu \tilde{Y}.[\top]\tilde{Y} \vee \langle l \rangle \top \vee \exists m{:}\mathbb{N}.\langle s(m)\rangle \top)).$$

We first translate the process to linear form:

$$\begin{aligned} C(b{:}\mathbb{B}, m{:}\mathbb{N}) = {} & \textstyle\sum_{k:\mathbb{N}} r(k) \cdot C(\bot, k) \lhd b \rhd \delta \\ & + s(m) \cdot C(\top, m) \lhd \neg b \rhd \delta \\ & + l \cdot C(\top, m) \lhd \neg b \rhd \delta \end{aligned}$$

The process $C_\top$ is equal to $C(\top, m)$ for any $m{:}\mathbb{N}$ and $C_\bot(m)$ is equal to $C(\bot, m)$.

The equation system we obtain is the following:

$$\begin{aligned} \nu X(b{:}\mathbb{B}, m{:}\mathbb{N}) &= (\forall k{:}\mathbb{N}.(b \to X(\bot, k)) \wedge (\neg b \to X(\top, m))) \wedge Y(b, m), \\ \mu Y(b{:}\mathbb{B}, m{:}\mathbb{N}) &= (\forall k{:}\mathbb{N}.(b \to Y(\bot, k)) \wedge (\neg b \to Y(\top, m))) \vee \\ & \qquad \neg b \vee \exists m'{:}\mathbb{N}.\neg b \wedge m{=}m'. \end{aligned}$$

Approximation quickly leads to a solution not involving m:

$$\begin{aligned} & Y_0(b, m) = \bot, \\ & Y_1(b, m) = \neg b \wedge (b \vee \neg b) = \neg b, \\ & Y_2(b, m) = (\neg b \to \neg b) \vee \neg b = \top, \\ & X_0(b, m) = \top, \end{aligned}$$

where $X_0(b, m) = \top$ is a stable solution. Thus, in whatever state the process C starts, messages always get across if not always lost.

A slightly more involved property, taken from [6, p. 309], says that delivery via action $s(m)$ is fairly treated if there are no paths where $s(m)$ is enabled infinitely often, but occurs only finitely often:

$$\begin{aligned} & \nu \tilde{X}.\mu \tilde{Y}.\nu \tilde{Z}.\forall m{:}\mathbb{N}.[s(m)]\tilde{X} \wedge \\ & \quad (\exists m{:}\mathbb{N}.\langle s(m)\rangle \top \to ([l]\tilde{Y} \wedge \forall m{:}\mathbb{N}.[r(m)]\tilde{Y})) \wedge [l]\tilde{Z} \wedge \forall m{:}\mathbb{N}.[r(m)]\tilde{Z}. \end{aligned}$$

This formula together with process C are translated to the following equation system:

$$\begin{aligned} \nu X(b{:}\mathbb{B}, m{:}\mathbb{N}) &= Y(b, m), \\ \mu Y(b{:}\mathbb{B}, m{:}\mathbb{N}) &= Z(b, m), \\ \nu Z(b{:}\mathbb{B}, m{:}\mathbb{N}) &= (\neg b \to X(\top, m)) \wedge (\neg b \to ((\neg b \to Y(\top, m)) \wedge \\ & \qquad \forall k{:}\mathbb{N}.(b \to Y(\bot, k)))) \wedge \\ & \qquad ((\neg b \to Z(\top, m)) \wedge \forall k{:}\mathbb{N}.(b \to Z(\bot, k))) \\ &= (\neg b {\to} X(\top, m) {\wedge} Y(\top, m) {\wedge} Z(\top, m)) {\wedge} (b {\to} \forall k{:}\mathbb{N}.Z(\bot, k)). \end{aligned}$$

We approximate Z and find a stable solution in three steps:

$$\begin{aligned} Z_0(b{:}\mathbb{B}, m{:}\mathbb{N}) &= \top, \\ Z_1(b{:}\mathbb{B}, m{:}\mathbb{N}) &= \neg b \to X(\top, m) \wedge Y(\top, m), \\ Z_2(b{:}\mathbb{B}, m{:}\mathbb{N}) &= (\neg b \to X(\top, m) \wedge Y(\top, m)) \wedge (\forall k{:}\mathbb{N}.X(\top, k) \wedge Y(\top, k)) \\ &= \forall k{:}\mathbb{N}.X(\top, k) \wedge Y(\top, k). \end{aligned}$$

We substitute the solution for Z in the second equation obtaining the following fixpoint equation:

$$\mu Y(b{:}\mathbb{B}, m{:}\mathbb{N}) = \forall k{:}\mathbb{N}. X(\top, k) \wedge Y(\top, k).$$

Using one approximation step it is easily seen that the solution of this equation is $Y(b, m) = \perp$. So, substitution of this solution in the first equation yields $X(b, m) = \perp$. The property does not hold for our process.

5.6. A client–server model

Here we verify a property of a simplified client server system. A client can place a number of orders using action o_c, and pay for these later, using action p_c. A server keeps track of the outstanding accounts of the client; as long as the outstanding accounts are below a certain threshold T, the server accepts all orders that fall within the budget of the client, using action o_s. The server receives payment of the outstanding accounts via action p_s. Whenever the outstanding account of the client is above threshold T, the server issues a warning via action w_s. The communications via the client and the server proceed as follows: actions p_c and p_s communicate to action p, whereas actions o_c and o_s communicate to action o. The total system is given below in μCRL:

$$\begin{aligned}
&\mathit{ClientServer}(n_c, n_s{:}\mathbb{N}) = \partial_{\{o_c, o_s, p_c, p_s\}}(\mathit{Client}(n_c) \| \mathit{Server}(n_s))\\
&\mathit{Client}(n{:}\mathbb{N}) = \textstyle\sum_{m:\mathbb{N}} o_c(m) \cdot \mathit{Client}(n{+}m)\\
&\qquad\qquad\quad + \textstyle\sum_{m:\mathbb{N}} p_c(m) \cdot \mathit{Client}(n{-}m) \lhd n \geqslant m \rhd \delta\\
&\mathit{Server}(n{:}\mathbb{N}) = \textstyle\sum_{m:\mathbb{N}} o_s(m) \cdot \mathit{Server}(n{+}m) \lhd n{+}m \leqslant T \rhd \delta\\
&\qquad\qquad\quad + \textstyle\sum_{m:\mathbb{N}} p_s(m) \cdot \mathit{Server}(n{-}m)\\
&\qquad\qquad\quad + w_s \cdot \mathit{Server}(n) \lhd n{>}T \rhd \delta.
\end{aligned}$$

A desirable property of client–server system is that it prevents the clients from placing too many orders and having a too large debt. The client–server system we specified issues a warning on these occasions. In order to check whether the client–server system behaves decently, we must show that no warnings are issued. Thus, the property we are interested in is

$$\nu \tilde{X}.([\top]\tilde{X} \wedge [w_s]\perp).$$

The verification of this property proceeds as follows. We rewrite the client–server process to linear form in effect removing all parallelism from the specification. The resulting linear process is combined with the modal formula, yielding the following equation:

$$\begin{aligned}
\nu X(n_c, n_s{:}\mathbb{N}) = {} & (\forall m{:}\mathbb{N}.(n_s{+}m \leqslant T \to X(n_c{+}m, n_s{+}m)) \wedge\\
& (n_c \geqslant m \to X(n_c{-}m, n_s{-}m))) \wedge\\
& (n_s > T \to X(n_c, n_s)) \wedge n_s \leqslant T.
\end{aligned}$$

Using approximation, the solution of this equation is obtained by two iterations:

$$\begin{aligned}
&X_0(n_c, n_s) = \top,\\
&X_1(n_c, n_s) = n_s \leqslant T.
\end{aligned}$$

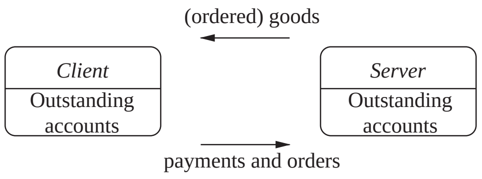

Fig. 5. Placing orders and transferring money.

The solution X_1 is stable. Thus, as long as initially, the outstanding account at the server is less than *T*, this client server model works as desired (Fig. 5).

6. Conclusions

We set out to develop a theory that allows to manually solve parameterised boolean equation systems. Our main motivation came from work reported in [15] where a symbolic model checker is described that works by fixpoint approximation and automated reasoning (actually an equality BDD package that also allows rewriting [12]). This was successful in the sense that automatically properties of large and infinite state systems could be proven. But we found that automated reasoning and finite approximations were often insufficient. We believe that ultimately an interplay between manual and automated techniques will turn out to be most effective, and therefore started this investigation.

Regarding the general theory in this paper we have some mixed feelings. Most theorems and corollaries have a nice and usable shape and these work very smoothly in the applications. But most proofs had to be given using Definition 3 which is very hard to comprehend. We would appreciate a much more insightful basic theory but do not know how to provide it. Such a theory could also help us to avoid the pitfalls of fixpoint equations. More than once we went awry formulating and believing conjectures that turned out to be utterly untrue.

Regarding the use of the theory the patterns, approximations and invariants are real marbles. It remains to be seen how the theory evolves under the strain of more involved verifications and most likely requires adaptation and strengthening. One of the most eye-catching questions is whether the patterns in Section 4.2.3 can be generalised to arbitrary right-hand sides, providing a universal way of solving parameterised equation systems or whether a whole plethora of techniques for many different forms will be developed.

Related work: The first accounts of using fixpoints for reasoning about programs date back to 1969, when Scott and de Bakker [23] defined the μ-calculus. The μ-calculus has a μ-operator that acts as a binder for relation variables, and is used to express recursion and iteration. Like parameterised boolean equation systems, the μ-calculus is a first-order formalism. Several theoretical results have been obtained for the μ-calculus (see e.g. [19]), but gradually, the propositional version became more popular.

With respect to the model checking problem for processes with data, several other approaches are noteworthy. Bradfield and Stirling [5,6,24] lay the foundations for finite and infinite state model checking based on the modal μ-calculus using *tableau systems*. Furthermore, the ideas of using Petri nets in combination with model checking are described.

As explained in [21], the techniques using tableaus and boolean equation systems are closely related, but boolean equation systems require less overhead.

In a similar vein, Gurov et al. [18], and Rathke and Hennessy [22] define (independently from each other) first-order extensions of the modal μ-calculus and use *symbolic transition systems* as the underlying models. Both Gurov et al. [18] and Rathke and Hennessy [22], provide tableau systems and proof systems, and in [22] completeness and soundness is shown. The main concern in [18] is that of compositionality. To the best of our knowledge, neither techniques have led to the development of tool support. From a theoretical point of view, it would be interesting to compare the expressive power of the logics of [18,22,11], as there appear to be some differences. For instance, the grammar in [22] prohibits the use of a diamond modality in combination of a fixpoint operator. Thus, the expression $\mu X \langle \top \rangle X$ (where $\top$ is the set of *all possible actions*) appears to be excluded by the grammar of the logic, whereas it is a valid expression in the logic of [11].

In contrast to these general approaches there is work that considers subclasses of systems or logical properties. The main focus in these approaches is mainly on decidability. Noteworthy approaches are CLU by Bryant et al. [7], the use of regular expression [1] and queue representations [4] for communication protocols and Pressburger arithmetic [8] for process networks.

Acknowledgements

We thank Marc Voorhoeve for the counterexample following Lemma 30, Joost-Pieter Katoen for suggesting the identity tag generator example and Kees van Hee for indicating that client–server systems are important systems for which properties should be provable.

References

[1] P. Abdulla, A. Bouajjani, B. Jonsson, On-the-fly analysis of systems with unbounded, lossy fifo channels, in: A.J. Hu, M.Y. Vardi (Eds.), 10th Internat. Conf. on Computer Aided Verification, CAV'98, Lecture Notes in Computer Science, Vol. 1427, Springer, 1998, pp. 305–318.

[2] H. Bekič, Definable operations in general algebras, and the theory of automata and flow charts, in: C.B. Jones (Ed.), Programming Languages and Their Definition – Hans Bekič (1936–1982), Lecture Notes in Computer Science, Vol. 177, Springer, 1984, pp. 30–55.

[3] M.A. Bezem, J.F. Groote, Invariants in process algebra with data, in: B. Jonsson, J. Parrow (Eds.), Proc. Concur'94, Uppsala, Sweden, Lecture Notes in Computer Science, Vol. 836, Springer, 1994, pp. 401–416.

[4] B. Boigelot, P. Godefroid, B. Willems, P. Wolper, The power of qdds, in: P. van Hentenryck (Ed.), Static Analysis, 4th Internat. Symp. SAS'97, Lecture Notes in Computer Science, Vol. 1302, Springer, 1997, pp. 172–186.

[5] J.C. Bradfield, Verifying Temporal Properties of Systems, Progress in Theoretical Computer Science, Birkhäuser, 1992.

[6] J. Bradfield, C. Stirling, Modal logics and mu-calculi: an introduction, in: J.A. Bergstra, A. Ponse, S.A. Smolka (Eds.), Handbook of Process Algebra, Elsevier, 2001, pp. 293–330.

[7] R.E. Bryant, S.K. Lahiri, S.A. Seshia, Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions, in: 14th Internat. Conf. on Computer Aided Verification, CAV 2002, Lecture Notes in Computer Science, Vol. 2404, Springer, 2002, pp. 78–92.

[8] T. Bultan, R. Gerber, W. Pugh, Symbolic model checking of infinite state systems using pressburger arithmetic, in: O. Grumberg (Ed.), Ninth Internat. Conf. on Computer Aided Verification, CAV'97, Lecture Notes in Computer Science, Vol. 1254, Springer, 1997, pp. 400–411.

[9] P. Cousot, Semantic foundations of program analysis, in: S.S. Muchnick, N.D. Jones (Eds.), Program Flow Analysis: Theory and Applications, Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1981, pp. 303–342 (Chapter 10).

[10] E.A. Emerson, C.-L. Lei, Efficient model checking in fragments of the propositional mu-calculus, in: First IEEE Symp. Logic in Computer Science, LICS'86, IEEE Computer Society Press, 1986, pp. 267–278.

[11] J.F. Groote, R. Mateescu, Verification of temporal properties of processes in a setting with data, in: A.M. Haeberer (Ed.), AMAST'98, Lecture Notes in Computer Science, Vol. 1548, Springer, 1999, pp. 74–90.

[12] J.F. Groote, J.C. van de Pol, Equational Binary Decision Diagrams, in: Proc. LPAR 2000, Reunion Island, LNAI 1955, 2000, pp. 161–178.

[13] J.F. Groote, A. Ponse, The syntax and semantics of μCRL, in: A. Ponse, C. Verhoef, S.F.M. van Vlijmen (Eds.), Algebra of Communicating Processes '94, Workshops in Computing Series, Springer, 1995, pp. 26–62.

[14] J.F. Groote, M.A. Reniers, Algebraic process verification, in: J.A. Bergstra, A. Ponse, S.A. Smolka (Eds.), Handbook of Process Algebra, Elsevier, North-Holland, 2001, pp. 1151–1208, (Chapter 17).

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

[16] J.F. Groote, T.A.C. Willemse, Parameterised boolean equation systems (Extended Abstract), in: P. Gardner, N. Yoshida (Eds.), Proc. of CONCUR 2004, London, Lecture Notes in Computer Science, Vol. 3170, Springer, 2004, pp. 308–324.

[17] J.F. Groote, T.A.C. Willemse, Parameterised boolean equation systems, Technical Report CSR 04-09, Eindhoven University of Technology, Department of Mathematics and Computer Science, 2004.

[18] D. Gurov, S. Berezin, B. Kapron, A modal μ-calculus and a proof system for value-passing processes, in: Proc. Infinity, Workshop on Verification of Infinite State Systems, Pisa, 1996, pp. 149–163.

[19] D. Kozen, Results on the propositional mu-calculus, Theoret. Comput. Sci. 27 (1983) 333–354.

[20] A. Mader, Modal μ-calculus, model checking and gauß elimination, in: E. Brinksma, R.W. Cleaveland, K.G. Larsen, T. Margaria, B. Steffen (Eds.), Tools and Algorithms for Construction and Analysis of Systems, First International Workshop, TACAS '95, Aarhus, Denmark, Lecture Notes in Computer Science, Vol. 1019, Springer, 1995, pp. 72–88.

[21] A. Mader, Verification of Modal Properties Using Boolean Equation Systems, PhD Thesis, Technical University of Munich, 1997.

[22] J. Rathke, M. Hennessy, Local model checking for value-passing processes, in: Proc. of TACS'97, the Internat. Symp. on Theoretical Aspects of Computer Software, Sendai, 1997, 1997.

[23] D.S. Scott, J.W. de Bakker, A theory of programs, 1969.

[24] C. Stirling, Modal and Temporal Properties of Processes, Texts in Computer Science, Springer, 2001.

[25] B. Vergauwen, J. Lewi, Efficient local correctness checking for single and alternating boolean equation systems, in: S. Abiteboul, E. Shamir (Eds.), Proc. ICALP'94, Lecture Notes in Computer Science, Vol. 820, Springer, 1994, pp. 302–315.

[26] T.A.C. Willemse, Semantics and Verification in Process Algebras with Data and Timing, PhD Thesis, Eindhoven University of Technology, February 2003.