Local Model-Checking of an Alternation-Free Value-Based Modal Mu-Calculus

Radu Mateescu[1]
CWI / SEN2 group
P.O. Box 94079, NL-1090 GB Amsterdam, The Netherlands
Radu.Mateescu@cwi.nl

Abstract

Programs written in value-passing description languages such as μCRL and LOTOS can be naturally translated into Labelled Transition Systems (LTSs) containing data values. In order to express temporal properties interpreted over these LTSs, we define a value-based alternation-free modal μ-calculus built from typed variables, pattern-matching modalities, and parameterised fixed point operators. The verification of a temporal formula over a (finite) LTS is reduced to the (partial) resolution of a Parameterised Boolean Equation System (PBES). We propose a resolution method for PBESs that leads to a local model-checking algorithm, which could also be applied in the framework of abstract interpretation.

1 Introduction

Formal verification is essential in order to ensure reliability of critical applications such as communication protocols and distributed systems. A state-of-the-art verification technique is the so-called *model-checking*. In this approach, the application is first described using a high-level language like μCRL[2] [7] or LOTOS[3] [9]. This description is subsequently translated into a (finite) Labelled Transition System (LTS), over which the desired correctness properties, expressed as temporal logic formulas, are verified using appropriate model-checking algorithms.

The modal μ-calculus [10] is a powerful formalism allowing to express temporal properties of programs. However, in its standard version, it does not allow to handle data values, therefore being not appropriate for value-passing description languages as μCRL or LOTOS, whose underlying LTS models contain typed values. Although several value-based extensions of the μ-calculus have been proposed and studied in the setting of theorem-proving [12, 5], much less attention has been given to the corresponding model-checking problem for finite-state systems.

In this paper, we propose an extension of the μ-calculus with data variables, pattern-matching modalities, and parameterised fixed point operators, allowing to express temporal properties involving data. We focus on the alternation-free fragment of the logic (i.e., with no mutual recursion between least and greatest fixed points), which offers a practical compromise between the expressive power and the efficiency of model-checking. Generalizing the approaches used for the efficient model-checking of the standard alternation-free μ-calculus [2, 4, 1, 14], we reduce the model-checking problem of a value-based temporal formula over a (finite) LTS to the (partial) resolution of a Parameterised Boolean Equation System (PBES). We propose a resolution method for PBESs that leads to a semi-decidable local (i.e., allowing to construct the LTS "on-the-fly", during the verification of the formula) model-checking algorithm.

The paper is organized as follows. Section 2 defines the value-based alternation-free μ-calculus and gives various examples of temporal properties involving data. Section 3 shows how the verification problem can be reduced to the (partial) resolution of a PBES and describes the local model-checking algorithm. Finally, Section 4 contains some concluding remarks and directions for future work.

[1] Current affiliation is: INRIA Rhône-Alpes / VASY, 655, avenue de l'Europe, F-38330 Montbonnot Saint Martin, France. E-mail: Radu.Mateescu@inria.fr

[2] *micro* Common Representation Language

[3] Language Of Temporal Ordering Specification

2 Alternation-free value-based modal μ-calculus

We first define the LTS models used in the remainder of the paper, next we describe the syntax and semantics of the value-based μ-calculus that we propose, and we finish this section by showing various examples of data-based temporal properties expressed in our logic.

2.1 Labelled transition systems

The LTS models we consider here are suitable for value-passing description languages, whose actions may contain typed values exchanged during communications. An LTS is a tuple $\langle S, A, T, s_0 \rangle$, where:

- S is a finite set of *states*;
- A is a finite set of *actions*. An action $a \in A$ is represented as a list $c\ \vec{v}$, where c is a channel name and $\vec{v}$ is a list of typed values;
- $T \subseteq S \times A \times S$ is the *transition relation*. A transition $(s_1, a, s_2) \in T$, also noted $s_1 \xrightarrow{a} s_2$, indicates that the program can move from state s_1 to state s_2 by performing action a;
- $s_0 \in S$ is the *initial state*.

In the sequel, we implicitly consider an LTS model $M = \langle S, A, T, s_0 \rangle$, over which the extended μ-calculus formulas will be interpreted.

2.2 Syntax and semantics of the logic

The logic that we propose is built from three types of entities, whose syntax is given in Figure 1.

Expressions	$e ::= x \mid f(\vec{e})$
Action formulas	$\alpha ::= c\ \vec{x}{:}\vec{t} \mid c\ \vec{e} \mid \neg\alpha \mid \alpha_1 \wedge \alpha_2$
State formulas	$\varphi ::= tt \mid e \rightarrow \varphi_1 \,[]\, \varphi_2 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\alpha\rangle\,\varphi \mid Y(\vec{e}) \mid \mu Y(\vec{x}{:}\vec{t}{:=}\vec{e}).\varphi$

Figure 1: Syntax of the logic

Expressions: these are algebraic terms $e \in \mathit{Exp}$, constructed over *data variables* $x \in \mathit{DVar}$ and *functions* $f \in \mathit{Func}$. We assume that each variable x has a unique type t, each function f has a unique profile $\vec{t}_1 \rightarrow t_2$ (where $\vec{t}_1$ is the list of argument types, and t_2 is the result type of f), and each expression e is well-typed.

Action formulas: these are logical formulas $\alpha \in \mathit{AForm}$, built from action predicates and the usual boolean connectives. A predicate $c\ \vec{x}{:}\vec{t}$ states that an action must have the form $c\ \vec{v}$, where $\vec{v} \in \vec{t}$; if this is the case, the values $\vec{v}$ are respectively assigned to the variables $\vec{x}$, which are *exported*, i.e., can be used outside this action formula. A predicate $c\ \vec{e}$ states that an action must have the form $c\ \vec{v}$, where $\vec{v}$ are the values of the expressions $\vec{e}$. The derived boolean operators are defined as usual: $tt = c \vee \neg c$ for some channel name c, $ff = \neg tt$, $\alpha_1 \vee \alpha_2 = \neg(\neg\alpha_1 \wedge \neg\alpha_2)$, $\alpha_1 \rightarrow \alpha_2 = \neg\alpha_1 \vee \alpha_2$, and $\alpha_1 \leftrightarrow \alpha_2 = (\alpha_1 \rightarrow \alpha_2) \wedge (\alpha_2 \rightarrow \alpha_1)$.

State formulas: these are temporal formulas $\varphi \in \mathit{SForm}$, built from boolean, modal, and fixed point operators. The $e \rightarrow \varphi_1 \,[]\, \varphi_2$ operator is a conditional construct, meaning that a state satisfies φ_1 if the boolean expression e is true and φ_2 otherwise. The diamond modality $\langle\alpha\rangle\,\varphi$ expresses that a state has an outgoing transition whose action satisfies α and which leads to a state satisfying φ. The $\mu Y(\vec{x}{:}\vec{t}{:=}\vec{e}).\varphi$ construct, where $Y \in \mathit{PVar}$ is a *propositional variable* parameterised by the data variables $\vec{x}$, is the least fixed point operator, denoting the least solution of the fixed point equation $Y(\vec{x}{:}\vec{t}) = \varphi$, evaluated with the arguments $\vec{e}$. Besides the usual derived boolean connectives, the following operators are also defined: the state predicate $e = e \rightarrow tt \,[]\, ff$, the box modality $[\alpha]\,\varphi = \neg\langle\alpha\rangle\neg\varphi$, and the greatest fixed point operator $\nu Y(\vec{x}{:}\vec{t}{:=}\vec{e}).\varphi = \neg\mu Y(\vec{x}{:}\vec{t}{:=}\vec{e}).\neg\varphi[\neg Y/Y]$, where $\varphi[\neg Y/Y]$ denotes the syntactic substitution of Y by $\neg Y$ in φ. The state formulas φ are assumed to be *alternation-free* (i.e., without mutually recursive least and greatest fixed point subformulas) and *syntactically monotonic* (i.e., with propositional variables occurring inside the fixed point formulas only under an even number of negations).

The semantics of these constructs is given in Figure 2. Expressions are interpreted w.r.t. *data environments* $\varepsilon \in \mathbf{DEnv}$, which are partial functions mapping data variables to values (elements of the domain $\mathbf{Val}$). An environment $\varepsilon = [\vec{v}/\vec{x}]$ assigns the values $\vec{v}$ to the variables $\vec{x}$, which are also called the *support* of ε, noted $supp(\varepsilon)$. The *overriding* of ε_1 by ε_2 is defined as $(\varepsilon_1 \oslash \varepsilon_2)(x) = \mathit{if}\ x \in supp(\varepsilon_2)\ \mathit{then}\ \varepsilon_2(x)\ \mathit{else}\ \varepsilon_1(x)$. The denotation $[\![e]\!]\,\varepsilon$, where $[\![.]\!] : \mathit{Exp} \to \mathbf{DEnv} \to \mathbf{Val}$ is the interpretation function for expressions, gives the value of e in the context of ε (which must assign values to all the data variables occurring in e).

Expressions	$[\![x]\!]\,\varepsilon$	$=$	$\varepsilon(x)$
	$[\![f(\vec{e})]\!]\,\varepsilon$	$=$	$f([\![\vec{e}]\!]\,\varepsilon)$
Action formulas	$[\![c\ \vec{x}{:}\vec{t}]\!]\,\varepsilon a$	$=$	$\mathit{if}\ \exists \vec{v}{:}\vec{t}.a = c\ \vec{v}\ \mathit{then}\ (\mathbf{tt}, [\vec{v}/\vec{x}])\ \mathit{else}\ (\mathbf{ff}, [\,])$
	$[\![c\ \vec{e}]\!]\,\varepsilon a$	$=$	$\mathit{if}\ a = c\ [\![\vec{e}]\!]\,\varepsilon\ \mathit{then}\ (\mathbf{tt}, [\,])\ \mathit{else}\ (\mathbf{ff}, [\,])$
	$[\![\neg\alpha]\!]\,\varepsilon a$	$=$	$(\mathit{not}\ ([\![\alpha]\!]\,\varepsilon a)_1, [\,])$
	$[\![\alpha_1 \wedge \alpha_2]\!]\,\varepsilon a$	$=$	$(([\![\alpha_1]\!]\,\varepsilon a)_1\ \mathit{and}\ ([\![\alpha_2]\!]\,\varepsilon a)_1, [\,])$
State formulas	$[\![tt]\!]\,\rho\varepsilon$	$=$	S
	$[\![e \to \varphi_1\,[]\,\varphi_2]\!]\,\rho\varepsilon$	$=$	$\mathit{if}\ \ [\![e]\!]\,\varepsilon\ \mathit{then}\ \ [\![\varphi_1]\!]\,\rho\varepsilon\ \mathit{else}\ \ [\![\varphi_2]\!]\,\rho\varepsilon$
	$[\![\neg\varphi]\!]\,\rho\varepsilon$	$=$	$S \setminus [\![\varphi]\!]\,\rho\varepsilon$
	$[\![\varphi_1 \wedge \varphi_2]\!]\,\rho\varepsilon$	$=$	$[\![\varphi_1]\!]\,\rho\varepsilon \cap [\![\varphi_2]\!]\,\rho\varepsilon$
	$[\![\langle\alpha\rangle\,\varphi]\!]\,\rho\varepsilon$	$=$	$\{s \in S \mid \exists s' \in S, a \in A.s \xrightarrow{a} s' \wedge ([\![\alpha]\!]\,\varepsilon a)_1 \wedge s \in [\![\varphi]\!]\,\rho(\varepsilon \oslash ([\![\alpha]\!]\,\varepsilon a)_2)\}$
	$[\![Y(\vec{e})]\!]\,\rho\varepsilon$	$=$	$\big(\rho(Y)\big)([\![\vec{e}]\!]\,\varepsilon)$
	$[\![\mu Y(\vec{x}{:}\vec{t}{:=}\vec{e}).\varphi]\!]\,\rho\varepsilon$	$=$	$\big(\prod\{F : \vec{t} \to 2^S \mid \Phi_{\rho\varepsilon}(F) \sqsubseteq F\}\big)([\![\vec{e}]\!]\,\varepsilon)$
	where $\Phi_{\rho\varepsilon}(F) = \lambda\vec{v}{:}\vec{t}.\,[\![\varphi]\!]\,(\rho \oslash [F/Y])(\varepsilon \oslash [\vec{v}/\vec{x}])$		

Figure 2: Semantics of the logic

Action formulas are interpreted w.r.t. data environments and actions. The denotation $[\![\alpha]\!]\,\varepsilon a$, where $[\![.]\!] : \mathit{AForm} \to \mathbf{DEnv} \to A \to \mathbf{Bool} \times \mathbf{DEnv}$ is the interpretation function for action formulas, gives a tuple containing a boolean value that indicates if a satisfies α in the context of ε, and a data environment that assigns values extracted from a to the data variables exported by α. For simplicity, we allow data variables to be exported only by the $c\ \vec{x}{:}\vec{t}$ action predicates. Tuple values are noted using parenthesis and projection is denoted by subscripts: the i-th element of a tuple value $(v_1, ..., v_n)$ is denoted by $(v_1, ..., v_n)_i$.

State formulas are interpreted w.r.t. *propositional environments* $\rho \in \mathbf{PEnv}$ and data environments, the former being partial functions mapping variables $Y(\vec{x}{:}\vec{t})$ to functions in $\vec{t} \to 2^S$. The denotation $[\![\varphi]\!]\,\rho\varepsilon$, where $[\![.]\!] : \mathit{SForm} \to \mathbf{PEnv} \to \mathbf{DEnv} \to 2^S$ is the interpretation function for state formulas, gives the set of states satisfying φ in the context of ρ and ε. The domain containing the values of all types $\vec{t}$ of the fixed point formula parameters is noted $\mathbf{Par}$. The functionals $\Phi_{\rho\varepsilon}$ associated to the fixed point formulas being monotonic (due to the syntactic monotonicity of the φ formulas) and the lattice $\mathbf{Par} \to 2^S$ being complete, Tarski's theorem [13] ensures that the interpretation of fixed point formulas given in Figure 2 denotes the corresponding extremal fixed points of $\Phi_{\rho\varepsilon}$.

2.3 Examples

We give below several examples of typical temporal properties involving data, expressed as formulas of our alternation-free value-based modal μ-calculus defined in the previous subsection.

Correct message transmission. The following formula states that each emission of a message must be eventually followed by the reception of the same message:

$$[\mathit{SEND}\ m{:}\mathit{Msg}]\,\mu Y.(\langle tt\rangle\ tt \wedge [\neg(\mathit{RECV}\ m)]\,Y)$$

where $\mathit{SEND}\ m$ and $\mathit{RECV}\,m$ denote the emission and the reception of the message m, respectively.

Mutual exclusion. The fact that several processes (identified as values of an enumerated type *Pid*) access a shared resource in mutual exclusion can be expressed by the formula below:

$$[OPEN\ p_1{:}Pid]\,\neg\mu Y.(\langle OPEN\ p_2{:}Pid\rangle\,(p_2 \neq p_1) \vee \langle\neg(CLOSE\ p_1)\rangle\,Y)$$

where the actions *OPEN p* and *CLOSE p* model the fact that process p has obtained and released the resource, respectively.

Alternation between message emission and reception. The strict alternation between emissions and receptions of messages in a communication protocol can be expressed using a fixed point formula parameterised by a boolean flag indicating if a message has been sent or not before the current state:

$$\nu Y(sent{:}Bool{:=}ff).([SEND]\,(\neg sent \wedge Y(\neg sent)) \wedge [RECV]\,(sent \wedge Y(\neg sent)) \wedge [\neg(SEND \vee RECV)]\,Y(sent))$$

This property can also be expressed in the standard μ-calculus (since it does not refer to the values contained in the LTS actions), but less concisely than the formula above (i.e., requiring two nested greatest fixed points).

Correct transmission of a segmented data packet. The Bounded Retransmission Protocol [8] allows to transmit (large) data packets over an unreliable communication medium by splitting them in (small) chunks that are sent sequentially. The correct transmission of a packet can be expressed by the formula below, in which the parameter p stores the portion of the packet remaining to be sent:

$$\begin{array}{l}[IN\ p_0{:}Packet]\ \nu Y(p{:}Packet{:=}p_0).(\\ \quad empty(p) \rightarrow tt\ []\ ([IN\ p_1{:}Packet]\,ff \wedge [OUT\ c{:}Chunk]\,(c = head(p) \wedge Y(tail(p))) \wedge\\ \quad\quad [\neg((IN\ p_2{:}Packet) \vee (OUT\ p_2{:}Packet))]\,Y(p)))\end{array}$$

where *IN p* denotes the emission of a packet p, *OUT c* denotes the reception of a chunk c, and *empty*, *head*, and *tail* are functions testing the emptiness and returning the head and the remainder of a packet, respectively.

3 Local model-checking

This section is devoted to the model-checking of temporal formulas over finite LTSs. We first define the syntax and semantics of Parameterised Boolean Equation Systems (PBESs), next we describe how the model-checking problem can be translated to the partial resolution of a PBES, and finally we give a semi-decidable local model-checking algorithm performing this resolution.

3.1 Parameterised Boolean Equation Systems

A Parameterised Boolean Equation System (PBES for short) is a set of fixed point equations whose left hand sides are *boolean variables* $Z \in BVar$ parameterised by data variables, and whose right hand sides are *boolean formulas* $\psi \in BForm$. Each equation i has a *sign* $\sigma_i \in \{\mu, \nu\}$. The syntax of boolean formulas (given directly in positive form) and of PBESs is shown in Figure 3. We consider here only PBESs that are alternation-free, i.e., without mutual recursion between least and greatest boolean variables.

Boolean formulas	$\psi ::= tt \mid ff \mid e \rightarrow \psi_1\ []\ \psi_2 \mid \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2 \mid Z(\vec{e})$
PBESs	$\left\{ Z_i(\vec{x}_i{:}\vec{t}_i) \stackrel{\sigma_i}{=} \psi_i \right\}_{1 \leq i \leq p}$

Figure 3: Syntax of boolean formulas and PBESs

The semantics of these constructs is defined in Figure 4. Boolean formulas are interpreted w.r.t. *boolean environments* $\delta \in \mathbf{BEnv}$ and data environments, the former being partial functions mapping boolean variables $Z(\vec{x}{:}\vec{t})$ to functions in $\vec{t} \rightarrow \mathbf{Bool}$. The denotation $[\![\psi]\!]\,\delta\varepsilon$, where $[\![.]\!] : BForm \rightarrow \mathbf{BEnv} \rightarrow \mathbf{DEnv} \rightarrow \mathbf{Bool}$ is the interpretation function for boolean formulas, gives the truth value of ψ in the context of δ and ε.

Boolean formulas	$[\![tt]\!]\,\delta\varepsilon = \mathbf{tt}$ $[\![ff]\!]\,\delta\varepsilon = \mathbf{ff}$ $[\![e \to \psi_1\,[]\,\psi_2]\!]\,\delta\varepsilon = \mathit{if}\ [\![e]\!]\,\varepsilon\ \mathit{then}\ [\![\psi_1]\!]\,\delta\varepsilon\ \mathit{else}\ [\![\psi_2]\!]\,\delta\varepsilon$ $[\![\psi_1 \vee \psi_2]\!]\,\delta\varepsilon = [\![\psi_1]\!]\,\delta\varepsilon\ \mathit{or}\ [\![\psi_2]\!]\,\delta\varepsilon$ $[\![\psi_1 \wedge \psi_2]\!]\,\delta\varepsilon = [\![\psi_1]\!]\,\delta\varepsilon\ \mathit{and}\ [\![\psi_2]\!]\,\delta\varepsilon$ $[\![Z(\vec{e})]\!]\,\delta\varepsilon = \big(\delta(Z)\big)\big([\![\vec{e}]\!]\,\varepsilon\big)$
σ-blocks	$[\![\{Z_i(\vec{x}_i{:}\vec{t}_i) \stackrel{\sigma}{=} \psi_i\}_{1\le i\le p}]\!]\,\delta = \sigma\overline{\Psi}_\delta$ where $\overline{\Psi}_\delta(\vec{G}) = \big(\lambda\vec{v}_i{:}\vec{t}_i.\,[\![\psi_i]\!]\,(\delta \oslash [\vec{G}/\vec{Z}])(\varepsilon \oslash [\vec{v}_i/\vec{x}_i])\big)_{1\le i\le p}$
Pbess	$[\![\{B_j\}_{1\le j\le m}]\!]\,\delta = (\vec{G}_1, ..., \vec{G}_m)$ where $\vec{G}_j = [\![B_j]\!]\,\delta_j$, $\delta_1 = \delta$, $\delta_j = \delta_{j-1} \oslash [\vec{G}_{j-1}/\vec{Z}_{j-1}]$

Figure 4: Semantics of boolean formulas and Pbess

The interpretation of σ-blocks (i.e., Pbess having equations of the same sign σ) w.r.t. a boolean environment δ is defined as the σ-fixed point of an associated boolean vectorial functional $\overline{\Psi}_\delta$. Since the Pbess we consider are alternation-free, they can always be partitioned in σ-blocks $\{B_j\}_{1\le j\le m}$ such that for every $1 \le j \le m$, the σ-block B_j depends only on $B_1, ..., B_{j-1}$. The interpretation of a Pbes w.r.t. a boolean environment δ is defined by the union of the semantics of its σ-blocks B_j (for $1 \le j \le m$), where each B_j is interpreted w.r.t. the semantics of $B_1, ..., B_{j-1}$.

3.2 Transformation of the verification problem

The model-checking problem, requiring to compute the truth value of a state formula φ of our logic over a finite Lts model, can be reduced to the partial resolution of a Pbes. This transformation, which generalizes the corresponding translations used in the framework of the standard alternation-free μ-calculus [1, 14], consists of two phases, briefly described below.

Normalisation. This phase performs a syntactical translation of φ into a formula φ' that is *normalised*, i.e., every fixed point subformula $\sigma_i Y_i(\vec{x}_i{:}\vec{t}_i{:=}\vec{e}_i).\varphi_i$ of φ' satisfies the following two properties. Firstly, φ_i is *fully parameterised*, i.e., all data variables free in φ_i are among the parameters $\vec{x}_i$ (this is obtained by adding to Y_i extra data parameters, initialised with the values of the free variables of φ_i and propagated through the calls of Y_i). Secondly, φ_i is *simple*, i.e., all its direct subformulas are atomic boolean formulas, propositional variables, or fixed point formulas (this is obtained by introducing dummy propositional variables, whose signs are inherited from the immediately enclosing fixed point operator; in order to simplify the second phase of the transformation, a dummy propositional variable is also added on top of the whole formula).

It can be shown that the normalisation preserves the semantics of state formulas and may cause at most a linear blow-up of their size.

Composition. This phase builds a Pbes by composing a normalised formula φ and an Lts model $M = \langle S, A, T, s_0\rangle$ in the following manner. For each fixed point subformula $\sigma_i Y_i(\vec{x}_i{:}\vec{t}_i{:=}\vec{e}_i).\varphi_i$ of φ and for each state $s \in S$ of the Lts, an equation of sign σ_i is constructed, whose left hand side is a boolean variable $Z_{i,s}(\vec{x}_i{:}\vec{t}_i)$ and whose right hand side is a boolean formula $(\varphi_i)_s$. Intuitively, a state s satisfies a subformula $\sigma_i Y_i(\vec{x}_i{:}\vec{t}_i{:=}\vec{e}_i).\varphi_i$ iff $Z_{i,s}(\vec{e}_i)$ is true. The construction of a Pbes from a state formula (containing the propositional variables $Y_1, ..., Y_n$) and an Lts is illustrated in Figure 5. (Note: in the translation of the modal formulas $(\langle\alpha\rangle\,\varphi)_s$ and $([\alpha]\,\varphi)_s$, the notation $(\varphi)'_s([\![\alpha]\!]\,[\,]a)_2$ represents the substitution in $(\varphi)'_s$ of the variables exported by α with the values extracted from a using the data environment $([\![\alpha]\!]\,[\,]a)_2$ produced by interpreting α on a.)

A partition of the resulting Pbes in σ-blocks can be naturally obtained by grouping together the equations corresponding to the fixed point operators that were adjacent in the normalised formula φ. Since the transition relation of the Lts is explored forward during the translation, the Pbes can be constructed by generating the Lts locally, i.e., in a demand-driven way.

$$\{Z_{i,s}(\vec{x}_i{:}\vec{t}_i) \stackrel{\sigma_i}{=} (\varphi_i)_s\}_{1\leq i\leq n, s\in S}$$

$$\begin{array}{rcl}
(e \rightarrow \varphi_1 \,[]\, \varphi_2)_s & = & e \rightarrow (\varphi_1)_s \,[]\, (\varphi_2)_s \\
(\varphi_1 \vee \varphi_2)_s & = & (\varphi_1)_s \vee (\varphi_2)_s \\
(\varphi_1 \wedge \varphi_2)_s & = & (\varphi_1)_s \wedge (\varphi_2)_s \\
(\langle\alpha\rangle\,\varphi)_s & = & \bigvee_{s \xrightarrow{a} s'} \big(a \models \alpha \wedge (\varphi)_{s'}([\![\alpha]\!]\,[\,]a)_2\big) \\
([\alpha]\,\varphi)_s & = & \bigwedge_{s \xrightarrow{a} s'} \big(a \models \alpha \rightarrow (\varphi)_{s'}([\![\alpha]\!]\,[\,]a)_2\big) \\
(Y_i(\vec{e}_i))_s & = & Z_{i,s}(\vec{e}_i) \\
G'\ \vec{v}' \models G\ \vec{x}{:}\vec{t} & = & (G' = G) \wedge (\vec{v}' \in \vec{t}) \\
G'\ \vec{v}' \models G\ \vec{e} & = & (G' = G) \wedge (\vec{v}' = \vec{e}) \\
G'\ \vec{v}' \models \neg\alpha & = & \neg(G'\ \vec{v}' \models \alpha) \\
G'\ \vec{v}' \models \alpha_1 \wedge \alpha_2 & = & (G'\ \vec{v}' \models \alpha_1) \wedge (G'\ \vec{v}' \models \alpha_2)
\end{array}$$

Figure 5: Construction of a PBES from a state formula and an LTS

3.3 Resolution algorithm

As shown in Section 3.2, the verification of a formula φ on the initial state s_0 of an LTS can be reduced to the calculation of a variable instance $Z_{0,s_0}(\vec{v}_0)$ of a PBES (where Y_0 is the propositional variable on top of the normalised form of φ and $\vec{v}_0$ are the values of its data parameters). We provide here a resolution algorithm for computing a variable instance belonging to a σ-block. This algorithm can also be used to compute a variable instance of an alternation-free PBES, by applying it iteratively on the σ-blocks resulted from a partition of the PBES. We focus here on the case $\sigma = \mu$, the case $\sigma = \nu$ being completely dual.

The resolution algorithm, shown in Figure 6, consists of a main function SOLVE and an auxiliary function EVAL. The SOLVE function uses an algorithm similar in spirit to, but more general then, the local boolean resolution algorithm given in [14]. Starting from the instance $Z_i(\vec{v}_i)$ to be calculated, SOLVE performs a forward exploration of the PBES, repeatedly expanding (i.e., evaluating ψ_j with the arguments $\vec{v}_j$, using the EVAL function) the instances $Z_j(\vec{v}_j)$ already reached, which are stored in a variable *visited*. The instances already expanded and the dependencies between them are stored in the variables *expanded* and *depend*, respectively. The instances evaluated to **tt** (which are therefore "stable") are kept in a work list *to_be_propagated* and (since they may cause other instances to become "unstable") their effects are propagated in the dependency graph, by reevaluating the other instances depending on them. The process stops when no new instances are needed in order to establish the truth value of $Z_i(\vec{v}_i)$. The EVAL function uses additional counter variables in order to optimize the reevaluation of instances (this technique has been used in the linear-time model-checking algorithms [4, 1] for the standard alternation-free μ-calculus).

The SOLVE function has a complexity linear in the size of the dependency graph between instances generated during the resolution, since each edge of the dependency graph is traversed (at most) twice: firstly, when its source instance is expanded, and secondly, when its target instance is (possibly) propagated. However, due to the fact that the data parameters of the PBES (inherited from the initial formula φ) may belong to infinite domains (e.g., natural numbers, lists, sets, etc.), the termination of SOLVE becomes undecidable in the general case, therefore making (extremely) difficult to predict the size of the dependency graph explored. We can nevertheless examine the complexity for particular classes of formulas: e.g., for fixed point formulas without parameters (but containing data variables in the modalities), the size of the dependency graph cannot exceed the size of the whole PBES, which is linear in the size of the initial formula (number of operators) and of the LTS (number of states and transitions). This happens also for the "pure" μ-calculus fragment of our logic (without any data variables), in which case SOLVE has the linear-time complexity of the best local algorithms available for the standard alternation-free μ-calculus [1, 14]. Moreover, for many of the data-based properties usually encountered (such as those presented in Section 2.3), the domains of the fixed point data parameters are bounded by the values contained in the LTS (number of processes, number of messages transmitted, etc.), thus ensuring the termination of the SOLVE function. Therefore, we expect in practice a good performance of our resolution algorithm.

```
function SOLVE ({Z_k(x⃗_k : t⃗_k) =^μ ψ_k}_{1≤k≤p} : BSys, Z_i : BVar, v⃗_i : t⃗_i, δ : BEnv) : Bool is
  var expanded : (BVar × Par) → (Bool × Nat),
      visited, to_be_propagated, needed : 2^{BVar×Par},
      depend : (BVar × Par) → 2^{BVar×Par}, b : Bool, c : Nat;
  visited := {(Z_i, v⃗_i)}; expanded := [ ];
  to_be_propagated := ∅; depend := [ ];
  repeat
    if visited ≠ ∅ then
      let (Z_j, v⃗_j) ∈ visited; visited := visited \ {(Z_j, v⃗_j)};
      (needed, b, c) := EVAL (ψ_j, δ, [v⃗_j/x⃗_j], expanded);
      expanded := expanded ∪ [(b, c)/(Z_j, v⃗_j)];
      if b then to_be_propagated := to_be_propagated ∪ {(Z_j, v⃗_j)} endif;
      forall (Z_l, v⃗_l) ∈ needed do
        depend := depend ∪ [{(Z_j, v⃗_j)}/(Z_l, v⃗_l)];
        if (Z_l, v⃗_l) ∉ visited then visited := visited ∪ {(Z_l, v⃗_l)} endif
      end
    elsif to_be_propagated ≠ ∅ then
      let (Z_l, v⃗_l) ∈ to_be_propagated; to_be_propagated := to_be_propagated \ {(Z_l, v⃗_l)};
      forall (Z_j, v⃗_j) ∈ depend(Z_l, v⃗_l) do
        (b, c) := expanded(Z_j, v⃗_j);
        if ¬b then
          expanded := expanded ⊘ [(c = 1, c − 1)/(Z_j, v⃗_j)];
          if c = 1 then to_be_propagated := to_be_propagated ∪ {(Z_j, v⃗_j)} endif
        endif
      end
    endif
  until visited = ∅ ∧ to_be_propagated = ∅;
  return (expanded(Z_i, v⃗_i))_1
end

function EVAL (ψ : BForm, δ : BEnv, ε : DEnv,
               expanded : (BVar × Par) → (Bool × Nat)) : 2^{BVar×Par} × Bool × Nat
is var needed_1, needed_2 : 2^{BVar×Par}, b_1, b_2 : Bool, c_1, c_2 : Nat;
  case ψ in
    tt → return ( ∅, tt, 0)
    ff → return ( ∅, ff, 0)
    Z_j(e⃗_j) → if Z_j ∈ supp(δ) then return ({(Z_j, [[e⃗_j]] ε)}, (δ(Z_j))([[e⃗_j]] ε), 0)
               elsif (Z_j, [[e⃗_j]] ε) ∈ supp(expanded) then
                 return ({(Z_j, [[e⃗_j]] ε)}, expanded(Z_j, [[e⃗_j]] ε))
               else return ({(Z_j, [[e⃗_j]] ε)}, ff, 1) endif
    ψ_1 or ψ_2 →  (needed_1, b_1, c_1) := EVAL (ψ_1, δ, ε, expanded);
                  (needed_2, b_2, c_2) := EVAL (ψ_2, δ, ε, expanded);
                  return (needed_1 ∪ needed_2, b_1 or b_2, max (c_1, c_2))
    ψ_1 and ψ_2 → (needed_1, b_1, c_1) := EVAL (ψ_1, δ, ε, expanded);
                  (needed_2, b_2, c_2) := EVAL (ψ_2, δ, ε, expanded);
                  return (needed_1 ∪ needed_2, b_1 and b_2, c_1 + c_2)
    e → ψ_1 [] ψ_2 → if [[e]] ε then return EVAL(ψ_1, δ, ε, expanded)
                     else return EVAL(ψ_2, δ, ε, expanded) endif
  endcase
end
```

Figure 6: The SOLVE and EVAL functions

4 Conclusion and future work

We defined in this paper an extended alternation-free μ-calculus allowing to express temporal properties involving data. The formulas are interpreted over LTSs generated from programs written in value-passing description languages such as μCRL or LOTOS. We proposed a model-checking method for this logic, based on the reduction of the verification problem to the partial resolution of a Parameterised Boolean Equation System (PBES), and we gave a semi-decidable local model-checking algorithm performing this resolution.

Several directions for further work can be considered. Firstly, the algorithm has to be completely implemented within the μCRL verification toolbox [6] and experimented on real-life applications. Secondly, the model-checking method presented here should be extended to the full μ-calculus (i.e., of arbitrary alternation depth), for instance by investigating the extension with data of the efficient local model-checking algorithm recently proposed in [11]. Finally, it would be interesting to apply the resolution algorithm proposed here in the framework of abstract interpretation, where similar techniques are used for efficient fixed point computation [3].

Acknowledgements

We are grateful to the anonymous referees for their valuable comments on this paper.

References

[1] H. R. Andersen. Model checking and boolean graphs. *Theoretical Computer Science*, 126(1):3–30, April 1994.

[2] A. Arnold and P. Crubillé. A linear algorithm to solve fixed-point equations on transition systems. *Information Processing Letters*, 29:57–66, 1988.

[3] L-L. Chen. Efficient Computation of Fixpoints that Arise in Abstract Interpretation. Technical Report UIUCDCS-R-94-1866, Univ. of Illinois, Urbana-Champaign, July 1994.

[4] R. Cleaveland and B. Steffen. A Linear-Time Model-Checking Algorithm for the Alternation-Free Modal Mu-Calculus. In K. G. Larsen and A. Skou, editors, *Proceedings of CAV'91*, volume 575 of *LNCS*, pages 48–58, Berlin, July 1991. Springer Verlag.

[5] M. Dam, L a. Fredlund, and D. Gurov. Compositional Verification of Erlang Programs. In J.F. Groote, B. Luttik, and J. van Wamel, editors, *3rd Int. Workshop on Formal Methods for Industrial Critical Systems*, pages 127–156, CWI, Amsterdam, May 1998.

[6] D. Dams and J. F. Groote. Specification and Implementation of Components of a μCRL Toolbox. Technical Report Logic Group Preprint Series 152, Utrecht University, December 1995.

[7] J. F. Groote and A. Ponse. The syntax and semantics of μCRL. Technical Report CS-R9076, CWI, Amsterdam, December 1990.

[8] J. F. Groote and J. C. van de Pol. A bounded retransmission protocol for large data packets. Technical Report Logic Group Preprint Series 100, Utrecht University, October 1993.

[9] ISO/IEC. LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807, ISO – OSI, Genève, September 1988.

[10] D. Kozen. Results on the Propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[11] X. Liu, C. R. Ramakrishnan, and S. A. Smolka. Fully Local and Efficient Evaluation of Alternating Fixed Points. In Bernhard Steffen, editor, *Proceedings of TACAS'98*, volume 1384 of *LNCS*, Berlin, March 1998. Springer Verlag.

[12] J. Rathke and M. Hennessy. Local model checking for a value-based modal μ-calculus. Report 5/96, School of Cognitive and Computing Sciences, University of Sussex, June 1996.

[13] A. Tarski. A Lattice-Theoretical Fixpoint Theorem and its Applications. *Pacific Journal of Mathematics*, (5):285–309, 1955.

[14] B. Vergauwen and J. Lewi. Efficient Local Correctness Checking for Single and Alternating Boolean Equation Systems. In S. Abiteboul and E. Shamir, editors, *Proceedings of the 21st ICALP*, volume 820 of *LNCS*, pages 304–315, Berlin, July 1994. Springer Verlag.