# Verification of Temporal Properties of Processes in a Setting with Data

Jan Friso Groote[1,2] and Radu Mateescu[3]*

[1] CWI, P.O. Box 94079, NL-1090 GB Amsterdam, The Netherlands
[2] Eindhoven University of Technology, P.O. Box 513
NL-5600 MB Eindhoven, The Netherlands
JanFriso.Groote@cwi.nl
[3] INRIA Rhône-Alpes / VASY group, 655, avenue de l'Europe
F-38330 Montbonnot Saint Martin, France
Radu.Mateescu@inria.fr

**Abstract.** We define a value-based modal $\mu$-calculus, built from first-order formulas, modalities, and fixed point operators parameterized by data variables, which allows to express temporal properties involving data. We interpret this logic over $\mu$CRL terms defined by linear process equations. The satisfaction of a temporal formula by a $\mu$CRL term is translated to the satisfaction of a first-order formula containing parameterized fixed point operators. We provide proof rules for these fixed point operators and show their applicability on various examples.

## 1 Introduction

In recent years we have applied process algebra in numerous settings [4,8,12]. The first lesson we learned is that process algebra *pur sang* is not very handy, and we need an extension with data. This led to the language $\mu$CRL (*micro Common Representation Language*) [13]. The next observation was that it is very convenient to eliminate the parallel operator from a process description and reduce it to a very restricted form, which we call a *linear process equation* or *linear process operator* [3]. Such an elimination can be done automatically [5,9] and generally yields a compact result, of the same size as the original system description. For proving equations of the form *specification=implementation*, a proof methodology has been developed [14] and has been applied to numerous examples (see e.g. [4,8,11,20]) that all have infinite or unbounded state spaces.

An obvious question that has not been addressed thus far is whether the linear process format can also be employed in proving temporal logic formulas. In this paper we provide a way of doing so that roughly goes as follows. First, we extend the modal $\mu$-calculus [16] to express properties about data, meaning that we include boolean expressions on data variables, parameterization of actions contained in the modalities, quantification over data, and parameterization

* This work has been funded by the grant no. 97-09 of the ERCIM fellowship programme for collaboration between INRIA and CWI.

A.M. Haeberer (Ed.): AMAST'98, LNCS 1548, pp. 74–90, 1998.

of minimal and maximal fixed point operators. A typical example of temporal property expressed in this logic is

$$\big(\nu Y(n{:}\mathrm{N}).\exists m{:}\mathrm{N}.\, \langle a(m+n)\rangle\, Y(m+n)\big)(2)$$

describing the states from which an infinite sequence of actions $a(i_0)a(i_1)a(i_2)\cdots$ can be performed, where $2 \leq i_0 \leq i_1 \leq i_2 \leq \cdots$. Another example of formula is

$$\forall i{:}\mathrm{N}.[a(i)](i > n)$$

stating that whenever an $a(i)$ action can be performed, $i$ must be larger than $n$.

The second step is to prove that a given linear process satisfies such a temporal formula. To achieve this, we first transform both the process and the temporal formula into a first-order fixed point formula. This approach is similar to the model-checking algorithms in [2,22,1], where a formula of standard $\mu$-calculus (i.e., without data) and a finite state automaton are combined to form a set of fixed point boolean equations, which can be solved in linear time, provided the formula is alternation-free. In our setting, this transformation applies to the full logic (formulas of arbitrary alternation depth), is purely syntactical, and in many cases can be carried out by hand, as both the linear process and the temporal formula are generally quite small.

In order to solve the first-order fixed point formulas obtained in this way, we use the standard proof rules for connectives and quantifiers, and we introduce a set of proof rules for fixed point operators allowing to approximate (towards either satisfaction, or refutation) the fixed point (sub)formulas. If the initial state of the process satisfies an approximation of a maximal fixed point formula, we know that it satisfies the maximal fixed point too. The approximation of minimal fixed points captures the fact that the property expressed by a minimal fixed point formula must be reached in a finite number of steps. These rules reflect the proof principles for safety and liveness properties discussed in [17].

We included a simple example and a slightly more elaborate one, in order to show how the proof method that we propose can be used. We have also successfully applied the method to verify a distributed summing protocol [11], but due to space limitations we have not included it in this paper. All these examples are quite promising, as they show that our method leads to straightforward arguments of validity of the temporal formulas.

Other approaches to prove temporal properties involving data that we are aware of [19,7] use tableau-based methods, often directed towards decomposing the property over the system. The approach we adopt here is different, being intended to facilitate manual verification in the natural deduction style (see also [15]). Since the linear processes obtained from $\mu$CRL specifications are generally small, we expect a good applicability of our method to various examples.

The paper is organized as follows. Section 2 defines the linear $\mu$CRL processes and their models. Section 3 gives the syntax and semantics of the extended $\mu$-calculus that we propose, together with examples of temporal properties. Section 4 presents the verification method, i.e., the translation into first-order fixed point formulas and the proof rules for extremal fixed points. Finally, Section 5 shows the application of this method on an infinite-state linear $\mu$CRL process.

## 2 Preliminaries

We define below the notions of data expression, linear process, and labeled transition system (LTS), over which the temporal logic formulas will be interpreted.

### 2.1 Expressions

The set *Exp* of *data expressions* is defined over a set *DVar* of *data variables* and a set *Func* of *functions*. Each data variable $x \in DVar$ has a type $D$ and each function $f \in Func$ has a profile $D_1 \times \cdots \times D_n \rightarrow D$, where $D_1, \ldots, D_n$ are the argument types of $f$ and $D$ is its result type. We write **Val** for the domain containing all the values belonging to the types $D$. The expressions $e \in Exp$ are defined by the following grammar:

$$e ::= x \mid f(e_1, \ldots, e_n)$$

The set of variables occurring in an expression $e$ is noted $var(e)$.

We define the domain $\mathbf{DEnv} = DVar \rightarrow \mathbf{Val}$ of *data environments*. A data environment $\varepsilon \in \mathbf{DEnv}$ is a partial function mapping data variables into values of their corresponding types. The *support* of an environment $\varepsilon$, noted $supp(\varepsilon)$, denotes the set of variables that are assigned a value in **Val** by $\varepsilon$. An environment mapping the variables $x_1, \ldots, x_n$ respectively to the values $v_1, \ldots, v_n$ is noted $[v_1/x_1, \ldots, v_n/x_n]$. The environment having an empty support is noted $[\,]$. The *overriding* of $\varepsilon$ by $[v_1/x_1, \ldots, v_n/x_n]$ is the data environment defined as follows: $(\varepsilon[v_1/x_1, \ldots, v_n/x_n])(x) = \mathit{if}\ \exists i \in [1,n].x = x_i\ \mathit{then}\ v_i\ \mathit{else}\ \varepsilon(x)$.

The semantics of data expressions is given by the interpretation function $[\![.]\!] : Exp \rightarrow \mathbf{DEnv} \rightarrow \mathbf{Val}$, defined inductively below. For an expression $e$ and a data environment $\varepsilon$ such that $var(e) \subseteq supp(\varepsilon)$, $[\![e]\!]\,\varepsilon$ denotes the value of $e$ in the context of $\varepsilon$:

$$[\![x]\!]\,\varepsilon \overset{\text{def}}{=} \varepsilon(x)$$

$$[\![f(e_1, \ldots, e_n)]\!]\,\varepsilon \overset{\text{def}}{=} f([\![e_1]\!]\,\varepsilon, \ldots, [\![e_n]\!]\,\varepsilon)$$

We assume that the domain $\mathbf{Bool} = \{\mathbf{tt}, \mathbf{ff}\}$ of boolean values is predefined, together with the usual operations $\wedge$, $\vee$, $\neg$, and $\rightarrow$. Boolean expressions are denoted by the symbol $b$.

### 2.2 Linear Processes

Linear processes share with LTSs the advantage of being a simple, straightforward notation, suitable for further analysis of processes in either automatic or manual form. But they do not share the most important disadvantage, namely the exponential blow-up caused by the parallel operator (see [5]). As we are interested in devising analysis methods for realistic distributed systems, it is clear that LTSs are not satisfactory. Therefore, we use the linear processes, of which we give a definition below.

Let Act be a set of actions, which may be parameterized by data values.

**Definition 1.** *Let $Act \subseteq \mathsf{Act} \cup \{\tau\}$ be a finite set of actions and $D$, $D_a$, $E_a$ be data types. A* linear process *over $Act$ and $D$ is defined by an equation of the following form:*

$$X(x{:}D) = \sum_{a \in Act} \sum_{x_a : E_a} a(e_a) \cdot X(e'_a) \lhd b_a \rhd \delta$$

*where $x$ is a parameter of type $D$, and for each action $a \in Act$, $x_a$ is a variable of type $E_a$, $e_a$ and $e'_a$ are expressions of type $D_a$ and $D$, respectively, and $b_a$ is an expression of type* **Bool**, *such that $var(e_a) \cup var(e'_a) \cup var(b_a) \subseteq \{x, x_a\}$. The constant $\delta$, called* deadlock, *cannot perform any action. The* initial state *of process $X$ may be specified by giving an initial value $v_0 \in D$ for $x$.*

A linear process expression must be read as follows. If a process is in state $x$, then it can perform actions $a(e_a)$ provided a value of $x_a$ in $E_a$ can be found such that $b_a$ holds. In such a case, the process ends up in a state $e'_a$.

For simplicity, we allow at most one data parameter for any action $a \in Act$ (we assume that $\tau$ has a dummy parameter) and for each linear process $X$. Using pairing and projection, the formalization can be straightforwardly used with multiple parameters.

### 2.3 Transition Systems

We consider a linear $\mu$CRL process $X$ as in Definition 1. According to the operational semantics of $\mu$CRL [13], the transition system modeling a linear process is defined as follows.

**Definition 2.** *The* transition system *of a linear process is a quadruple $M = (S, L, \rightarrow, s_0)$, where:*

- $S \overset{\text{def}}{=} \{X(v) \mid v \in D\}$ *is the set of* states;
- $L \overset{\text{def}}{=} \{a(v_a) \mid a \in Act \wedge v_a \in D_a\}$ *is the set of* labels;
- $\rightarrow \overset{\text{def}}{=} \{X(v) \xrightarrow{a(v'_a)} X(v') \mid a \in Act \wedge \exists v_a \in E_a.([\![b_a]\!]\,[v/x, v_a/x_a] \wedge v'_a = [\![e_a]\!]\,[v/x, v_a/x_a] \wedge v' = [\![e'_a]\!]\,[v/x, v_a/x_a])\}$ *is the* transition relation;
- $s_0 \overset{\text{def}}{=} X(v_0) \in S$ *is the* initial state.

The definition of the initial state of the process is not mandatory, unless there are properties of $X$ that must be explicitly verified on $X(v_0)$.

## 3 Temporal Logic

The logic we consider is based upon an extension of the modal $\mu$-calculus [16] with data variables, quantifiers, and parameterization, in order to express properties involving data. Other similar value-based formalisms extending the modal $\mu$-calculus have been used in the framework of symbolic transition systems [19] and of the polyadic $\pi$-calculus [7].

The logic we propose here contains a set *AForm* of *action formulas* and a set *SForm* of *state formulas*, whose syntax and semantics are defined below. To simplify the notations, we implicitly consider throughout this section a transition system $M = (S, L, \rightarrow, s_0)$, over which the formulas are interpreted.

The action formulas $\alpha \in AForm$ are defined by the following grammar:

$$\alpha ::= a(e) \mid tt \mid \neg\alpha \mid \alpha_1 \wedge \alpha_2 \mid \exists y{:}D.\alpha$$

where $a \in Act$, $e \in Exp$, and $y \in DVar$ is a data variable of type $D$. The usual derived operators are also allowed: $ff = \neg tt$, $\alpha_1 \vee \alpha_2 = \neg(\neg\alpha_1 \wedge \neg\alpha_2)$, $\alpha_1 \rightarrow \alpha_2 = \neg\alpha_1 \vee \alpha_2$, $\forall y{:}D.\alpha = \neg\exists y{:}D.\neg\alpha$. Data variables are bound by quantifiers in the usual way. The set of free data variables of a formula $\alpha$ is noted $fdv(\alpha)$.

The semantics of action formulas is given by the interpretation function $[\![.]\!] : AForm \rightarrow \mathbf{DEnv} \rightarrow 2^L$, defined inductively below. Given an action formula $\alpha$ and a data environment $\varepsilon$ such that $fdv(\alpha) \subseteq supp(\varepsilon)$, $[\![\alpha]\!]\,\varepsilon$ denotes the set of labels satisfying $\alpha$ in the context of $\varepsilon$:

$$\begin{aligned}
[\![a(e)]\!]\,\varepsilon &\stackrel{\text{def}}{=} \{a([\![e]\!]\,\varepsilon)\}\\
[\![tt]\!]\,\varepsilon &\stackrel{\text{def}}{=} L\\
[\![\neg\alpha]\!]\,\varepsilon &\stackrel{\text{def}}{=} L \setminus [\![\alpha]\!]\,\varepsilon\\
[\![\alpha_1 \wedge \alpha_2]\!]\,\varepsilon &\stackrel{\text{def}}{=} [\![\alpha_1]\!]\,\varepsilon \cap [\![\alpha_2]\!]\,\varepsilon\\
[\![\exists y{:}D.\alpha]\!]\,\varepsilon &\stackrel{\text{def}}{=} \textstyle\bigcup_{v \in D} [\![\alpha]\!]\,\varepsilon[v/y].
\end{aligned}$$

The state formulas $\varphi \in SForm$, built over the set $AForm$ and over a set $PVar$ of *propositional variables*, are defined by the following grammar:

$$\varphi ::= b \mid Y(e) \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\alpha\rangle\,\varphi \mid \exists y{:}D.\varphi \mid \big(\mu Y(y{:}D).\varphi\big)(e)$$

where $b \in Exp$ is a boolean expression, $Y \in PVar$ is a (parameterized) propositional variable, $\alpha \in AForm$ is an action formula and $y \in DVar$ is a data variable of type $D$. Besides the usual derived connectives, we also define the box modal operator $[\alpha]\,\varphi = \neg\,\langle\alpha\rangle\,\neg\varphi$ and the maximal fixed point operator $(\nu Y(y{:}D).\varphi)(e) = \neg(\mu Y(y{:}D).\neg\varphi[\neg Y/Y])(e)$, where $\varphi[\neg Y/Y]$ denotes the syntactic substitution of $Y$ by $\neg Y$ in $\varphi$. In the sequel, we let $\sigma$ range over $\{\mu, \nu\}$.

Data variables are bound by quantifiers and by parameterization, and propositional variables are bound by fixed point operators, in the usual way. The sets of free data variables and free propositional variables of $\varphi$ are noted $fdv(\varphi)$ and $fpv(\varphi)$, respectively. A formula $\varphi$ is said *closed* if $fdv(\varphi) = \emptyset$ and $fpv(\varphi) = \emptyset$.

We assume that state formulas are *syntactically monotonic*, i.e., for each formula $(\sigma Y(y{:}D).\varphi)(e)$, every free occurrence of $Y$ in $\varphi$ falls under an even number of negations. This enables to convert any formula $\varphi$ in *Positive Normal Form* (PNF for short) by pushing the negations downwards to its atomic subformulas and (if necessary) by $\alpha$-converting it such that there is no variable $Y$ having both free and bound occurrences in $\varphi$. In the sequel, we consider only closed state formulas in PNF.

We define the domain $\mathbf{PEnv} = PVar \to (\mathbf{Val} \to 2^S)$ of *propositional environments*. A propositional environment $\rho \in \mathbf{PEnv}$ is a partial function mapping propositional variables to functions from the domains of their parameters to sets of transition system states. The support, bracketed notation, and overriding of propositional environments are defined in the same way as for data environments.

The semantics of state formulas is given by the interpretation function $\llbracket . \rrbracket : SForm \to \mathbf{PEnv} \to \mathbf{DEnv} \to 2^S$, defined inductively below. For a state formula $\varphi$, a propositional environment $\rho$, and a data environment $\varepsilon$ such that $fpv(\varphi) \subseteq supp(\rho)$ and $fdv(\varphi) \subseteq supp(\varepsilon)$, $\llbracket \varphi \rrbracket \rho\varepsilon$ denotes the set of states satisfying $\varphi$ in the context of $\rho$ and $\varepsilon$:

$$\begin{aligned}
\llbracket b \rrbracket \rho\varepsilon &\overset{\text{def}}{=} \textit{if } \llbracket b \rrbracket \varepsilon \textit{ then } S \textit{ else } \emptyset \\
\llbracket Y(e) \rrbracket \rho\varepsilon &\overset{\text{def}}{=} (\rho(Y))(\llbracket e \rrbracket \varepsilon) \\
\llbracket \varphi_1 \wedge \varphi_2 \rrbracket \rho\varepsilon &\overset{\text{def}}{=} \llbracket \varphi_1 \rrbracket \rho\varepsilon \cap \llbracket \varphi_2 \rrbracket \rho\varepsilon \\
\llbracket \langle \alpha \rangle \varphi \rrbracket \rho\varepsilon &\overset{\text{def}}{=} \{X(v) \in S \mid \exists v' \in D . \exists a \in Act . \exists v_a \in D_a . \\
&\qquad\qquad X(v) \xrightarrow{a(v_a)} X(v') \wedge a(v_a) \in \llbracket \alpha \rrbracket \varepsilon \wedge X(v') \in \llbracket \varphi \rrbracket \rho\varepsilon\} \\
\llbracket \exists y{:}D . \varphi \rrbracket \rho\varepsilon &\overset{\text{def}}{=} \{X(v) \in S \mid \exists v' \in D . X(v) \in \llbracket \varphi \rrbracket \rho(\varepsilon[v'/y])\} \\
\llbracket (\mu Y(y{:}D) . \varphi)(e) \rrbracket \rho\varepsilon &\overset{\text{def}}{=} (\mu \Phi_{\rho\varepsilon})(\llbracket e \rrbracket \varepsilon)
\end{aligned}$$

where the functional $\Phi_{\rho\varepsilon} : (D \to 2^S) \to (D \to 2^S)$, associated to the formula $\mu Y(y{:}D).\varphi$, is defined as $\Phi_{\rho\varepsilon} = \lambda F{:}D \to 2^S . \lambda v{:}D . \llbracket \varphi \rrbracket (\rho[F/Y])(\varepsilon[v/y])$.

It is straightforward to check that, for state formulas in PNF, every functional $\Phi_{\rho\varepsilon}$ associated to a fixed point (sub)formula is monotonic over $D \to 2^S$. Since the underlying lattices $D \to 2^S$ are complete, it follows from Tarski's theorem [21] that every $\Phi_{\rho\varepsilon}$ functional has a unique minimal fixed point $\mu\Phi_{\rho\varepsilon}$ and a unique maximal fixed point $\nu\Phi_{\rho\varepsilon}$.

### 3.1 Example

We describe a simple infinite state process, together with some temporal properties, in order to illustrate the techniques presented in here. In Section 4.3 we will translate the temporal formulas and in Section 4.5 we will prove the validity of the first-order fixed point formulas that we have obtained this way. The example is given by the following linear process equation, describing a slot machine:

$$\begin{aligned}
X(v{:}\mathrm{N}, b{:}\mathbf{Bool}) = \quad & s \cdot X(v+1, \neg b) \triangleleft \neg b \triangleright \delta \; + \\
& \textstyle\sum_{m:\mathrm{N}} w(m) \cdot X(v-m, \neg b) \triangleleft b \wedge m \leq v \triangleright \delta
\end{aligned}$$

The parameters $v$ and $b$ denote the current amount of money and the current state of the machine, respectively. When $b$ equals *ff*, a user can activate the machine by inserting a coin (action $s$); afterwards, $b$ becomes *tt* and the machine will deliver the money $m$ won by the user (action $w(m)$). The initial state of the system is $X(v_0, \mathit{ff})$, for some fixed $v_0 \geq 0$. (Actually, the linear process above allows a user to collect any amount of money he wants, but for the sake of th example we do not complicate the slot machine in order to avoid this.)

We are interested in the temporal properties below.

1. A basic liveness property is that, for any amount of money $l \in \mathrm{N}$, the machin can potentially deliver it to a user:

$$\varphi_1 \stackrel{\text{def}}{=} \mu Y.\, \langle w(l)\rangle\, tt \vee \langle tt\rangle\, Y$$

2. A stronger liveness property would be that, for any amount of money $l \in$ Ɲ the machine must eventually deliver it:

$$\varphi_2 \stackrel{\text{def}}{=} \mu Y.\, \langle tt\rangle\, tt \wedge [\neg w(l)]\, Y$$

3. A basic safety property is that every $l \in \mathrm{N}$ won in a $w(l)$ action cann exceed the initial amount of money $v_0$ of the machine, updated with the and $r$ money that have been inserted and won by users since the initial sta of the system, respectively:

$$\varphi_3 \stackrel{\text{def}}{=} \big(\nu Y(p, r{:}\mathrm{N}).\forall l{:}\mathrm{N}.\, [w(l)]\, (l \leq v_0{+}p{-}r \wedge Y(p, r{+}l)) \wedge [s]\, Y(p{+}1, r)\big)(0, 0$$

Clearly, $\varphi_1$ and $\varphi_3$ are valid for $X$, but $\varphi_2$ does not hold.

## 4 Verification

The verification problem consists to check whether a transition system $M$ (give by a linear $\mu$CRL process) satisfies a given temporal formula $\varphi$. Two differe cases are usually distinguished: *global* verification, consisting to decide if all t states of $M$ satisfy $\varphi$, and *local* verification, consisting to decide if one particul state (e.g., the initial state $s_0$) of $M$ satisfies $\varphi$. Both instances of the problem ca be reduced to the satisfaction of a first-order fixed point formula. First we defi the language of first-order fixed point formulas, next we describe the translati of a model $M$ and a state formula $\varphi$ into a first-order fixed point formula, a finally we provide sound proof rules for reasoning about fixed point operators

### 4.1 First-Order Fixed Point Formulas

We define the syntax and semantics of the set *BForm* of *first-order fixed poi formulas*, which will be used as an intermediate formalism for verification pu poses. The formulas $\psi \in \mathit{BForm}$, built over a set $\mathit{BVar}$ of *boolean variables*, a defined by the following grammar:

$$\psi ::= b \mid Z(e) \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \exists z{:}D.\psi_1 \mid \big(\mu Z(z{:}D).\psi_1\big)(e)$$

where $b \in \mathit{Exp}$ is a boolean expression and $Z \in \mathit{BVar}$ is a (parameterize boolean variable. The derived boolean, first-order, modal, and fixed point op ators are defined as usual. The data and boolean variables are bound in a mann similar to the state formulas $\varphi$. The sets of free data variables and free boolean variables of $\psi$ are noted $fdv(\psi)$ and $fbv(\psi)$, respectively. For simplicity, we use only one data parameter in first-order fixed point formulas; the formalization could be easily extended to allow multiple parameters. In the same way as for state formulas, we consider here only closed first-order fixed point formulas that have been translated in PNF.

We introduce the domain $\mathbf{BEnv} = BVar \rightarrow (\mathbf{Val} \rightarrow \mathbf{Bool})$ of *boolean environments.* A boolean environment $\eta \in \mathbf{BEnv}$ is a partial function mapping boolean variables to predicates over the domains of the data parameters. The support, bracketed notation, and overriding of boolean environments are defined in the same way as for propositional environments.

The semantics of first-order fixed point formulas is given by the interpretation function $[\![.]\!] : BForm \rightarrow \mathbf{BEnv} \rightarrow \mathbf{DEnv} \rightarrow \mathbf{Bool}$, defined inductively below. For a formula $\psi$, a boolean environment $\eta$, and a data environment $\varepsilon$ such that $fbv(\psi) \subseteq supp(\eta)$ and $fdv(\psi) \subseteq supp(\varepsilon)$, $[\![\psi]\!]\,\eta\varepsilon$ denotes the truth value of $\psi$ in the context of $\eta$ and $\varepsilon$:

$$\begin{aligned}
[\![b]\!]\,\eta\varepsilon &\stackrel{\text{def}}{=} [\![b]\!]\,\varepsilon \\
[\![Z(e)]\!]\,\eta\varepsilon &\stackrel{\text{def}}{=} (\eta(Z))([\![e]\!]\,\varepsilon) \\
[\![\psi_1 \wedge \psi_2]\!]\,\eta\varepsilon &\stackrel{\text{def}}{=} [\![\psi_1]\!]\,\eta\varepsilon \wedge [\![\psi_2]\!]\,\eta\varepsilon \\
[\![\exists z{:}D.\psi]\!]\,\eta\varepsilon &\stackrel{\text{def}}{=} \exists v \in D.\, [\![\psi]\!]\,\eta(\varepsilon[v/z]) \\
[\![(\mu Z(z{:}D).\psi)(e)]\!]\,\eta\varepsilon &\stackrel{\text{def}}{=} (\mu\Psi_{\eta\varepsilon})([\![e]\!]\,\varepsilon)
\end{aligned}$$

where the functional $\Psi_{\eta\varepsilon} : (D \rightarrow \mathbf{Bool}) \rightarrow (D \rightarrow \mathbf{Bool})$, associated to the formula $\mu Z(z{:}D).\psi$, is defined as $\Psi_{\eta\varepsilon} = \lambda G{:}D \rightarrow \mathbf{Bool}.\lambda v{:}D.\, [\![\psi]\!]\,(\eta[G/Z])(\varepsilon[v/z])$.

The functionals $\Psi_{\eta\varepsilon}$ associated to the first-order fixed point formulas being monotonic, and the underlying lattices $D \rightarrow \mathbf{Bool}$ being complete, it follows from Tarski's theorem that each functional $\Psi_{\eta\varepsilon}$ has a unique minimal fixed point $\mu\Psi_{\eta\varepsilon}$ and a unique maximal fixed point $\nu\Psi_{\eta\varepsilon}$.

### 4.2 Transformation of the Verification Problem

Consider the following linear $\mu$CRL process:

$$X(x{:}D) = \sum_{a \in Act} \sum_{x_a : E_a} a(e_a) \cdot X(e'_a) \lhd b_a \rhd \delta$$

As we precised in Section 2.3, the states of the corresponding transition system are identified with terms $X(v)$, where $v \in D$. We assume that the data variables used in the temporal formulas are disjoint from those used in the linear process.

According to the interpretation of state formulas, a state $X(v)$ satisfies a formula $\varphi$ in the context of a propositional environment $\rho$ and of a data environment $\varepsilon$ if and only if $X(v) \in [\![\varphi]\!]\,\rho\varepsilon$. As we will show, this is equivalent to the fact that a first-order fixed point formula $\text{TR}(\varphi)$ is true in the context of a boolean environment $\textsc{Tr}(\rho)$ and of $\varepsilon[v/x]$, where the translations $\textsc{Tr}(\varphi)$ and $\textsc{Tr}(\rho)$, which take the process $X$ as an implicit parameter, are defined below.

Given $\rho \in \mathbf{PEnv}$, the boolean environment $\textsc{Tr}(\rho)$, whose support is given by $supp(\textsc{Tr}(\rho)) = \{Z_Y(x{:}D, y{:}D') \mid Y(y{:}D') \in supp(\rho)\}$, is defined as follows:

$$\big(\textsc{Tr}(\rho)\big)(Z_Y) \stackrel{\text{def}}{=} \lambda v{:}D, v'{:}D'.\big(X(v) \in (\rho(Y))(v')\big)$$

for each $Z_Y \in supp(\textsc{Tr}(\rho))$.

Given $\varphi \in SForm$, the translation $\textsc{Tr}(\varphi)$ is defined inductively below:

$$\begin{aligned}
\textsc{Tr}(b) &\stackrel{\text{def}}{=} b\\
\textsc{Tr}(Y(e)) &\stackrel{\text{def}}{=} Z_Y(x,e)\\
\textsc{Tr}(\varphi_1 \wedge \varphi_2) &\stackrel{\text{def}}{=} \textsc{Tr}(\varphi_1) \wedge \textsc{Tr}(\varphi_2)\\
\textsc{Tr}(\langle\alpha\rangle\,\varphi) &\stackrel{\text{def}}{=} \textstyle\bigvee_{a\in Act} \exists x_a{:}E_a.(b_a \wedge (a(e_a) \models \alpha) \wedge \textsc{Tr}(\varphi)[e'_a/x])\\
\textsc{Tr}(\exists y{:}D'.\varphi) &\stackrel{\text{def}}{=} \exists y{:}D'.\textsc{Tr}(\varphi)\\
\textsc{Tr}((\mu Y(y{:}D').\varphi)(e)) &\stackrel{\text{def}}{=} (\mu Z_Y(x_Y{:}D, y{:}D').\textsc{Tr}(\varphi)[x_Y/x])(x,e)
\end{aligned}$$

where the predicate $a(e_a) \models \alpha$, expressing that an action $a(e_a)$ satisfies an action formula $\alpha \in AForm$, is defined inductively as follows:

$$\begin{aligned}
a(e_a) \models a'(e') &\stackrel{\text{def}}{=} a = a' \wedge e_a = e'\\
a(e_a) \models tt &\stackrel{\text{def}}{=} tt\\
a(e_a) \models \neg\alpha &\stackrel{\text{def}}{=} \neg(a(e_a) \models \alpha)\\
a(e_a) \models \alpha_1 \wedge \alpha_2 &\stackrel{\text{def}}{=} (a(e_a) \models \alpha_1) \wedge (a(e_a) \models \alpha_2)\\
a(e_a) \models \exists y{:}D.\alpha &\stackrel{\text{def}}{=} \exists y{:}D.(a(e_a) \models \alpha).
\end{aligned}$$

The following lemma states some auxiliary technical properties necessary for showing the correctness of the $\textsc{Tr}(\varphi)$ translation.

**Lemma 1.** *The following properties hold:*

1. *For all $a \in Act$, $e_a \in Exp$, $\alpha \in AForm$, and $\varepsilon \in$ **DEnv** such that $var(e_a) \cup fdv(\alpha) \subseteq supp(\varepsilon)$:*

$$[\![a(e_a) \models \alpha]\!]\,\varepsilon = \big(a([\![e_a]\!]\,\varepsilon) \in [\![\alpha]\!]\,\varepsilon\big).$$

2. *For all $a \in Act$ and $\varphi \in SForm$:*

$$fdv(\textsc{Tr}(\varphi)) \subseteq (fdv(\varphi) \cup \{x\}) \setminus \{x_a\}.$$

3. *For all $\psi \in BForm$, $e \in Exp$, $x \in DVar$, $\eta \in$ **BEnv**, and $\varepsilon \in$ **DEnv** such that $var(e) \cup fdv(\psi) \subseteq supp(\varepsilon)$:*

$$[\![\psi[e/x]]\!]\,\eta\varepsilon = [\![\psi]\!]\,\eta(\varepsilon[[\![e]\!]\,\varepsilon/x]).$$

*Proof.* Straightforward, by structural induction on $\alpha$ (property 1), on $\varphi$ (property 2), and on $\psi$ (property 3).

The following proposition expresses the relation between a linear process $X$, a state formula $\varphi$, and the corresponding first-order fixed point formula $\text{TR}(\varphi)$ obtained after translation.

**Proposition 1.** *Let $X(x{:}D)$ be a linear process as defined above and let $\varphi$ be a state formula. Then, for any $\rho \in$* **PEnv** *and $\varepsilon \in$* **DEnv** *such that $fpv(\varphi) \subseteq supp(\rho)$ and $fdv(\varphi) \subseteq supp(\varepsilon)$:*

$$\llbracket\varphi\rrbracket\,\rho\varepsilon = \{X(v) \in S \mid \llbracket\text{TR}(\varphi)\rrbracket\,\text{TR}(\rho)(\varepsilon[v/x])\}.$$

*Proof.* By structural induction on $\varphi$, using Lemma 1.

Using the result above, we can now restate the verification problem of a closed state formula $\varphi$ by a linear process $X$ in terms of the satisfaction of a first-order fixed point formula $\text{TR}(\varphi)$. The global model-checking problem, consisting to verify that the formula is satisfied by every state of the process, becomes:

$$\begin{array}{ll}
\forall v{:}D.(X(v) \in \llbracket\varphi\rrbracket\,[\,][\,]) \leftrightarrow & \text{by Proposition 1} \\
\forall v{:}D.\,\llbracket\text{TR}(\varphi)\rrbracket\,\text{TR}([\,])([\,][v/x]) \leftrightarrow & \text{by definition of } \text{TR}(\rho) \\
\forall v{:}D.\,\llbracket\text{TR}(\varphi)\rrbracket\,[\,][v/x] \leftrightarrow & \text{by definition of } \llbracket . \rrbracket\,\eta\varepsilon \\
\llbracket\forall x{:}D.\text{TR}(\varphi)\rrbracket\,[\,][\,]. &
\end{array}$$

(Note that we can use empty environments whenever the formulas are closed w.r.t. the corresponding variables.) The local model-checking problem, consisting to verify that the formula is satisfied by the initial state of the process, becomes:

$$\begin{array}{ll}
X(v_0) \in \llbracket\varphi\rrbracket\,[\,][\,] \leftrightarrow & \text{by Proposition 1} \\
\llbracket\text{TR}(\varphi)\rrbracket\,\text{TR}([\,])([\,][v_0/x]) \leftrightarrow & \text{by definition of } \text{TR}(\rho) \\
\llbracket\text{TR}(\varphi)\rrbracket\,[\,][v_0/x] \leftrightarrow & \text{by definition of } \llbracket . \rrbracket\,\eta\varepsilon \\
\llbracket\forall x{:}D.(x = v_0) \rightarrow \text{TR}(\varphi)\rrbracket\,[\,][\,]. &
\end{array}$$

Using the standard proof rules for first-order logic, together with the rules for minimal and maximal fixed point operators that will be given in Section 4.4, we have the basic tools available for proving the first-order fixed point formulas above.

### 4.3 Example (Continued)

We continue the example from Section 3.1 by giving the translations of the formulas $\varphi_1$, $\varphi_2$, and $\varphi_3$. So, to establish the validity of these formulas we must prove, respectively:

1. $(\mu Z(v{:}\mathrm{N}, b{:}\mathbf{Bool}).(b \wedge l \leq v) \vee (\neg b \wedge Z(v+1, \neg b)) \vee \exists m{:}\mathrm{N}.(b \wedge m \leq v \wedge Z(v - m, \neg b)))(v, b)$;

2. $(\mu Y(v{:}\mathrm{N}, b{:}\mathbf{Bool}).(\neg b \rightarrow Z(v+1, \neg b)) \wedge \forall m{:}\mathrm{N}.((b \wedge m \leq v \wedge m \neq l) \rightarrow Z(v-m, \neg b)))(v, b)$;
3. $(\nu Z(p, r, v{:}\mathrm{N}, b{:}\mathbf{Bool}).\forall l, m{:}\mathrm{N}.((b \wedge m \leq v \wedge m = l) \rightarrow (l \leq v_0 + p - r \wedge Z(p, r+l, v-m, \neg b))) \wedge (\neg b \rightarrow Z(p+1, r, v+1, \neg b)))(0, 0, v, b)$.

### 4.4 Proof Rules

As shown in Section 4.2, the verification of a data-based temporal logic formula on a linear $\mu$CRL process can be reduced to the satisfaction of a first-order formula containing fixed point operators. We provide here proof rules associated to the minimal and maximal fixed point operators. These rules can be naturally used in conjunction with some proof system for first-order logic (e.g., Gentzen's natural deduction system [6]) in order to prove the validity of first-order fixed point formulas.

We first define some auxiliary notations. Consider a fixed point formula $\sigma Z(z{:}D).\psi_1$ representing a predicate over $D$, and let $\psi_2 \in \mathit{BForm}$ such that $\mathit{fbv}(\psi_2) \subseteq \mathit{fbv}(\psi_1)$ and $\mathit{fdv}(\psi_2) \subseteq \mathit{fdv}(\psi_1)$. The *application* of $\psi_1$ on $\psi_2$ is defined as follows:

$$\psi_1[\psi_2] \stackrel{\mathrm{def}}{=} \psi_1[\psi_2[e/z]/Z(e)]$$

Intuitively, $\psi_1[\psi_2]$ is obtained by substituting all the occurrences of $Z(e)$ in $\psi_1$ by $\psi_2$, in which all occurrences of $z$ have been replaced with the actual parameter $e$. The conditions on the variables of $\psi_2$ ensure that no free variables of $\psi_2$ become bound in $\psi_1[\psi_2]$. For simplicity, whenever $\mathit{fdv}(\psi_2) = \{z\}$, we will write $\psi_2(e)$ for $\psi_2[e/z]$. We also assume that the domain N of natural numbers is predefined. For every $k \in \mathrm{N}$, the application $k$ times of $\psi_1$ on $\psi_2$, noted $\psi_1^k[\psi_2]$, is defined as follows:

$$\psi_1^0[\psi_2] \stackrel{\mathrm{def}}{=} \psi_2, \;\; \psi_1^{k+1}[\psi_2] \stackrel{\mathrm{def}}{=} \psi_1[\psi_1^k[\psi_2]]$$

Using these notations, the proof rules for minimal and maximal fixed point operators are given below:

$$\frac{\forall k \geq 0.(\psi_2(k) \rightarrow \psi_1^k[\mathit{ff}])}{(\exists k \geq 0.\psi_2(k)) \rightarrow (\mu Z(z{:}D).\psi_2)(z)}\ \textsc{LfpUp} \qquad \frac{\psi_1[\psi_2] \rightarrow \psi_2}{(\mu Z(z{:}D).\psi_1)(z) \rightarrow \psi_2}\ \textsc{LfpDn}$$

$$\frac{\forall k \geq 0.(\psi_1^k[\mathit{tt}] \rightarrow \psi_2(k))}{(\nu Z(z{:}D).\psi_1)(z) \rightarrow (\forall k \geq 0.\psi_2(k))}\ \textsc{GfpDn} \qquad \frac{\psi_2 \rightarrow \psi_1[\psi_2]}{\psi_2 \rightarrow (\nu Z(z{:}D).\psi_1)(z)}\ \textsc{GfpUp}$$

where $\psi_2(k)$ means that the variable $k$, denoting a natural number, occurs free in $\psi_2$. Intuitively, the rules LfpUp, GfpUp and LfpDn, GfpDn allow to approximate the extremal fixed points towards satisfaction and towards refutation, respectively. The following proposition states the soundness of these rules.

**Proposition 2.** *The rules* LfpUp, LfpDn, GfpUp, *and* GfpDn *defined above are sound w.r.t. the semantics of the first-order fixed point formulas* $\psi \in \mathit{BForm}$.

*Proof.* Given in [10].

### 4.5 Example (Continued)

We show the use of the rules given above by proving the formulas given in Section 4.3. We consider the three formulas separately. We give the proof of these formulas in extreme detail, such that every reasoning step can be understood.

1. For the first case we let $\psi_1 \stackrel{\text{def}}{=} (b \wedge l \leq v) \vee (\neg b \wedge Z(v+1, \neg b)) \vee \exists m{:}\text{N}.(b \wedge m \leq v \wedge Z(v-m, \neg b))$. In order to apply the rule LFPUP we must find some $\psi_2(k)$. We propose $\psi_2(k) \stackrel{\text{def}}{=} k > \mathit{if}(l \leq v, |\neg b|, 2(l-v) - |\neg b|)$. Here, $\mathit{if}(b, x, y)$ equals $x$ if $b$ holds and $y$ otherwise; $|b|$ equals 1 if $b$ holds and 0 otherwise. (Intuitively, $k$ denotes the minimal number of steps necessary to reach a $w(l)$ action, starting from any state of the system.) Note that the left hand side in the conclusion of LFPUP becomes $\exists k \geq 0.(k > \mathit{if}(l \leq v, |\neg b|, 2(l-v) - |\neg b|))$, which is a tautology. So, if we can prove the premises of LFPUP we have shown that the temporal formula $\varphi_1$ is valid in all states of $X(v, b)$.
The premise of LFPUP has become $\forall k \geq 0.(k > \mathit{if}(l \leq v, |\neg b|, 2(l-v) - |\neg b|) \rightarrow \psi_1^k[\mathit{ff}])$. We prove this premise by induction on $k$. For $k = 0$ this holds vacuously, because the left hand side of the implication equals falsum. For $k = k' + 1$, we must prove: $k' \geq \mathit{if}(l \leq v, |\neg b|, 2(l-v) - |\neg b|) \rightarrow (b \wedge l \leq v) \vee (\neg b \wedge \psi_1^{k'}[\mathit{ff}](v+1, \neg b)) \vee \exists m{:}\text{N}.(b \wedge m \leq v \wedge \psi_1^{k'}[\mathit{ff}](v-m, \neg b))$. This is done by making a few case distinctions:
   - Suppose $b$ holds and $l \leq v$. Clearly, the statement above is true, as the first disjunct of the right hand side trivially holds.
   - Now, suppose $b$ holds and $l > v$. We want to show that the third disjunct holds. As $b$ holds by assumption, it suffices to show that $\exists m{:}\text{N}.(m \leq v \wedge \psi_1^{k'}[\mathit{ff}](v-m, \neg b))$. Take $m = 0$. The proof obligation reduces to $\psi_1^{k'}[\mathit{ff}](v, \neg b)$. This is implied by the induction hypothesis, because $(\psi_2(k'))(v, \neg b) = k' > 2(l-v) - 1$, which is equivalent in this case to the left hand side $k' \geq 2(l-v)$ of the implication.
   - We still must consider the case where $\neg b$. We show that the second disjunct holds in this case. We must prove that $\psi_1^{k'}[\mathit{ff}](v+1, \neg b)$. The left hand side of the implication becomes $k' \geq \mathit{if}(l \leq v, 1, 2(l-v) - 1)$, which is easily seen (by distinguishing between the cases $l \leq v$, $l = v+1$, and $l > v+1$) to imply $(\psi_2(k'))(v+1, \neg b) = k' > \mathit{if}(l \leq v+1, 0, 2(l-v) - 2)$. So, the proof obligation follows from the inductive hypothesis.

   This finishes the proof of the first temporal formula.
2. We show that this formula does not hold in any state of $X$. Let $\psi_1$ be the body of the $\mu Z$ formula. We apply LFPDN, taking $\psi_2 \stackrel{\text{def}}{=} \mathit{ff}$. The left hand side $\psi_1[\psi_2]$ of the premise looks like $(\neg b \rightarrow \mathit{ff}) \wedge \forall m{:}\text{N}.((b \wedge m \leq v \wedge m \neq l) \rightarrow \mathit{ff})$, which is equivalent to $\mathit{ff}$. Thus, the fixed point formula is false for all $v \in \text{N}$ and $b \in \textbf{Bool}$.
3. We show that this formula is satisfied by the initial state of the system. Let $\psi_1$ be the body of the $\nu Z$ formula. We must prove that $(v = v_0 \wedge b = \mathit{ff}) \rightarrow (\nu Z(p, r, v{:}\text{N}, b{:}\textbf{Bool}).\psi_1)(0, 0, v, b)$ for all $v \in \text{N}$ and $b \in \textbf{Bool}$. We solve this by showing a slightly stronger property, namely that $(v = v_0 + p - r) \rightarrow (\nu Z(p, r, v{:}\text{N}, b{:}\textbf{Bool}).\psi_1)(p, r, v, b)$, which implies the above boolean

property by instantiating $v$, $b$, $p$, and $r$ with $v_0$, $\mathit{ff}$, 0, and 0, respectively. We apply GFPUP, taking $\psi_2 \overset{\mathrm{def}}{=} (v = v_0+p-r)$. The premise of GFPUP reduces to $(v = v_0+p-r) \rightarrow (\forall l, m{:}\mathrm{N}.((b \wedge m \leq v \wedge m = l) \rightarrow (l \leq v_0+p-r \wedge v-m = v_0+p-r-l)) \wedge (\neg b \rightarrow v+1 = v_0+p+1-r))$, which is easily seen to be a tautology. Hence, the initial state $X(v_0, \mathit{ff})$ satisfies $\varphi_3$.

## 5 Application

We present here a more involved verification example using the methodology described in Section 4. Consider the following linear process $Q(q)$ describing a queue $q$:

$$Q(q) = \textstyle\sum_{d:D} r(d) \cdot Q(in(d,q)) + s(toe(q)) \cdot Q(untoe(q)) \lhd |q| > 0 \rhd \delta$$

Data elements $d \in D$ are inserted in $Q$ via $r(d)$ actions and are delivered by $Q$ via $s(d)$ actions. The $|.|$ operator returns the number of elements in a queue. The *in* function inserts an element into a queue, the *untoe* function eliminates the element which was inserted first into a queue, and the *toe* function returns that element. We assume that the domain $D$ has at least one element. The concatenation of two queues $q_1$ and $q_2$ is described by the linear process below:

$$\begin{array}{rll} Q(q_1,q_2) = \sum_{d:D} r(d) \cdot Q(in(d,q_1),q_2) & \lhd \quad tt & \rhd \delta + \\ \tau \cdot Q(untoe(q_1), in(toe(q_1),q_2)) & \lhd |q_1| > 0 & \rhd \delta + \\ s(toe(q_2)) \cdot Q(q_1, untoe(q_2)) & \lhd |q_2| > 0 & \rhd \delta \end{array}$$

The initial state of this process is $Q(nil, nil)$, where $nil$ is a function returning an empty queue. In the following paragraphs we present the description and verification of several safety and liveness properties of the process $Q$.

*Property 1.* The essential safety property of the system is that every sequence of elements inserted in $Q$ will be delivered in the same order. This can be neatly expressed using a fixed point operator parameterized by a queue $q$ storing all the elements that have been inserted in $Q$ but not yet delivered:

$$\begin{array}{l} \varphi_1 \overset{\mathrm{def}}{=} (\nu Y(q).\forall d_0{:}D.\ [r(d_0)]\, Y(in(d_0,q)) \wedge \\ \qquad\qquad\qquad [s(d_0)]\, (|q| > 0 \wedge toe(q) = d_0 \wedge Y(untoe(q))) \wedge \\ \qquad\qquad [\neg\exists d_1{:}D.(s(d_1) \vee r(d_1))]\, Y(q) \\ \quad )(nil) \end{array}$$

This formula captures exactly the desired behaviour of the system: the two concatenated queues must behave as a single queue. (Note the presence of the quantifier in the action formula of the last box modality, in order to express that an action is different from any $s(\ldots)$ or $r(\ldots)$ action.) We verify $\varphi_1$ in the initial state $Q(nil, nil)$ of the system. This translates as follows:

$$\begin{array}{l}\forall q_1, q_2.(q_1 = nil \wedge q_2 = nil) \rightarrow \\ \quad (\nu Z(q_1, q_2, q).\forall d_0{:}D.\forall d{:}D.(d_0 = d \rightarrow Z(in(d, q_1), q_2, in(d_0, q))) \wedge \\ \qquad\qquad ((|q_2| > 0 \wedge d_0 = toe(q_2)) \rightarrow (|q| > 0 \wedge d_0 = toe(q) \wedge \\ \qquad\qquad\qquad\qquad Z(q_1, untoe(q_2), untoe(q)))) \wedge \\ \qquad\qquad (|q_1| > 0 \rightarrow Z(untoe(q_1), in(toe(q_1), q_2), q)) \\ \quad )(q_1, q_2, nil)\end{array}$$

Let $\psi_1$ be the body of the $\nu Z$ formula. To show the first-order fixed point formula above, we prove a slightly stronger property, namely that $(q_1 + q_2 = q) \rightarrow (\nu Z(q_1, q_2, q).\psi_1)(q_1, q_2, q)$ for all $q_1$, $q_2$, and $q$, where $q_1 + q_2$ denotes the concatenation of $q_1$ and $q_2$. We use the rule GFPUP, taking $\psi_2 \overset{\text{def}}{=} (q_1 + q_2 = q)$. The premise $\psi_2 \rightarrow \psi_1[\psi_2]$ of GFPUP reduces to the following three implications:

1. $\forall d_0, d{:}D.(q_1 + q_2 = q \wedge d_0 = d) \rightarrow (in(d, q_1) + q_2 = in(d_0, q))$;
2. $\forall d_0{:}D.(q_1 + q_2 = q \wedge |q_2| > 0 \wedge d_0 = toe(q_2)) \rightarrow (|q| > 0 \wedge d_0 = toe(q) \wedge q_1 + untoe(q_2) = untoe(q))$;
3. $\forall d_0{:}D.(q_1 + q_2 = q \wedge |q_1| > 0) \rightarrow (untoe(q_1) + in(toe(q_1), q_2) = q)$.

These properties can be easily shown using an appropriate axiomatization of the queue operators. Now, by instantiating $q$ to *nil*, and since $(q_1 = nil \wedge q_2 = nil) \rightarrow (q_1 + q_2 = nil)$, this implies that $(q_1 = nil \wedge q_2 = nil) \rightarrow (\nu Z(q_1, q_2, q).\psi_1)(q_1, q_2, nil)$ for all $q_1$ and $q_2$. Hence, $Q(nil, nil)$ satisfies $\varphi_1$.

*Property 2.* A simple liveness property (which also implies deadlock freedom) is that every datum $d_0 \in D$ can be potentially inserted in $Q$ by an action $r(d_0)$:

$$\varphi_2 \overset{\text{def}}{=} \mu Y.\, \langle r(d_0)\rangle\, tt \vee \langle tt\rangle\, Y$$

The verification of $\varphi_2$ in all the states of $Q$ translates as follows:

$$\begin{array}{l}\forall q_1, q_2.(\mu Z(q_1, q_2).\exists d{:}D.(d = d_0) \vee \exists d{:}D.Z(in(d, q_1), q_2) \vee \\ \qquad\qquad (|q_1| > 0 \wedge Z(untoe(q_1), in(toe(q_1), q_2))) \vee \\ \qquad\qquad (|q_2| > 0 \wedge Z(q_1, untoe(q_2))) \\ \quad )(q_1, q_2)\end{array}$$

We write $\psi_1$ for the body of the $\mu Z$ formula. Since the disjunct $\exists d{:}D.(d = d_0)$ is trivially true, $\psi_1$ reduces to *tt* and, by applying the rule LFPUP with $\psi_2(k) = tt$, it follows that $(\mu Z(q_1, q_2).\psi_1)(q_1, q_2)$ is valid for all values of $q_1$ and $q_2$. Hence, $\varphi_2$ holds in all states of $Q$.

*Property 3.* A more involved liveness property is that every datum $d_0$ which is inserted in $Q$ by an action $r(d_0)$ will be eventually delivered by an action $s(d_0)$:

$$\varphi_3 \overset{\text{def}}{=} [r(d_0)]\, \mu Y.\, \langle tt\rangle\, tt \wedge [\neg s(d_0)]\, Y$$

The verification of $\varphi_3$ in all the states of $Q$ translates as follows:

$$
\begin{array}{l}
\forall q_1, q_2 . \forall d{:}D . d = d_0 \rightarrow \\
\quad (\mu Z(q_1, q_2) . \forall d{:}D . Z(in(d, q_1), q_2) \wedge \\
\qquad\qquad (|q_1| > 0 \rightarrow Z(untoe(q_1), in(toe(q_1), q_2))) \wedge \\
\qquad\qquad ((|q_2| > 0 \wedge toe(q_2) \neq d_0) \rightarrow Z(q_1, untoe(q_2))) \\
\quad )(in(d, q_1), q_2)
\end{array}
$$

Let $\psi_1$ be the body of the $\mu Z$ formula. Observing that $\psi_1[\mathit{ff}] = \mathit{ff}$, the rule LFPDN leads to $(\mu Z(q_1, q_2).\psi_1)(q_1, q_2) \rightarrow \mathit{ff}$ for every $q_1$ and $q_2$. Then, the whole first-order fixed point formula reduces to $\forall d{:}D.d \neq d_0$, which is obviously false. Hence, $\varphi_3$ does not hold in any state of $Q$. This happens because one can always insert data elements into $Q$ (see formula $\varphi_2$ above) and, under an unfair scheduling of actions (but see next paragraph), the process may never deliver an element, letting $q_1$ and $q_2$ grow unboundedly.

*Property 4.* We may express the formula $\varphi_3$ by taking into account only the execution paths that are *fair* w.r.t. the action $s(d_0)$, i.e., those paths which cannot infinitely often enable $s(d_0)$ without infinitely often executing it:

$$\varphi_4 \overset{\text{def}}{=} [r(d_0)]\, \nu Y_1 .\, [\neg s(d_0)]\, Y_1 \wedge \mu Y_2 .\, \langle s(d_0) \rangle\, tt \vee \langle tt \rangle\, Y_2$$

The formula $\varphi_4$ specifies that after $d_0$ has been inserted in $Q$, as long as it has not yet been delivered, it is still possible to deliver it. This is an action-based instance of the fairness operator proposed in [18], where it was shown that it expresses the reachability on fair paths.

The verification of $\varphi_4$ in all the states of $Q$ translates as follows:

$$
\begin{array}{l}
\forall q_1, q_2 \,.\, (\nu Z_1(q_1, q_2) . \forall d{:}D . Z_1(in(d, q_1), q_2) \wedge \\
\qquad\quad (|q_1| > 0 \rightarrow Z_1(untoe(q_1), in(toe(q_1), q_2))) \wedge \\
\qquad\quad ((|q_2| > 0 \wedge toe(q_2) \neq d_0) \rightarrow Z_1(q_1, untoe(q_2))) \wedge \\
\qquad\quad (\mu Z_2(q_1, q_2) . (|q_2| > 0 \wedge toe(q_2) = d_0) \vee \exists d{:}D . Z_2(in(d, q_1), q_2) \vee \\
\qquad\qquad\qquad (|q_1| > 0 \wedge Z_2(untoe(q_1), in(toe(q_1), q_2))) \vee \\
\qquad\qquad\qquad (|q_2| > 0 \wedge Z_2(q_1, untoe(q_2))) \\
\qquad\quad )(q_1, q_2) \\
\quad )(in(d_0, q_1), q_2)
\end{array}
$$

Let $\psi_1$ be the body of the $\nu Z_1$ formula. We show the first-order fixed point formula above by proving a slightly stronger property, namely that $d_0 \in q_1 + q_2 \rightarrow (\nu Z_1(q_1, q_2).\psi_1)(q_1, q_2)$ for all $q_1$ and $q_2$, where $\in$ denotes the membership of an element in a queue. (Having shown this, the validity of the first-order fixed point formula above follows by instantiating $q_1$ with $in(d_0, q_1)$, since $d_0 \in in(d_0, q_1) + q_2$ is trivially true.) We apply the rule GFPUP on $\psi_1$, taking $\psi_1' \overset{\text{def}}{=} d_0 \in q_1 + q_2$. The premise $\psi_1[\psi_1']$ reduces to the following four implications:

1. $(d_0 \in q_1 + q_2) \rightarrow (\forall d{:}D.d_0 \in \mathit{in}(d, q_1) + q_2)$;
2. $(d_0 \in q_1 + q_2 \wedge |q_1| > 0) \rightarrow (d_0 \in \mathit{untoe}(q_1) + \mathit{in}(\mathit{toe}(q_1), q_2))$;
3. $(d_0 \in q_1 + q_2 \wedge |q_2| > 0 \wedge \mathit{toe}(q_2) \neq d_0) \rightarrow (d_0 \in q_1 + \mathit{untoe}(q_2))$;
4. $(d_0 \in q_1 + q_2) \rightarrow (\mu Z_2(q_1, q_2).\psi_2)(q_1, q_2)$

where $\psi_2$ is the body of the $\mu Z_2$ subformula. The first three properties follow easily from an axiomatization of the queue type. We show the last property using the rule LFPUP, by taking $\psi_2'(k) \overset{\mathrm{def}}{=} d_0 \in q_1 + q_2 \wedge 2|q_1| + |q_2| \leq k$ (intuitively, $k$ denotes the minimal number of steps in which an element $d_0$ already present in $Q$ can be delivered). Note that the left hand side in the conclusion of LFPUP becomes $\exists k \geq 0.(d_0 \in q_1 + q_2 \wedge 2|q_1| + |q_2| \leq k)$, which is trivially equivalent to $d_0 \in q_1 + q_2$.

We show the premise $\forall k \geq 0.(\psi_2'(k) \rightarrow \psi_2^k[\mathit{ff}])$ of LFPUP by induction on $k$. For $k = 0$ this holds vacuously, because $\psi_2'(0)$ is false. For $k = k' + 1$, we must prove that $(d_0 \in q_1 + q_2 \wedge 2|q_1| + |q_2| \leq k' + 1) \rightarrow \psi_2^{k'+1}[\mathit{ff}]$. We distinguish two cases:

- $|q_1| > 0$. We show that the left hand side of the implication above implies the disjunct $|q_1| > 0 \wedge \psi_2^{k'}[\mathit{ff}](\mathit{untoe}(q_1), \mathit{in}(\mathit{toe}(q_1), q_2))$ of $\psi_2^{k'+1}[\mathit{ff}]$. The first conjunct is true by assumption. The second conjunct is implied by the inductive hypothesis, because: (a) $d_0 \in q_1 + q_2 \rightarrow d_0 \in \mathit{untoe}(q_1) + \mathit{in}(\mathit{toe}(q_1), q_2)$, and (b) $2|\mathit{untoe}(q_1)| + |\mathit{in}(\mathit{toe}(q_1), q_2)| = 2|q_1| + |q_2| - 1 \leq k'$.
- $|q_1| = 0$. This implies that $|q_2| > 0$, because $d_0 \in q_1 + q_2$ by hypothesis. If $\mathit{toe}(q_2) = d_0$, then the disjunct $|q_2| > 0 \wedge \mathit{toe}(q_2) = d_0$ of $\psi_2^{k'+1}[\mathit{ff}]$ is true. If $\mathit{toe}(q_2) \neq d_0$, the disjunct $|q_2| > 0 \wedge \psi_2^{k'}[\mathit{ff}](q_1, \mathit{untoe}(q_2))$ of $\psi_2^{k'+1}[\mathit{ff}]$ follows from the inductive hypothesis, because: (a) $d_0 \in q_1 + \mathit{untoe}(q_2)$, and (b) $2|q_1| + |\mathit{untoe}(q_2)| = 2|q_1| + |q_2| - 1 \leq k'$.

This concludes the proof that all the states of $Q$ satisfy $\varphi_4$.

## Acknowledgements

We are grateful to Yaroslav Usenko, Anubhav Gupta, and to the anonymous referees for their careful reading and judicious comments on this paper.

## References

1. Andersen, H.R.: Model Checking and Boolean Graphs. *Theoretical Computer Science*, 126(1):3–30, 1994.
2. Arnold, A., Crubillé, P.: A Linear Algorithm to Solve Fixed-Point Equations on Transition Systems. *Information Processing Letters*, 29:57–66, 1988.
3. Bezem, M.A., Groote, J.F.: Invariants in Process Algebra with Data. In: Jonsson, B., Parrow, J. (eds.): *Proceedings of CONCUR'94 (Uppsala, Sweden)*, LNCS 836, pp. 401–416, Springer Verlag, 1994.
4. Bezem, M.A., Groote, J.F.: A Correctness Proof of a One Bit Sliding Window Protocol in $\mu$CRL. *The Computer Journal*, 37(4):289–307, 1994.

5. Bosscher, D., Ponse, A.: Translating a Process Algebra with Symbolic Data Values to Linear Format. In: Engberg, U.H., Larsen, K.G., Skou, A. (eds.), *Proceedings of TACAS'95 (Aarhus, Denmark)*, BRICS Notes Series, pp. 119–130, University of Aarhus, 1995.
6. van Dalen, D.: Logic and Structure. Springer Verlag, 1994.
7. Dam, M.: Model Checking Mobile Processes. *Information and Computation*, 129:35–51, 1996.
8. Fredlund, L.-Å., Groote, J.F., Korver, H.: Formal Verification of a Leader Election Protocol in Process Algebra. *Theoretical Computer Science*, 177:459–486, 1997.
9. Groote, J.F.: A Note on $n$ Similar Parallel Processes. In: Gnesi, S., Latella, D. (eds.), *Proceedings of the 2nd ERCIM Int. Workshop on Formal Methods for Industrial Critical Systems (Cesena, Italy)*, pp. 65–75, 1997. (See also Report CS-R9626, CWI, Amsterdam, 1996).
10. Groote, J.F., Mateescu, R.: Verification of Temporal Properties of Processes in a Setting with Data. Technical Report SEN-R9804, CWI, Amsterdam, 1998.
11. Groote, J.F., Monin, F., Springintveld, J.: A Computer Checked Algebraic Verification of a Distributed Summing Protocol. Computer Science Report 97/14, Dept. of Math. and Comp. Sci., Eindhoven University of Technology, 1997.
12. Groote, J.F., van de Pol, J.C.: A Bounded Retransmission Protocol for Large Data Packets. A Case Study in Computer Checked Verification. In: Wirsing, M., Nivat, M. (eds.), *Proceedings of AMAST'96 (Munich, Germany)*, LNCS 1101, pp. 536–550, Springer Verlag, 1996.
13. Groote, J.F., Ponse, A.: The Syntax and Semantics of $\mu$CRL. In: Ponse, A., Verhoef, C., van Vlijmen, S.F.M. (eds.), *Algebra of Communicating Processes*, Workshops in Computing, pp. 26–62, 1994.
14. Groote, J.F., Springintveld, J.: Focus Points and Convergent Process Operators. A Proof Strategy for Protocol Verification. Technical Report 142, Logic Group Preprint Series, Utrecht University, 1995. (See also Technical Report CS-R9566, CWI, Amsterdam, 1995).
15. Kindler, A., Reisig, W., Völzer, H., Walter, R.: Petri Net Based Verification of Distributed Algorithms: an Example. *Formal Aspects of Computing*, 9:409–424, 1997.
16. Kozen, D.: Results on the Propositional $\mu$-calculus. *Theoretical Computer Science* 27, pp. 333–354, 1983.
17. Manna, Z., Pnueli, A.: Adequate Proof Principles for Invariance and Liveness Properties of Concurrent Programs. *Science of Computer Programming* 32:257–289, 1984.
18. Queille, J-P., Sifakis, J.: Fairness and Related Properties in Transition Systems — a Temporal Logic to Deal with Fairness. *Acta Informatica*, 19:195–220, 1983.
19. Rathke, J., Hennessy, M.: Local Model Checking for a Value-Based Modal $\mu$-calculus. Technical Report 5/96, School of Cognitive and Computing Sciences, University of Sussex, 1996.
20. Shankland, C.: The Tree Identify Protocol of IEEE 1394. In: Groote, J.F., Luttik, B., van Wamel, J. (eds.), *Proceedings of the 3rd ERCIM Int. Workshop on Formal Methods for Industrial Critical Systems (Amsterdam, The Netherlands)*, pp. 299–319, 1998.
21. Tarski, A.: A Lattice-Theoretical Fixpoint Theorem and its Applications. *Pacific Journal of Mathematics* 5, pp. 285–309, 1955.
22. Vergauwen, B., Lewi, J.: A Linear Algorithm for Solving Fixed-Point Equations on Transition Systems. *Proceedings of CAAP'92 (Rennes, France)*, LNCS 581, pp. 322–341, Springer Verlag, 1992.