



Centrum voor Wiskunde en Informatica

REPORTRAPPORT

Verification of Temporal Properties of Processes in a Setting
with Data

J.F. Groote, R. Mateescu

Software Engineering (SEN)

SEN-R9804 May 31, 1998

Report SEN-R9804
ISSN 1386-369X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

CWI is the National Research Institute for Mathematics and Computer Science. CWI is part of the Stichting Mathematisch Centrum (SMC), the Dutch foundation for promotion of mathematics and computer science and their applications.

SMC is sponsored by the Netherlands Organization for Scientific Research (NWO). CWI is a member of ERCIM, the European Research Consortium for Informatics and Mathematics.

Copyright © Stichting Mathematisch Centrum
P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

Verification of Temporal Properties of Processes in a Setting with Data

J.F. Groote[†] and R. Mateescu^{*}

CWI^{†*}

P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Department of Mathematics and Computing Science, Eindhoven University of Technology[†]

P.O. Box 513, 5600 MB Eindhoven, The Netherlands

E-mail: jfg@cwi.nl, mateescu@cwi.nl

ABSTRACT

We define a value-based modal μ -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 μ CRL terms defined by linear process equations. The satisfaction of a temporal formula by a μ 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.

1991 Mathematics Subject Classification: 68M99, 68N99

1991 Computing Reviews Classification System: D.2.1, D.2.4, D.3.1

Keywords and Phrases: Mu-Calculus, Process Algebra, Specification, Temporal Logic, Verification

1 Introduction

In recent years we have applied process algebra in numerous settings [4, 13, 17]. 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 μ CRL (*micro Common Representation Language*) [18]. 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, 15] 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 [19] and has been applied to numerous examples (see e.g. [4, 13, 16, 26]) 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 μ -calculus [21] to express properties about data. This means that the actions in diamond and box modalities may contain data parameters, boolean expressions on data may be included, quantification over data is allowed, and minimal and maximal fixed points can also be parameterized with data. A typical example of temporal property expressed in this logic is

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

describing the states from which an infinite sequence of actions $a(i_0) a(i_1) a(i_2) \cdots$ can be performed,

where $2 \leq i_0 \leq i_1 \leq i_2 \leq \dots$. Another example of formula is

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

which says 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 to a first order boolean formula containing fixed point operators. This approach is similar to the model-checking algorithms in [2, 28, 1], where a formula of standard μ -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 that 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 boolean formulas that we obtain in this way, we introduce a set of proof rules for the fixed point operators (the rules for connectives and quantifiers are well known). These proof rules have been devised such that they allow to approximate the fixed point (sub)formulas. The rules are symmetric, enabling either to prove, or to refute the fixed point formulas. Having such approximation rules available is important for proving correctness of systems. For instance, determining exactly the set of reachable states in a distributed system is often extremely hard. Therefore, this set of states is generally approximated via a sequence of invariants. Now, if we can approximate a maximal fixed point and then show that such an approximation is valid in the initial state, we know that the maximal fixed point holds in the initial state too. The approximation of minimal fixed points is slightly more complex, capturing the fact that the property expressed by a minimal fixed point formula will be reached in a finite number of steps. These approximation rules reflect, in a way, the proof principles for safety and liveness properties discussed in [22].

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 [16], 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 [25, 9] 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 [20]), and therefore it is interesting to see how the methods compare. We have not yet investigated this in depth, but we already observed that the examples treated in [25, 9] appear much smaller than for instance the distributed summing protocol.

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

2 Preliminaries

We define below the notions of data expression, linear process, and labelled 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 $D_1 \times \dots \times D_n \rightarrow D$,

where D_1, \dots, D_n are the argument types of f and D is its result type. We write \mathbf{Val} for the domain containing all the values belonging to the types D . The expressions $e \in \mathit{Exp}$ are defined by the following grammar:

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

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

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 ε , noted $\mathit{supp}(\varepsilon)$, denotes the set of variables that are assigned a value in \mathbf{Val} by ε . An environment mapping the variables x_1, \dots, x_n respectively to the values v_1, \dots, v_n is noted $[v_1/x_1, \dots, v_n/x_n]$. The environment having an empty support is noted $[\]$. The *overriding* of ε by $[v_1/x_1, \dots, v_n/x_n]$ is the data environment defined as follows: $(\varepsilon[v_1/x_1, \dots, v_n/x_n])(x) = \text{if } \exists i \in [1, n]. x = x_i \text{ then } v_i \text{ else } \varepsilon(x)$.

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

$$\begin{aligned} \llbracket x \rrbracket \varepsilon &\stackrel{\text{def}}{=} \varepsilon(x) \\ \llbracket f(e_1, \dots, e_n) \rrbracket \varepsilon &\stackrel{\text{def}}{=} f(\llbracket e_1 \rrbracket \varepsilon, \dots, \llbracket e_n \rrbracket \varepsilon) \end{aligned}$$

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 the advantage with LTSS that it is 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 which is 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 \mathbf{Act} be a set of actions, which may be parameterized by data values.

Definition 2.1. Let $\mathit{Act} \subseteq \mathbf{Act} \cup \{\tau\}$ be a finite set of actions and D and D_a be data types. A *linear process* over Act and D is defined by an equation of the following form:

$$X(x:D) = \sum_{a \in \mathit{Act}} \sum_{x_a : D_a} a(e_a) \cdot X(e'_a) \triangleleft b_a \triangleright \delta$$

where x is a parameter of type D , and for each action $a \in \mathit{Act}$, x_a is a variable of type D_a , e_a and e'_a are expressions of type D_a and D , respectively, and b_a is an expression of type \mathbf{Bool} , such that $\mathit{var}(e_a) \cup \mathit{var}(e'_a) \cup \mathit{var}(b_a) \subseteq \{x, x_a\}$. The *initial state* of process X may be specified by giving an initial value $v_0 \in D$ for x . \square

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 D_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 $a \in \mathit{Act}$ (we assume that τ 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 μCRL process X as in Definition 2.1. According to the operational semantics of μCRL [18], the transition system modelling a linear process is defined as follows.

Definition 2.2. The *transition system* of a linear process is a quadruple $M = (S, L, \rightarrow, s_0)$, where:

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

3 Temporal logic

The temporal logic we consider is based upon an extension of the modal μ -calculus [21] with data variables, quantifiers and parameterization, in order to express properties involving data. The choice of the μ -calculus as basic formalism is motivated by its expressiveness, since it allows straightforward encodings (see [12]) of dynamic logics as PDL [14], branching time temporal logics as CTL [6], as well as (more involved) translations (see [8]) of temporal logics containing linear time modalities, like PTL [23] or CTL* [11]. Other similar value-based formalisms extending the modal μ -calculus have been used in the framework of symbolic transition systems [25] and of the polyadic π -calculus [9].

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.

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

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

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

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

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

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

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

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 . The derived boolean operators and the \forall quantifier are defined as usual. The box modal operator and the maximal fixed point operator are defined respectively as $[\alpha]\varphi = \neg \langle \alpha \rangle \neg\varphi$ and $(\nu Y(y:D).\varphi)(e) = \neg(\mu Y(y:D).\neg\varphi[\neg Y/Y])(e)$, where $\varphi[\neg Y/Y]$ denotes the syntactic substitution of Y by $\neg Y$ in φ . In the sequel, we let σ 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 φ are noted $fdv(\varphi)$ and $fpv(\varphi)$, respectively. A formula φ is said *closed* if $fdv(\varphi) = \emptyset$ and $fpv(\varphi) = \emptyset$.

We assume that state formulas are *syntactically monotonic*, i.e., for each fixed point formula $(\sigma Y(y:D).\varphi)(e)$, every free occurrence of Y in φ falls under an even number of negations. Any syntactically monotonic formula φ can be converted in *Positive Normal Form* (PNF for short) by pushing the negations down to its atomic subformulas and (if necessary) by α -converting it such that there is no variable Y having both free and bound occurrences in φ . A closed formula in PNF does not contain any negated occurrences of propositional variables. In the sequel, we consider only state formulas in PNF.

We define the domain $\mathbf{PEnv} = PVar \rightarrow (\mathbf{Val} \rightarrow 2^S)$ of *propositional environments*. A propositional environment $\rho \in \mathbf{PEnv}$ is a partial function mapping propositional variables to functions from the data domains of the parameters to sets of states of the transition system. 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 $\llbracket \cdot \rrbracket : SForm \rightarrow \mathbf{PEnv} \rightarrow \mathbf{DEnv} \rightarrow 2^S$, defined inductively below. For a state formula φ , a propositional environment ρ , and a data environment ε such that $fpv(\varphi) \subseteq \text{supp}(\rho)$ and $fdv(\varphi) \subseteq \text{supp}(\varepsilon)$, $\llbracket \varphi \rrbracket \rho \varepsilon$ denotes the set of states satisfying φ in the context of ρ and ε :

$$\begin{aligned} \llbracket b \rrbracket \rho \varepsilon &\stackrel{\text{def}}{=} \begin{cases} S & \text{if } \llbracket b \rrbracket \varepsilon \\ \emptyset & \text{otherwise} \end{cases} \\ \llbracket Y(e) \rrbracket \rho \varepsilon &\stackrel{\text{def}}{=} (\rho(Y))(\llbracket e \rrbracket \varepsilon) \\ \llbracket \varphi_1 \vee \varphi_2 \rrbracket \rho \varepsilon &\stackrel{\text{def}}{=} \llbracket \varphi_1 \rrbracket \rho \varepsilon \cup \llbracket \varphi_2 \rrbracket \rho \varepsilon \\ \llbracket \varphi_1 \wedge \varphi_2 \rrbracket \rho \varepsilon &\stackrel{\text{def}}{=} \llbracket \varphi_1 \rrbracket \rho \varepsilon \cap \llbracket \varphi_2 \rrbracket \rho \varepsilon \\ \llbracket \langle \alpha \rangle \varphi \rrbracket \rho \varepsilon &\stackrel{\text{def}}{=} \{X(v) \in S \mid \exists v' \in D. \exists a \in Act. \exists v_a \in D_a. \\ &\quad 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 [\alpha] \varphi \rrbracket \rho \varepsilon &\stackrel{\text{def}}{=} \{X(v) \in S \mid \forall v' \in D. \forall a \in Act. \forall v_a \in D_a. \\ &\quad (X(v) \xrightarrow{a(v_a)} X(v') \wedge a(v_a) \in \llbracket \alpha \rrbracket \varepsilon) \rightarrow X(v') \in \llbracket \varphi \rrbracket \rho \varepsilon\} \\ \llbracket \exists y:D'. \varphi \rrbracket \rho \varepsilon &\stackrel{\text{def}}{=} \{X(v) \in S \mid \exists v' \in D'. X(v) \in \llbracket \varphi \rrbracket \rho(\varepsilon[v'/y])\} \\ \llbracket \forall y:D'. \varphi \rrbracket \rho \varepsilon &\stackrel{\text{def}}{=} \{X(v) \in S \mid \forall v' \in D'. X(v) \in \llbracket \varphi \rrbracket \rho(\varepsilon[v'/y])\} \\ \llbracket (\mu Y(y:D'). \varphi)(e) \rrbracket \rho \varepsilon &\stackrel{\text{def}}{=} (\mu \Phi_{\rho \varepsilon})(\llbracket e \rrbracket \varepsilon) \\ \llbracket (\nu Y(y:D'). \varphi)(e) \rrbracket \rho \varepsilon &\stackrel{\text{def}}{=} (\nu \Phi_{\rho \varepsilon})(\llbracket e \rrbracket \varepsilon) \end{aligned}$$

where $\Phi_{\rho \varepsilon} : (D' \rightarrow 2^S) \rightarrow (D' \rightarrow 2^S)$, $\Phi_{\rho \varepsilon} \stackrel{\text{def}}{=} \lambda F:D' \rightarrow 2^S. \lambda v':D'. \llbracket \varphi \rrbracket (\rho[F/Y])(\varepsilon[v'/y])$.

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 $D' \rightarrow 2^S$. Since the underlying lattices $D' \rightarrow 2^S$ are complete, it follows from Tarski's theorem [27] 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 quantified boolean fixed point formulas that we have obtained this way. The example is given by the following linear process equation, describing a slot machine:

$$X(v:\mathbb{N}, b:\mathbf{Bool}) = \sum_{m:\mathbb{N}} w(m) \cdot X(v-m, \neg b) \triangleleft b \wedge m \leq v \triangleright \delta + s \cdot X(v+1, \neg b) \triangleleft \neg b \triangleright \delta$$

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 $X(v_0, ff)$, 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 the example we do not complicate the slot machine description 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 $l \in \mathbb{N}$, the machine 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 $l \in \mathbb{N}$, the machine must eventually deliver it:

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

3. A safety property is that every $l \in \mathbb{N}$ won in a $w(l)$ action cannot exceed the initial amount of money v_0 of the machine, updated with the p and r money that have been inserted and won by users since the initial state of the system, respectively:

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

Clearly, φ_1 and φ_3 are valid for X , but φ_2 does not hold.

4 Verification

The verification problem consists to check whether a transition system M (given by a linear μCRL process) satisfies a given temporal formula φ . Two different cases are usually distinguished: *global* verification, consisting to decide if all the states of M satisfy φ , and *local* verification, consisting to decide if one particular state (e.g., the initial state s_0) of M satisfies φ . Both instances of the problem can be reduced to the satisfaction of a boolean formula expressed in a first-order logic extended with fixed point operators. We first define the language of boolean formulas, next we describe the translation of a model M and a formula φ into a boolean formula, and finally we provide sound proof rules allowing to reason about the fixed point operators.

4.1 Boolean formulas

We define the syntax and semantics of the set $BForm$ of *boolean formulas*, which will be used as an intermediate formalism for verification purposes. The boolean formulas $\psi \in BForm$, built over a set $BVar$ of *boolean variables*, are defined by the following grammar (given directly in positive form):

$$\psi ::= b \mid Z(e) \mid \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2 \mid \exists z:D.\psi_1 \mid \forall z:D.\psi_1 \mid (\mu Z(z:D).\psi_1)(e) \mid (\nu Z(z:D).\psi_1)(e)$$

where $b \in Exp$ is a boolean expression and $Z \in BVar$ is a (parameterized) boolean variable. The data and boolean variables are bound in a manner similar to the state formulas φ . The sets of free data variables and free boolean variables occurring in ψ are noted $fdv(\psi)$ and $fbv(\psi)$, respectively. As for the state formulas, we use only one data parameter for the boolean fixed point formulas; using pairing and projection, the formalizations could be easily extended to multiple parameters.

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 boolean formulas is given by the interpretation function $\llbracket \cdot \rrbracket : BForm \rightarrow \mathbf{BEnv} \rightarrow \mathbf{DEnv} \rightarrow \mathbf{Bool}$, defined inductively below. For a boolean formula ψ , a boolean environment η , and a data environment ε such that $fbv(\psi) \subseteq \text{supp}(\eta)$ and $fdv(\psi) \subseteq \text{supp}(\varepsilon)$, $\llbracket \psi \rrbracket \eta \varepsilon$ denotes the truth value of ψ in the context of η and ε :

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

where $\Psi_{\eta \varepsilon} : (D \rightarrow \mathbf{Bool}) \rightarrow (D \rightarrow \mathbf{Bool})$, $\Psi_{\eta \varepsilon} \stackrel{\text{def}}{=} \lambda G : D \rightarrow \mathbf{Bool}. \lambda v : D. \llbracket \psi \rrbracket (\eta[G/Z])(\varepsilon[v/z])$.

The functionals $\Psi_{\eta \varepsilon}$ associated to the fixed point boolean 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 Translation of the verification problem into boolean formulas

Consider the following linear μCRL process:

$$X(x:D) = \sum_{a \in Act} \sum_{x_a : D_a} a(e_a) \cdot X(e'_a) \triangleleft b_a \triangleright \delta$$

As we precised in Section 2.3, the states of the corresponding transition system are identified with terms of the form $X(v)$, 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 φ in the context of a propositional environment ρ and of a data environment ε if and only if $X(v) \in \llbracket \varphi \rrbracket \rho \varepsilon$. As we will show, this is equivalent to the fact that a boolean formula $\text{Tr}(\varphi)$ is true in the context of a boolean

environment $\text{TR}(\rho)$ and of $\varepsilon[v/x]$, where the translations $\text{TR}(\varphi)$ and $\text{TR}(\rho)$, which take the process X as an implicit parameter, are defined below.

Given $\rho \in \mathbf{PEnv}$, the boolean environment $\text{TR}(\rho)$, whose support is $\text{supp}(\text{TR}(\rho)) \stackrel{\text{def}}{=} \{Z_Y(x:D, y:D') \mid Y(y:D') \in \text{supp}(\rho)\}$, is defined as follows:

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

for each $Z_Y \in \text{supp}(\text{TR}(\rho))$.

The translation $\text{TR}(\varphi)$ is defined inductively below:

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

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 $\text{TR}(\varphi)$ translation.

Lemma 4.1. *The following properties hold:*

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

$$\llbracket a(e_a) \models \alpha \rrbracket \varepsilon = (a(\llbracket e_a \rrbracket \varepsilon) \in \llbracket \alpha \rrbracket \varepsilon).$$

2. For all $a \in \text{Act}$ and $\varphi \in SForm$:

$$\text{fdv}(\text{TR}(\varphi)) \subseteq (\text{fdv}(\varphi) \cup \{x\}) \setminus \{x_a\}.$$

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

$$\llbracket \psi[e/x] \rrbracket \eta \varepsilon = \llbracket \psi \rrbracket \eta(\varepsilon[\llbracket e \rrbracket \varepsilon/x]).$$

Proof. Straightforward, by structural induction on α (property 1), on φ (property 2), and on ψ (property 3). \square

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

Proposition 4.2. *Let $X(x:D)$ be a linear process as defined above and let φ be a state formula. Then, for any $\rho \in \mathbf{PEnv}$ and $\varepsilon \in \mathbf{DEnv}$ such that $\text{fpv}(\varphi) \subseteq \text{supp}(\rho)$ and $\text{fdv}(\varphi) \subseteq \text{supp}(\varepsilon)$:*

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

Proof. By structural induction on φ , using Lemma 4.1. \square

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

$$\begin{aligned} \forall v:D. (X(v) \in \llbracket \varphi \rrbracket [[]]) &\leftrightarrow && \text{by Proposition 4.2} \\ \forall v:D. \llbracket \text{TR}(\varphi) \rrbracket \text{TR}([[]])([[]]v/x) &\leftrightarrow && \text{by definition of } \text{TR}(\rho) \\ \forall v:D. \llbracket \text{TR}(\varphi) \rrbracket [[]]v/x &\leftrightarrow && \text{by definition of } [\cdot] \eta \varepsilon \\ \llbracket \forall x:D. \text{TR}(\varphi) \rrbracket [[]] & & & \end{aligned}$$

(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:

$$\begin{aligned} X(v_0) \in \llbracket \varphi \rrbracket [[]] &\leftrightarrow && \text{by Proposition 4.2} \\ \llbracket \text{TR}(\varphi) \rrbracket \text{TR}([[]])([[]]v_0/x) &\leftrightarrow && \text{by definition of } \text{TR}(\rho) \\ \llbracket \text{TR}(\varphi) \rrbracket [[]]v_0/x &\leftrightarrow && \text{by definition of } [\cdot] \eta \varepsilon \\ \llbracket \forall x:D. (x = v_0) \rightarrow \text{TR}(\varphi) \rrbracket [[]] & & & \end{aligned}$$

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

4.3 Example (continued)

We continue the example from Section 3.1 by giving the translations of the formulas φ_1 , φ_2 , and φ_3 . So, to establish the validity of these formulas we must prove, respectively:

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

4.4 Proof rules

As shown in Section 4.2, the verification of a data-based temporal logic formula on a linear μCRL 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 [7]) in order to prove the validity of fixed point boolean 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 $\psi_2 \in BForm$ such that $fbv(\psi_2) \subseteq fbv(\psi_1)$ and $fdv(\psi_2) \subseteq fdv(\psi_1)$. The *application* of ψ_1 on ψ_2 is defined as follows:

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

Intuitively, $\psi_1[\psi_2]$ is obtained by substituting all the occurrences of $Z(e)$ in ψ_1 by ψ_2 , in which all occurrences of z have been replaced with the actual parameter e . The conditions on the variables of ψ_2 ensure that no free variables of ψ_2 become bound in $\psi_1[\psi_2]$. Also, whenever $fdv(\psi_2) = \{z\}$, we will write $\psi_2(e)$ for $\psi_2[e/z]$. The application k times of ψ_1 on ψ_2 , noted $\psi_1^k[\psi_2]$, is defined as follows:

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

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

$$\begin{array}{ll} \text{LFPUP} \frac{\forall k \geq 0. (\psi_2(k) \rightarrow \psi_1^k[ff])}{(\exists k \geq 0. \psi_2(k)) \rightarrow (\mu Z(z:D).\psi_2)(z)} & \text{GFPUP} \frac{\psi_2 \rightarrow \psi_1[\psi_2]}{\psi_2 \rightarrow (\nu Z(z:D).\psi_1)(z)} \\ \text{LFPDN} \frac{\psi_1[\psi_2] \rightarrow \psi_2}{(\mu Z(z:D).\psi_1)(z) \rightarrow \psi_2} & \text{GFPDN} \frac{\forall k \geq 0. (\psi_1^k[tt] \rightarrow \psi_2(k))}{(\nu Z(z:D).\psi_1)(z) \rightarrow (\forall k \geq 0. \psi_2(k))} \end{array}$$

where $\psi_2(k)$ means that the variable k , denoting a natural number, occurs free in ψ_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 4.3. *The rules LFPUP, LFPDN, GFPUP, and GFPDN defined above are sound w.r.t. the semantics of the boolean formulas $\psi \in BForm$.*

Proof. Given in Annex A. □

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 $\psi_1 \stackrel{\text{def}}{=} (b \wedge l \leq v) \vee (\neg b \wedge Z(v+1, \neg b)) \vee \exists m:\mathbb{N}. (b \wedge m \leq v \wedge Z(v-m, \neg b))$. In order to apply the rule LFPUP we must find some $\psi_2(k)$. We propose $\psi_2(k) \stackrel{\text{def}}{=} k > \text{if}(l \leq v, |\neg b|, 2(l-v) - |\neg b|)$. Here, *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 $\exists k \geq 0. (k > \text{if}(l \leq v, |\neg b|, 2(l-v) - |\neg b|))$, which is a tautology. So, if we can prove the premises of LFPUP we have shown that the temporal formula φ_1 is valid in all states of $X(v, b)$.

The premise of LFPUP has become $\forall k \geq 0. (k > \text{if}(l \leq v, \neg b, 2(l-v) - |\neg b|) \rightarrow \psi_1^k[\text{ff}])$. 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: $k' \geq \text{if}(l \leq v, \neg b, 2(l-v) - |\neg b|) \rightarrow (b \wedge l \leq v) \vee (\neg b \wedge \psi_1^{k'}[\text{ff}](v+1, \neg b)) \vee \exists m:\mathbb{N}. (b \wedge m \leq v \wedge \psi_1^{k'}[\text{ff}](v-m, \neg b))$. 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 $\exists m:\mathbb{N}. (m \leq v \wedge \psi_1^{k'}[\text{ff}](v-m, \neg b))$. Take $m = 0$. The proof obligation reduces to $\psi_1^{k'}[\text{ff}](v, \neg b)$. 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 $\psi_1^{k'}[\text{ff}](v+1, \neg b)$. The left hand side of the implication becomes $k' \geq \text{if}(l \leq v, 1, 2(l-v) - 1)$, which is easily seen (by distinguishing between the cases $l \leq v$, $l = v+1$, and $l > v+1$) to imply $(\psi_2(k'))(v+1, \neg b) = k' > \text{if}(l \leq v+1, 0, 2(l-v) - 2)$. 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 ψ_1 be the body of the μZ formula. We apply LFPDN, taking $\psi_2 \stackrel{\text{def}}{=} \text{ff}$. The left hand side $\psi_1[\psi_2]$ of the premise looks like $(\neg b \rightarrow \text{ff}) \wedge \forall m:\mathbb{N}. ((b \wedge m \leq v \wedge m \neq n) \rightarrow \text{ff})$, which is equivalent to ff . Thus, the fixed point formula is false for all $v \in \mathbb{N}$ and $b \in \mathbf{Bool}$.
3. We show that this formula is satisfied by the initial state of the system. Let ψ_1 be the body of the νZ formula. We must prove that $(v = v_0 \wedge b = \text{ff}) \rightarrow (\nu Z(p, r, v:\mathbb{N}, b:\mathbf{Bool}). \psi_1)(0, 0, v, b)$ for all $v \in \mathbb{N}$ and $b \in \mathbf{Bool}$. We solve this by showing a slightly stronger property, namely that $(v = v_0 + p - r) \rightarrow (\nu Z(p, r, v:\mathbb{N}, b:\mathbf{Bool}). \psi_1)(p, r, v, b)$, which implies the above boolean property by instantiating v, b, p , and r with $v_0, \text{ff}, 0$, and 0 , respectively. We apply GFPUP, taking $\psi_2 \stackrel{\text{def}}{=} (v = v_0 + p - r)$. The premise of GFPUP reduces to $(v = v_0 + p - r) \rightarrow (\forall l, m:\mathbb{N}. ((b \wedge m \leq v \wedge m = l) \rightarrow (l \leq v_0 + p - r \wedge v - m = v_0 + p - r - l)) \wedge (\neg b \rightarrow v + 1 = v_0 + p + 1 - r))$, which is easily seen to be a tautology. Hence, the initial state $X(v_0, \text{ff})$ satisfies φ_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 :

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

Data elements $d \in D$ are inserted in Q via $r(d)$ actions and are delivered by Q via $s(d)$ actions. The $|\cdot|$ 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 can be described by the linear process below:

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

The initial state of this process is $Q(\text{nil}, \text{nil})$, 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:

$$\begin{aligned} \varphi_1 \stackrel{\text{def}}{=} & (\nu Y(q). \forall d_0: D. [r(d_0)] Y(\text{in}(d_0, q)) \wedge [s(d_0)] (|q| > 0 \wedge \text{toe}(q) = d_0 \wedge Y(\text{untoe}(q))) \wedge \\ & [\neg \exists d_1: D. (s(d_1) \vee r(d_1))] Y(q) \\ &)(\text{nil}) \end{aligned}$$

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 $s(\dots)$ or $r(\dots)$ action.) We verify φ_1 in the initial state $Q(\text{nil}, \text{nil})$ of the system. This translates as follows:

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

Let ψ_1 be the body of the νZ formula. To show the boolean 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 $\psi_2 \stackrel{\text{def}}{=} (q_1 + q_2 = q)$. The premise $\psi_2 \rightarrow \psi_1[\psi_2]$ of GFPUP reduces to the following three implications:

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

These properties can be easily shown using an appropriate axiomatization of the queue operators. Now, by instantiating q to nil , and since $(q_1 = \text{nil} \wedge q_2 = \text{nil}) \rightarrow (q_1 + q_2 = \text{nil})$, this implies that $(q_1 = \text{nil} \wedge q_2 = \text{nil}) \rightarrow (\nu Z(q_1, q_2, q). \psi_1)(q_1, q_2, \text{nil})$ for all q_1 and q_2 . Hence, $Q(\text{nil}, \text{nil})$ satisfies φ_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 $r(d_0)$:

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

The verification of φ_2 in all the states of Q translates to the following boolean formula:

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

We write ψ_1 for the body of the μZ formula. Since the disjunct $\exists d: D. (d = d_0)$ is trivially true, ψ_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, φ_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 $s(d_0)$:

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

The verification of φ_3 in all the states of Q translates as follows:

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

Let ψ_1 be the body of the μZ formula. Observing that $\psi_1[\text{ff}] = \text{ff}$, the rule LFPDN leads to $(\mu Z(q_1, q_2). \psi_1)(q_1, q_2) \rightarrow \text{ff}$ for every q_1 and q_2 . Then, the whole boolean formula reduces to $\forall d: D. d \neq d_0$, which is obviously false. Hence, φ_3 does not hold in any state of Q . This happens because one can always insert data elements into Q (see formula φ_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 φ_3 by taking into account only the execution paths which are *fair* w.r.t. the action $s(d_0)$, i.e., those paths which cannot infinitely often enable $s(d_0)$ without executing infinitely often an $s(d_0)$ -transition:

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

The formula φ_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 [24], where it was shown that it expresses the reachability on fair paths.

The verification of φ_4 in all the states of Q translates as follows:

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

Let ψ_1 be the body of the νZ_1 formula. We show the boolean 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 boolean formula above follows by instantiating q_1 with $\text{in}(d_0, q_1)$, since $d_0 \in \text{in}(d_0, q_1) + q_2$ is trivially true.)

We apply the rule GFPUP on ψ_1 , taking $\psi'_1 \stackrel{\text{def}}{=} d_0 \in q_1 + q_2$. The premise $\psi_1[\psi'_1]$ reduces to the following four implications:

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

where ψ_2 is the body of the μZ_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 $\psi_2'(k) \stackrel{\text{def}}{=} d_0 \in q_1 + q_2 \wedge 2|q_1| + |q_2| \leq k$ (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 $\forall k \geq 0. (\psi_2'(k) \rightarrow \psi_2^k[\text{ff}])$ 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 $(d_0 \in q_1 + q_2 \wedge 2|q_1| + |q_2| \leq k' + 1) \rightarrow \psi_2^{k'+1}[\text{ff}]$. We distinguish two cases:

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

This concludes the proof that all the states of Q satisfy φ_4 .

A Proofs

We provide in this annex the proof of Proposition 4.3, which states the soundness of the deduction rules LFPUP, LFPDN, GFPUP, and GFPDN associated to the minimal and maximal fixed point operators. We start by giving a lemma that relates the semantics of an application $\psi_1[\psi_2]$ with the semantics of the boolean formulas ψ_1 and ψ_2 .

Lemma A.1. *Let $\sigma Z(z:D). \psi_1$ be a fixed point formula and let $\psi_2 \in BForm$ such that $\text{fbv}(\psi_2) \subseteq \text{fbv}(\psi_1)$ and $\text{fdv}(\psi_2) \subseteq \text{fdv}(\psi_1)$. Then, for every $\eta \in \mathbf{BEnv}$ and $\varepsilon \in \mathbf{DEnv}$ such that $\text{fbv}(\psi_1) \subseteq \text{supp}(\eta)$ and $\text{fdv}(\psi_1) \subseteq \text{supp}(\varepsilon)$:*

$$\llbracket \psi_1[\psi_2] \rrbracket \eta \varepsilon = \llbracket \psi_1 \rrbracket (\eta[\lambda v:D. \llbracket \psi_2 \rrbracket \eta(\varepsilon[v/z])/Z]) \varepsilon.$$

Proof. Straightforward, by structural induction on ψ_1 . □

The following lemma, which relates the semantics of the iterative application $\psi_1^k[\psi_2]$ with the semantics of ψ_2 and the functional associated to ψ_1 , will be useful in showing the soundness of the LFPUP and GFPDN rules.

Lemma A.2. Let $\sigma Z(z:D).\psi_1$ be a fixed point formula and let $\psi_2 \in BForm$ such that $fbv(\psi_2) \subseteq fbv(\psi_1)$ and $fdv(\psi_2) \subseteq fdv(\psi_1)$. Let $\eta \in \mathbf{BEnv}$ and $\varepsilon \in \mathbf{DEnv}$ such that $fbv(\psi_1) \subseteq \text{supp}(\eta)$ and $fdv(\psi_1) \subseteq \text{supp}(\varepsilon)$. Then, for all $k \geq 0$:

$$\lambda v:D. \llbracket \psi_1^k[\psi_2] \rrbracket \eta(\varepsilon[v/z]) = \Psi_{1\eta\varepsilon}^k(\lambda v':D. \llbracket \psi_2 \rrbracket \eta(\varepsilon[v'/z]))$$

where $\Psi_{1\eta\varepsilon} \stackrel{\text{def}}{=} \lambda G : D \rightarrow \mathbf{Bool}.\lambda v:D. \llbracket \psi_1 \rrbracket (\eta[G/Z])(\varepsilon[v/z])$ is the functional associated to $\sigma Z(z:D).\psi_1$.

Proof. By induction on k , using also Lemma A.1.

- $k = 0$:

$$\begin{aligned} \lambda v:D. \llbracket \psi_1^0[\psi_2] \rrbracket \eta(\varepsilon[v/z]) &= && \text{by definition of } \psi_1^k \\ \lambda v:D. \llbracket \psi_2 \rrbracket \eta(\varepsilon[v/z]) &= && \text{by definition of } \Psi_{1\eta\varepsilon}^k \\ \Psi_{1\eta\varepsilon}^0(\lambda v':D. \llbracket \psi_2 \rrbracket \eta(\varepsilon[v'/z])). & & & \end{aligned}$$

- $k = k' + 1$:

$$\begin{aligned} \lambda v:D. \llbracket \psi_1^{k'+1}[\psi_2] \rrbracket \eta(\varepsilon[v/z]) &= && \text{by definition of } \psi_1^k \\ \lambda v:D. \llbracket \psi_1[\psi_1^{k'}[\psi_2]] \rrbracket \eta(\varepsilon[v/z]) &= && \text{by Lemma A.1} \\ \lambda v:D. \llbracket \psi_1 \rrbracket (\eta[\lambda v':D. \llbracket \psi_1^{k'}[\psi_2] \rrbracket \eta((\varepsilon[v/z])[v'/z])/Z])(\varepsilon[v/z]) &= && \text{by definition of } \Psi_{1\eta\varepsilon} \\ \Psi_{1\eta\varepsilon}(\lambda v':D. \llbracket \psi_1^{k'}[\psi_2] \rrbracket \eta(\varepsilon[v'/z])) &= && \text{by inductive hypothesis} \\ \Psi_{1\eta\varepsilon}(\Psi_{1\eta\varepsilon}^k(\lambda v':D. \llbracket \psi_2 \rrbracket \eta(\varepsilon[v'/z]))) &= && \text{by definition of } \Psi_{1\eta\varepsilon}^k \\ \Psi_{1\eta\varepsilon}^{k'+1}(\lambda v':D. \llbracket \psi_2 \rrbracket \eta(\varepsilon[v'/z])). & & & \end{aligned}$$

□

We are now able to prove Proposition 4.3.

Proof of Proposition 4.3. We only show the soundness of the rules GFPUP and LFPUP, the proofs of the rules LFPDN and GFPDN being their dual counterparts, respectively. Let $\sigma Z(z:D).\psi_1$ be a fixed point formula and let $\psi_2 \in BForm$ such that $fbv(\psi_2) \subseteq fbv(\psi_1)$ and $fdv(\psi_2) \subseteq fdv(\psi_1)$.

- Rule GFPUP. We must show that the following implication holds:

$$\forall \eta.\forall \varepsilon. \llbracket \psi_2 \rightarrow \psi_1[\psi_2] \rrbracket \eta\varepsilon \rightarrow \forall \eta.\forall \varepsilon. \llbracket \psi_2 \rightarrow (\nu Z(z:D).\psi_1)(z) \rrbracket \eta\varepsilon \quad (1)$$

where $fbv(\psi_1) \subseteq \text{supp}(\eta)$ and $fdv(\psi_1) \subseteq \text{supp}(\varepsilon)$. Let $\eta \in \mathbf{BEnv}$ and $\varepsilon \in \mathbf{DEnv}$ having appropriate supports. We consider the functional $\Psi_{1\eta\varepsilon} \stackrel{\text{def}}{=} \lambda G : D \rightarrow \mathbf{Bool}.\lambda v:D. \llbracket \psi_1 \rrbracket (\eta[G/Z])(\varepsilon[v/z])$ associated to the $\nu Z(z:D).\psi_1$ formula, and we define the function $G_{2\eta\varepsilon} \stackrel{\text{def}}{=} \lambda v:D. \llbracket \psi_2 \rrbracket \eta(\varepsilon[v/z])$ associated to ψ_2 . Assuming that the left hand side of (1) is true, we first show that $G_{2\eta\varepsilon} \rightarrow \Psi_{1\eta\varepsilon}(G_{2\eta\varepsilon})$. Let $v \in D$.

$$\begin{aligned} G_{2\eta\varepsilon}(v) &= && \text{by definition of } G_{2\eta\varepsilon} \\ \llbracket \psi_2 \rrbracket \eta(\varepsilon[v/z]) &\rightarrow && \text{by assumption} \\ \llbracket \psi_1[\psi_2] \rrbracket \eta(\varepsilon[v/z]) &= && \text{by Lemma A.1} \\ \llbracket \psi_1 \rrbracket (\eta[\lambda v':D. \llbracket \psi_2 \rrbracket \eta((\varepsilon[v/z])[v'/z])/Z])(\varepsilon[v/z]) &= && \text{by definition of } G_{2\eta\varepsilon} \\ \llbracket \psi_1 \rrbracket (\eta[G_{2\eta\varepsilon}/Z])(\varepsilon[v/z]) &= && \text{by definition of } \Psi_{1\eta\varepsilon} \\ (\Psi_{1\eta\varepsilon}(G_{2\eta\varepsilon}))(v). & & & \end{aligned}$$

From $G_{2\eta\varepsilon} \rightarrow \Psi_{1\eta\varepsilon}(G_{2\eta\varepsilon})$, since $\Psi_{1\eta\varepsilon}$ is monotonic and $(D \rightarrow \mathbf{Bool}) \rightarrow (D \rightarrow \mathbf{Bool})$ is a complete lattice, Tarski's theorem implies that:

$$G_{2\eta\varepsilon} \rightarrow \nu\Psi_{1\eta\varepsilon} \quad (2)$$

For all $\eta \in \mathbf{BEnv}$ and $\varepsilon \in \mathbf{DEnv}$, whose supports satisfy the appropriate conditions, we obtain:

$$\begin{aligned} \llbracket \psi_2 \rrbracket \eta\varepsilon &= && \text{by definition of } G_{2\eta\varepsilon} \\ G_{2\eta\varepsilon}(\varepsilon(z)) &\rightarrow && \text{by (2)} \\ (\nu\Psi_{1\eta\varepsilon})(\varepsilon(z)) &= && \text{by definition of } \llbracket \cdot \rrbracket \eta\varepsilon \\ \llbracket (\nu Z(z:D).\psi_1)(z) \rrbracket \eta\varepsilon & & & \end{aligned}$$

which is equivalent to the right hand side of (1).

- Rule LFUP. We must show that the following implication holds:

$$\forall \eta. \forall \varepsilon. \llbracket \forall k \geq 0. (\psi_2(k) \rightarrow \psi_1^k[\mathit{ff}]) \rrbracket \eta\varepsilon \rightarrow \forall \eta. \forall \varepsilon. \llbracket (\exists k \geq 0. \psi_2(k) \rightarrow (\mu Z(z:D).\psi_1)(z)) \rrbracket \eta\varepsilon \quad (3)$$

where $fbv(\psi_1) \subseteq \text{supp}(\eta)$ and $fdv(\psi_1) \subseteq \text{supp}(\varepsilon)$. Let $\eta \in \mathbf{BEnv}$ and $\varepsilon \in \mathbf{DEnv}$ having appropriate supports. The functional $\Psi_{1\eta\varepsilon}$ associated to the $\mu Z(z:D).\psi_1$ formula is defined as usual. We also define the function $G_{2\eta\varepsilon} \stackrel{\text{def}}{=} \lambda l:\mathbb{N}. \lambda v:D. \llbracket \psi_2(k) \rrbracket \eta(\varepsilon[l/k, v/z])$ associated to $\psi_2(k)$. Assuming that the left hand side of (3) is true, we first show that $G_{2\eta\varepsilon}(l) \rightarrow \Psi_{1\eta\varepsilon}^l(\lambda v':D.\mathit{ff})$ for all $l \geq 0$. Let $v \in D$.

$$\begin{aligned} (G_{2\eta\varepsilon}(l))(v) &= && \text{by definition of } G_{2\eta\varepsilon} \\ \llbracket \psi_2(k) \rrbracket \eta(\varepsilon[l/k, v/z]) &\rightarrow && \text{by assumption} \\ \llbracket \psi_1^k[\mathit{ff}] \rrbracket \eta(\varepsilon[l/k, v/z]) &= && \text{by definition of } \psi_1^k \\ \llbracket \psi_1^l[\mathit{ff}] \rrbracket \eta(\varepsilon[v/z]) &= && \text{by Lemma A.2} \\ (\Psi_{1\eta\varepsilon}^l(\lambda v':D.\mathit{ff}))(v) & & & \end{aligned}$$

Since $\Psi_{1\eta\varepsilon}$ is monotonic, it is simple to show, by induction on l , that $\Psi_{1\eta\varepsilon}^l(\lambda v':D.\mathit{ff}) \rightarrow \mu\Psi_{1\eta\varepsilon}$ for all $l \geq 0$. Together with the property $\forall l \geq 0. (G_{2\eta\varepsilon}(l) \rightarrow \Psi_{1\eta\varepsilon}^l(\lambda v':D.\mathit{ff}))$ shown above, this implies:

$$(\exists l \geq 0. G_{2\eta\varepsilon}(l)) \rightarrow \mu\Psi_{1\eta\varepsilon} \quad (4)$$

For all $\eta \in \mathbf{BEnv}$ and $\varepsilon \in \mathbf{DEnv}$, whose supports satisfy the appropriate conditions, we obtain:

$$\begin{aligned} \llbracket \exists k \geq 0. \psi_2(k) \rrbracket \eta\varepsilon &= && \text{by definition of } \llbracket \cdot \rrbracket \eta\varepsilon \\ \exists l \geq 0. \llbracket \psi_2(k) \rrbracket \eta(\varepsilon[l/k]) &= && \text{by definition of } G_{2\eta\varepsilon} \\ \exists l \geq 0. (G_{2\eta\varepsilon}(l))(\varepsilon(z)) &\rightarrow && \text{by (4)} \\ (\mu\Psi_{1\eta\varepsilon})(\varepsilon(z)) &= && \text{by definition of } \llbracket \cdot \rrbracket \eta\varepsilon \\ \llbracket (\mu Z(z:D).\psi_1)(z) \rrbracket \eta\varepsilon & & & \end{aligned}$$

which is equivalent to the right hand side of (3). \square

References

- [1] H. R. Andersen. Model checking and boolean graphs. *Theoretical Computer Science*, 126(1):3–30, 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] M.A. Bezem and J.F. Groote. Invariants in process algebra with data. In B. Jonsson and J. Parrow, editors, *Proceedings CONCUR'94*, Uppsala, Sweden, LNCS 836, pp. 401–416, Springer Verlag, 1994.
- [4] M.A. Bezem and J.F. Groote. A correctness proof of a one bit sliding window protocol in μ CRL. *The Computer Journal*, 37(4):289–307, 1994.
- [5] D. Bosscher and A. Ponse. Translating a process algebra with symbolic data values to linear format. In U.H. Engberg, K.G. Larsen, and A. Skou, editors, *Proceedings TACAS'95*, Aarhus, Denmark, BRICS Notes Series, pages 119–130, University of Aarhus, 1995.
- [6] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
- [7] D. van Dalen. *Logic and Structure*. Springer Verlag, 1994.
- [8] M. Dam. CTL* and ECTL* as fragments of the modal μ -calculus. *Theoretical Computer Science*, 126(1):77–96, 1994.
- [9] M. Dam. Model checking mobile processes. *Information and Computation*, 129:35–51, 1996.
- [10] R. De Nicola and F. W. Vaandrager. Action versus state based logics for transition systems. *Proceedings Ecole de Printemps on Semantics of Concurrency*, LNCS 469, Springer Verlag, 1990.
- [11] E. A. Emerson and J. Y. Halpern. “Sometimes” and “Not Never” revisited: on branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.
- [12] E. A. Emerson and C-L. Lei. Efficient model checking in fragments of the propositional μ -calculus. *Proceedings of the 1st LICS*, pp. 267–278, 1986.
- [13] L.-Å. Fredlund, J.F. Groote and H. Korver. Formal verification of a leader election protocol in process algebra. *Theoretical Computer Science*, 177:459–486, 1997.
- [14] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18, pp. 194–211, 1979.
- [15] J.F. Groote. A note on n similar parallel processes. In S. Gnesi and D. Latella, editors, *Second International ERCIM Workshop on Formal Methods for Industrial Critical Systems*, pp. 65–75, Cesena, Italy, 1997. (See also Report CS-R9626, CWI, Amsterdam, 1996).
- [16] J.F. Groote, F. Monin and J. Springintveld. A computer checked algebraic verification of a distributed summing protocol. Computer Science Report 97/14, Department of Mathematics and Computer Science, Eindhoven University of Technology. 1997.
- [17] J.F. Groote and J.C. van de Pol. A bounded retransmission protocol for large data packets. A case study in computer checked verification. In M. Wirsing and M. Nivat, Editors, *Proceedings of AMAST'96*, Munich, LNCS 1101, pp. 536–550, Springer Verlag, 1996.

- [18] J.F. Groote and A. Ponse. The syntax and semantics of μ CRL. In A. Ponse, C. Verhoef and S.F.M. van Vlijmen, eds, *Algebra of Communicating Processes*, Workshops in Computing, pp. 26–62, 1994.
- [19] J.F. Groote and J. Springintveld. Focus points and convergent process operators. A proof strategy for protocol verification. Technical Report 142, Logic Group Preprint Series, Utrecht University, 1995. This report also appeared as Technical Report CS-R9566, Centrum voor Wiskunde en Informatica, 1995.
- [20] A. Kindler, W. Reisig, H. Völzer, and R. Walter. Petri net based verification of distributed algorithms: an example. *Formal Aspects of Computing*, 9:409–424, 1997.
- [21] D. Kozen. Results on the propositional μ -calculus. *Theoretical Computer Science* 27, pp. 333–354, 1983.
- [22] Z. Manna and A. Pnueli. Adequate proof principles for invariance and liveness properties of concurrent programs. *Science of Computer Programming* 32:257–289, 1984.
- [23] Z. Manna and A. Pnueli. The Temporal Logic of Reactive and Concurrent Systems, volume I: Specification. Springer-Verlag, 1992.
- [24] J-P. Queille and J. Sifakis. Fairness and related properties in transition systems — a temporal logic to deal with fairness. *Acta Informatica*, 19:195–220, 1983.
- [25] J. Rathke and M. Hennessy. Local model checking for a value-based modal μ -calculus. Technical Report 5/96, School of Cognitive and Computing Sciences, University of Sussex, 1996.
- [26] C. Shankland. The Tree Identify Protocol of IEEE 1394. Unpublished technical report, University of Stirling, 1998.
- [27] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics* 5, pp. 285–309, 1955.
- [28] B. Vergauwen and J. Lewi. A linear algorithm for solving fixed-point equations on transition systems. *Proceedings of the 17th Colloquium on Trees in Algebra and Programming CAAP '92 (Rennes, France)*, LNCS 581, pp. 322–341, Springer Verlag, 1992.