Specifying and Verifying Parametric Processes*

Wiesław Pawłowski[1] Paweł Pączkowski[2] Stefan Sokołowski[1]

[1] Institute of Computer Science, Polish Academy of Sciences,
ul. Abrahama 18, 81-825 Sopot, Poland.
E-mail: {w.pawlowski,s.sokolowski}@ipipan.gda.pl
[2] Department of Computing Science, Chalmers University of Technology
and Göteborgs University, 412-96 Göteborg, Sweden, and
Institute of Mathematics, University of Gdańsk,
ul. Wita Stwosza 57, 80-952 Gdańsk, Poland,
E-mail: matpmp@univ.gda.pl

Abstract. A process algebra and readiness specifications introduced by Olderog are extended to a framework in which processes parameterized with other processes can be specified, defined and verified. Higher order process-parameters are allowed. The formalism resembles typed lambda calculus built on top of a process algebra, where specifications play the role of types. A proof system for deriving judgements "parametric process meets a specification" is given.

1 Introduction

A typical approach to software development is that of decomposition of large tasks into smaller subtasks. To formalize the decomposition principle, let us consider a set of constructions *Constr* (programs, processes, etc.), a set of specifications *Spec*, and satisfaction relation **sat** between constructions and specifications (**sat** can be given, for example, by a proof system). The triple

$$(\mathit{Constr}, \mathit{Spec}, \mathbf{sat}) \tag{1}$$

will be referred to in the sequel as *specification system* (see [8]). Assume we are faced with a task of providing a construction c that satisfies a given specification S (denoted $c\ \mathbf{sat}\ S$). Decomposing this task amounts to dividing it into the following subtasks:

1. provide sub-constructions $c_1, \ldots, c_n$ that satisfy some sub-specifications $S_1, \ldots, S_n$ into which the original specification S is decomposed, and
2. provide a method of combining $c_1, \ldots, c_n$ so as to obtain such a c that $c\ \mathbf{sat}\ S$.

After [8] we adopt a point of view that the method mentioned in point 2 above should be a *construction* itself, i.e. belong to *Constr* and we rephrase 2 as

* This work was partially supported by the following grants: KBN Grant No. PB-1312/P3/92/02, CRIT IC 1010/II, and (in case of the second author) ESPRIT BRA CONCUR2.

$2'$. provide a *parametric construction* f such that

$$f \textbf{ sat } S_1 \to S_2 \to \cdots \to S_n \to S$$

The "functional" specification $S_1 \to S_2 \to \cdots \to S_n \to S$, which is assumed to belong to *Spec*, is intended to specify precisely those parametric constructions that yield objects satisfying S when applied to sub-constructions satisfying $S_1, \ldots, S_n$. Thus, if the decomposition steps 1 and $2'$ are realized then $fc_1 \ldots c_n$ defines a c we were looking for.

The specifications S_1,...,S_n and $S_1 \to S_2 \to \cdots \to S_n \to S$, may be further decomposed to ease the realization; the decomposition of the functional specification $S_1 \to S_2 \to \cdots \to S_n \to S$ may lead to higher order parameterization.

The purpose of this work is to apply the methodology sketched above in the context of concurrent process specification and verification. As a starting point we adopt a process algebra and readiness logic for specifying processes proposed in [5]. Olderog extends the process algebra by introducing mixed terms, where a specification can become a part of a process term. This enables transformation based development of processes. We consider a different option and introduce parametric processes that contain process variables. We allow higher order parameters: process variables can represent also parametric processes. Technically speaking, Olderog's process algebra is extended with abstraction and application primitives. This leads to a formalism resembling a typed lambda calculus built on top of a process algebra, where specifications play the role of types. Mixed terms can be viewed as a special case of parametric processes.

We provide a proof system for deriving judgements "(parametric) process meets a specification". To this end, we needed to reformulate the proof system for nonparametric processes that was given in the final chapter of [5]. Derivations in that proof system use mixed terms. We provide a direct proof system for nonparametric processes, which does not appeal to mixed terms and which is shown to be equivalent to that of [5]. The direct proof system is extended to the parametric case. These are the main technical contributions of this paper.

The parametric framework we introduce allows us to express intermediate stages of process development in a more explicit way than in Olderog's mixed-term approach. Although we do not use mixed terms, we retain the ability to construct processes in a similar manner as with mixed-term transformations, adding the capacity for higher order parameters. To illustrate these points let us sketch an example inspired by [5].

Consider a sequence of transformations that derives cnt_2, a counter of capacity 2, as a parallel composition of two counters of capacity 1.

$$cnt_2 \;\equiv\!> \;\cdots\; \equiv\!> \; m \; \equiv\!> \;\cdots\; \equiv\!> \; CNT_2$$

This derivation starts from a specification CNT_2 of a counter of capacity 2 and progresses through a number of transformation steps. The crucial intermediate stage is a mixed term m such that $cnt_2 \equiv\!> \cdots \equiv\!> m$ and the only specification appearing in m is CNT_1, a specification of a counter of capacity 1.

In our approach, an analogous process construction is rendered as follows. We formulate our task as: provide cnt_2 such that

$$cnt_2 \textbf{ sat } CNT_2$$

This task is decomposed into two subtasks: constructing a counter of capacity 1, cnt_1, and a parametric process $double_1$ that constructs a counter of capacity 2 given a counter of capacity 1, i.e. cnt_1 and $double_1$ such that

$$cnt_1 \textbf{ sat } CNT_1$$

$$double_1 \textbf{ sat } CNT_1 \to CNT_2 \tag{2}$$

Parametric process $double_1$ is a counterpart of m. The assertion (2), which expresses the required property of $double_1$, is equivalent to the more verbose description of m given above.

We can easily specify more elaborate tasks such as

$$(CNT_1 \to CNT_2) \to (CNT_1 \to CNT_4) \tag{3}$$

which means "combine counters of capacity 1 to get a counter of capacity 4, assuming that we can construct counter of capacity 2 using counters of capacity 1", The ability to specify such more involved tasks as well as intermediate construction stages such as $double_1$ clarifies the decomposition steps and allows for the reuse of intermediate construction stages.

The concept of processes with process parameters (but only first order ones) has already appeared in [4] (Chapter 9). However, this method of constructing CCS processes is not reflected in their verification because the bisimulation proof technique used in this reference requires a considerable effort in decomposing large verification tasks into smaller ones (see e.g. [3]).

The formalism we develop allows decomposition steps of the form 1 and $2'$ that may involve processes with higher-order parameters and helps one to do a systematic book-keeping of process dependencies, which is useful in higher-order constructions. We view our approach as an alternative to process development methods that use transformation or refinement techniques (see e.g. [2], [5], [6]).

The paper is organized as follows. In Section 2 we give an overview of a process algebra of nonparametric processes and a logic for specifying them that appear in [5]. A new, direct proof system for nonparametric processes is given in Section 3. In Section 4 we define parametric processes and extend the proof system to the parametric case. Section 5 contains examples.

Due to space limitations most proofs were omitted. They can be found in [7].

2 Nonparametric processes and specifications

The class of processes that we consider and a logic for specifying their properties are taken from [5]. We give a brief overview of the main notions, referring to [5] for a comprehensive exposition. [7] contains an extended version of this section.

Processes. Let *Comm* be an infinite set of *communications*. Together with a special symbol τ representing a hidden or internal activity of a process it will constitute the set of process *actions*.

Processes will be defined as a subset of the set of *recursive terms*. Recursive terms have the following abstract syntax

$$P ::= \ stop_A \mid a.P \mid P + Q \mid P \parallel Q \mid P[b/a] \mid P\backslash a \mid X \mid \mu X.P$$

where P and Q range over recursive terms, X ranges over process variables, A over finite sets of communications, a and b over communications. Essentially, the process operators in the syntax above are those of CSP apart from plus, which is given semantics as in CCS.

Iterated applications of renaming and hiding will be abbreviated as follows: $P[\bar{b}/\bar{a}]$ will stand for $P[b_1/a_1]\dots[b_n/a_n]$, where $\bar{b} = b_1, \dots, b_n, \bar{a} = a_1, \dots, a_n$; $P\backslash A$ will denote $P\backslash a_1 \dots \backslash a_n$, where $A = \{a_1, \dots a_n\}$. Notation $\{\dots/\dots\}$ is used for syntactic substitutions.

With each recursive term P we associate its *(communication) alphabet* $\alpha(P)$. It is assumed that each process variable X has an associated alphabet $\alpha(X)$. The communication alphabets of remaining terms are defined inductively as follows:

$$\begin{aligned} \alpha(stop_A) &= A \\ \alpha(a.P) &= \{a\} \cup \alpha(P) \\ \alpha(P + Q) &= \alpha(P) \cup \alpha(Q) \\ \alpha(P \parallel Q) &= \alpha(P) \cup \alpha(Q) \end{aligned} \qquad \begin{aligned} \alpha(P[b/a]) &= (\alpha(P) - \{a\}) \cup \{b\} \\ \alpha(P\backslash a) &= (\alpha(P) - \{a\}) \\ \alpha(\mu X.P) &= \alpha(X) \cup \alpha(P) \end{aligned}$$

Definition 1. We say that a recursive term P is a *process term* (or simply, a process) if it satisfies the following conditions: *(1)* P is action guarded, i.e. appearances of process variables in recursive subprocesses of P are guarded, *(2)* every subterm $a.Q$ of P satisfies $a \in \alpha(Q)$, *(3)* every subterm $Q+R$ of P satisfies $\alpha(Q) = \alpha(R)$, *(4)* every subterm $\mu X.Q$ of P satisfies $\alpha(X) = \alpha(Q)$.

The set of all process terms will be denoted by *Proc*. A process term is *closed* if it does not contain free process identifiers. The set of all closed process terms will be denoted by *CProc*.

Semantics. Processes are given *readiness semantics*. It describes process behaviour in terms of *ready pairs* $(tr, \mathcal{F})$, where tr is finite sequence of communications called *trace*, and $\mathcal{F}$, called *ready set*, is a set of communications in which a process is ready to engage after "performing" the trace tr. Besides the ready pairs the semantic domain $\mathrm{DOM}_{\mathcal{R}}$ contains elements representing undesirable behaviours of processes: initial instability and divergence.

The readiness semantics for closed process terms is a function:

$$\mathcal{R}^*[\![\cdot]\!] : CProc \to \mathrm{DOM}_{\mathcal{R}}$$

which can be given both both operational and denotational definitions. There is a natural partial ordering on the semantic domain $\mathrm{DOM}_{\mathcal{R}}$ denoted by $\sqsubseteq$.

Example 1. Below we define a counter of capacity 1. Upon communication *up* (*dn*) the value stored in the counter is increased (decreased) by 1. Symbol $\equiv$ stands for the definitional equality.

$$cnt_1 \;\equiv\; \mu X\,.\,up.dn.X$$

$\alpha(cnt_1) = \{up, dn\}$, $\mathcal{R}^*[\![cnt_1]\!] = (\alpha(cnt_1), R)$, where R, is the set of ready pairs of cnt_1:

$$R \;=\; \{(tr, \mathcal{F}) \mid \exists k \geq 0\ (tr = (up\,dn)^k \,\wedge\, \mathcal{F} = \{up\}) \,\vee\, (tr = (up\,dn)^k up \,\wedge\, \mathcal{F} = \{dn\})\}$$

Readiness specifications. Processes are specified by formulas of a many-sorted predicate logic, called *readiness logic*, which has sorts of traces and ready sets, among others, and allows one to define sets of ready pairs in a direct manner. Readiness logic is fairly expressive, for example, regular sets over *Comm* can be expressed in it.

A *readiness specification* is a formula of readiness logic, in which at most the distinguished trace variable h and the distinguished ready set variable F are free, and which satisfies some natural syntactic restrictions. Since a specification has at most two free variables h and F, it can be interpreted semantically with respect to an evaluation $(tr, \mathcal{F})$ of h and F. We will write

$$(tr, \mathcal{F}) \models S \quad \text{and} \quad\quad \models S$$

if S is satisfied by, respectively, a ready pair $(tr, \mathcal{F})$ and all ready pairs.

A specification determines a set of ready pairs which satisfy it. Moreover, with every specification S its alphabet, denoted by $\alpha(S)$, can be associated. These two facts allow one to define *readiness semantics of readiness specifications*

$$\mathcal{R}^*[\![\cdot]\!] : \mathit{Spec} \rightarrow \mathrm{DOM}_{\mathcal{R}}$$

where *Spec* denotes the set of specifications.

Example 2. CNT_n below is a formula of readiness logic that instantiated with any natural number n gives a specification of a counter of capacity n. The alphabet of CNT_n consists of communications *up* and *dn*, which have the same meaning as in Example 1. The symbol $up\#h$ ($dn\#h$) denotes the number of *up* (*dn*) communications in trace h.

$$\begin{aligned} CNT_n \equiv\ & (0 \leq up\#h - dn\#h \leq n) \wedge (up\#h - dn\#h < n \;\Rightarrow\; up \in F) \wedge \\ & (up\#h - dn\#h > 0 \;\Rightarrow\; dn \in F) \end{aligned}$$

By instantiating n to 1 or 2 in CNT_n we get CNT_1 and CNT_2 that appeared in Introduction.

Satisfaction relation. Since processes as well as specifications are interpreted semantically over readiness domain, the satisfaction relation has a simple semantic definition:

$$P \text{ } \mathbf{sat} \text{ } S \quad \text{iff} \quad \mathcal{R}^*[\![P]\!] \sqsupseteq \mathcal{R}^*[\![S]\!],$$

which means that

– process P and specification S have the same alphabets, i.e. $\alpha(P) = \alpha(S)$,
– $\mathcal{R}^*[\![P]\!]$ contains only ready pairs (no divergence or initial instability) and for each ready pair $(tr, \mathcal{F})$ in $\mathcal{R}^*[\![P]\!]$ we have $(tr, \mathcal{F}) \models S$.

Proof system for sat via mixed terms. *Mixed terms* are a mixture of readiness specifications and process terms and are formally defined by allowing specifications in recursive terms

$$P \ ::= \ S \mid \mathit{stop}_A \mid a.P \mid \ \ldots$$

and generalizing the definitions of alphabets and processes in a straightforward way.

Since processes and specifications are interpreted in the same semantic domain, there is no difficulty in extending readiness semantics to mixed terms. The satisfaction relation on mixed terms, denoted by $\equiv>$, is defined as

$$M_1 \equiv> M_2 \quad \text{iff} \quad \mathcal{R}^*[\![M_1]\!] \sqsupseteq \mathcal{R}^*[\![M_2]\!],$$

where M_1, M_2 are mixed terms. The relation **sat** can be viewed as a special case of $\equiv>$ restricted to the pairs $P \equiv> S$ where P is a process and S is a specification.

In the final chapter of [5] a proof system for $\equiv>$ is given, which obviously allows us to derive assertions of the form P **sat** S, but mixed terms will be used in the derivations.

3 Proof system for sat

The first step towards introducing parametricity into the formalism proposed by Olderog is to provide a proof system for **sat**, which does not appeal to mixed terms. In Table 1 we propose such a proof system, whose rules bear a clear resemblance to Olderog's proof rules for $\equiv>$.

The judgements of the proof system for **sat** have the form

$$\Gamma \vdash P \text{ } \mathbf{sat} \text{ } S, \tag{4}$$

where $P \in \mathit{Proc}$, $S \in \mathit{Spec}$ and Γ is a *set of assumptions*,

$$\Gamma = \{X_1 \text{ } \mathbf{sat} \text{ } S_1\,, \ldots, \, X_n \text{ } \mathbf{sat} \text{ } S_n\}.$$

The sets of assumptions are needed to handle recursion.

(Consequence)
$$\frac{\models S \Rightarrow T \qquad \Gamma \vdash P \textbf{ sat } S}{\Gamma \vdash P \textbf{ sat } T}, \quad \text{where } \alpha(S) = \alpha(T)$$

(Deadlock)
$$\Gamma \vdash \mathit{stop}_A \textbf{ sat } h{\downarrow_A} \leq \varepsilon$$

(Prefix)
$$\frac{\models S\{\varepsilon/h, \{a\}/F\} \qquad \Gamma \vdash P \textbf{ sat } S\{ah/h\}}{\Gamma \vdash a.P \textbf{ sat } S}, \quad \text{where } a \in \alpha(S)$$

(Choice)
$$\frac{\Gamma \vdash P \textbf{ sat } S \qquad \Gamma \vdash Q \textbf{ sat } T}{\Gamma \vdash P + Q \textbf{ sat } \varphi_+(S,T)}, \quad \text{where } \alpha(S) = \alpha(T)$$

(Parallel)
$$\frac{\Gamma \vdash P \textbf{ sat } S \qquad \Gamma \vdash Q \textbf{ sat } T}{\Gamma \vdash P \parallel Q \textbf{ sat } \varphi_{\parallel}(S,T)}$$

(Renaming)
$$\frac{\Gamma \vdash P \textbf{ sat } S}{\Gamma \vdash P[\bar{b}/\bar{a}] \textbf{ sat } \varphi_{[\,]}(S, \bar{b}, \bar{a})}$$

(Hiding)
$$\frac{\models S\{F - B/F\} \Rightarrow T \qquad \models S \Rightarrow \forall b \in B\ h{\downarrow_A} \neq b \qquad \models S \Rightarrow \exists n \forall b_1 \ldots b_n \in B^* \ \neg \exists F\ S\{h\, b_1 \ldots b_n/h\} \qquad \Gamma \vdash P \textbf{ sat } S}{\Gamma \vdash P \backslash B \textbf{ sat } T},$$
where $\alpha(T) = \alpha(S) - B$ and $\alpha(S) = A$

(Recursion)
$$\frac{\Gamma,\ X \textbf{ sat } S \vdash P \textbf{ sat } S}{\Gamma \vdash \mu X.P \textbf{ sat } S},$$
where $\mu X.P$ is communication guarded and $\alpha(S) = \alpha(X)$

(Assumption)
$$\Gamma,\ X \textbf{ sat } S \vdash X \textbf{ sat } S, \quad \text{where } \alpha(S) = \alpha(X)$$

$\varphi_+(S,T)$, $\varphi_{\parallel}(S,T)$ and $\varphi_{[\,]}(S,\bar{b},\bar{a})$ abbreviate formulas that appear also in Olderog's proof system for $\equiv>$ and are defined below

$$\begin{aligned}
\varphi_+(S,T) &\equiv (h{\downarrow_A} = \varepsilon \Rightarrow S \wedge T) \wedge (h{\downarrow_A} \neq \varepsilon \Rightarrow S \vee T), \quad \text{where } A = \alpha(S) = \alpha(T) \\
\varphi_{\parallel}(S,T) &\equiv \exists G\, \exists H\, (S\{G/F\} \wedge T\{H/F\} \wedge G \cap H \subseteq F \wedge (G \cup H) - (\alpha(S) \cap \alpha(T)) \subseteq F) \\
\varphi_{[\,]}(S,\bar{b},\bar{a}) &\equiv \exists t\, \exists G\, (S\{t/h, G/F\} \wedge t[\bar{b}/\bar{a}] = h{\downarrow_A} \wedge G[\bar{b}/\bar{a}] \subseteq F), \quad \text{where } A = \alpha(S)\{\bar{b}/\bar{a}\}
\end{aligned}$$

Projection operator $\downarrow_A$ and communication guardedness are defined as in [5], ε is an empty trace.

Table 1.

The theorems below relate the proposed proof system for **sat** to Olderog's proof system for $\equiv>$. It will follow that the special case of (4) with empty Γ coincides with Olderog's "$P \equiv> S$ is derivable".

To state the theorems we need the following auxiliary notation: for a process term P and $\Gamma = \{X_1 \textbf{ sat } S_1, \ldots, X_n \textbf{ sat } S_n\}$, by $P\{\Gamma\}$ we shall denote the mixed term $P\{S_1/X_1, \ldots, S_n/X_n\}$, obtained from P by replacing free occurrences of variables $X_1, \ldots, X_n$ by specifications $S_1, \ldots, S_n$. We will also write $\vdash P \equiv> S$ to denote that $P \equiv> S$ is derivable in Olderog's proof system.

Theorem 2. *If* $\Gamma \vdash P \textbf{ sat } S$ *then* $\vdash P\{\Gamma\} \equiv> S$.

Proof. Follows by by induction on the derivation of $\Gamma \vdash P \textbf{ sat } S$.

Corollary 3. *Proof system for* **sat** *is sound, that is,*

$$\textit{if} \quad \vdash P \textbf{ sat } S \quad \textit{then} \quad \mathcal{R}^*[\![P]\!] \sqsupseteq \mathcal{R}^*[\![S]\!]$$

Proof. Follows from Theorem 2 and soundness of Olderog's proof system for $\equiv>$.

Theorem 4. *Let P be a process and Γ a set of assumptions such that $P\{\Gamma\}$ is a closed term. If* $\vdash P\{\Gamma\} \equiv> S$ *for a specification S then* $\Gamma \vdash P \textbf{ sat } S$.

Proof. The proof is long and omitted (see [7]).

We end this section with the following useful observation, which can be proved by straightforward induction on the derivation of $\Gamma \vdash P \textbf{ sat } S$

Proposition 5. *If for a recursive term P a judgement* $\Gamma \vdash P \textbf{ sat } S$ *can be derived for some S and Γ, then $P \in Proc$, that is, P satisfies the requirements of Definition 1.*

4 Parametric Processes and Specifications

The triple

$$(\mathit{CProc}, \mathit{Spec}, \textbf{sat}) \tag{5}$$

can be seen as a specification system that does not support parametric constructions, in the sense of this notions explained in Introduction. We lift (5) to the parametric case by extending the process algebra with abstraction and application primitives, in a manner resembling the typed lambda calculus, where types are replaced with specifications.

First we introduce a syntactic class *PSpec* of *parametric specifications* ranged over by σ and defined by the following abstract syntax:

$$\sigma ::= S \mid \sigma \to \sigma$$

Intuitively, specification $\sigma_1 \to \sigma_2$ describes a set of parametric processes, each of which when supplied with an argument, a process that satisfies (possibly parametric) specification σ_1, yields a process that satisfies specification σ_2.

The syntax of recursive terms is extended with two new clauses

$$P ::= \ \ldots \mid \lambda X : \sigma \,.\, P \mid PP$$

The set of *parametric processes* will be defined as a subset of so extended recursive terms. In the case of nonparametric processes, syntactic restrictions enumerated in Definition 1 were used to distinguish the set *Proc* of processes in the set of (nonparametric) recursive terms. However, as it has been noted in Proposition 5, these restrictions are encoded also in proof system for **sat**. In the parametric case, it is more convenient to adopt a counterpart of Proposition 5 as a definition rather than separately formulate syntactic restrictions that define the desired set of parametric processes. This is the approach we are going to take, but first we need to refine some notions.

So far the alphabets of processes and specifications were just finite subsets of *Comm*, the set of communications. Alphabets of parametric processes (and specifications), we shall call them *hyper-alphabets* in the sequel, have to be more complex objects, indicating the alphabets of possible arguments of a process. We define the set $\mathcal{H}$ of all hyper-alphabets as follows:

$$\begin{array}{ll} \mathcal{H}_0 & = \text{set of finite subsets of } \mathit{Comm} \\ \mathcal{H}_{n+1} & = \{(A, B) \mid A, B \in \mathcal{H}_n\} \cup \mathcal{H}_n \\ \mathcal{H} & = \bigcup_{n=0}^{\infty} \mathcal{H}_n \end{array}$$

The hyper-alphabet $\alpha^*(\sigma)$ of a specification σ is defined recursively by the following clauses:

$$\begin{array}{ll} \alpha^*(\sigma) = \alpha(S), & \text{if } \sigma = S \\ \alpha^*(\sigma) = (\alpha^*(\sigma_1), \alpha^*(\sigma_2)), & \text{if } \sigma = \sigma_1 \to \sigma_2 \end{array}$$

The hyper-alphabet $\alpha^*(P)$ of a recursive term P is defined by the clauses given below. If it is not possible to associate an alphabet with a term using these defining clauses, $\alpha^*(P)$ is considered to be undefined. Just as in the non-parametric case we assume that every process variable X has an associated hyper-alphabet $\alpha^*(X)$.

$$\begin{array}{lll} \alpha^*(\mathit{stop}_A) & = A & \\ \alpha^*(a \,.\, P) & = \alpha^*(P) \cup \{a\} & \text{if } \alpha^*(P) \in \mathcal{H}_0 \\ \alpha^*(P + Q) & = \alpha^*(P) \cup \alpha^*(Q) & \text{if } \alpha^*(P), \alpha^*(Q) \in \mathcal{H}_0 \\ \alpha^*(P \| Q) & = \alpha^*(P) \cup \alpha^*(Q) & \text{if } \alpha^*(P), \alpha^*(Q) \in \mathcal{H}_0 \\ \alpha^*(P[b/a]) & = (\alpha^*(P) - \{a\}) \cup \{b\} & \text{if } \alpha^*(P) \in \mathcal{H}_0 \\ \alpha^*(P \backslash a) & = (\alpha^*(P) - \{a\}) & \text{if } \alpha^*(P) \in \mathcal{H}_0 \\ \alpha^*(\mu X \,.\, P) & = \alpha^*(P) \cup \alpha^*(X) & \text{if } \alpha^*(P) \in \mathcal{H}_0 \\ \alpha^*(\lambda X : \sigma \,.\, P) & = (\alpha^*(X), \alpha^*(P)) & \text{if } \alpha^*(X) = \alpha^*(\sigma) \\ \alpha^*(PQ) & = B & \text{if } \alpha^*(P) = (A, B) \text{ and } \alpha^*(Q) = A \end{array}$$

Note that $\alpha^*(P) = \alpha(P)$ for $P \in \mathit{Proc}$ and that alphabets of terms such as $(\lambda X : \sigma.P) + Q$, where a process algebra operator $+$ is applied to a higher order term, are not defined.

In the sequel we will consider only those recursive terms, whose hyper-alphabets are defined, and the notion of recursive terms will refer to such terms from now on. The set of so understood recursive terms can be stratified into two subsets: the subset of terms whose alphabet is in $\mathcal{H}_0$ and to which process constructors $+$, $a.$, $\|$, $\backslash$, $[\,]$ and μ can be applied, and the subset of higher order terms, on which only abstractions and applications can be performed. Note however, that the hyper-alphabet of an application PQ, where P is a higher order term, can belong to $\mathcal{H}_0$ so process algebra operators can be applied to such PQ.

The syntactic notion of communication guardedness that appears as a side condition for Recursion rule of Table 1 also needs to be redefined. We extend Olderog's definition of communication guardedness by requiring that for every recursive subterm $\mu X.P$ of a communication guarded term no free occurrence of X in P lies within an application RQ, thus restricting the interference between process recursion and parametricity.

Now we can formulate the proof system for the satisfaction relation **sat** extended to the parametric case. The judgements will have the form

$$\Gamma \;\vdash\; P \textbf{ sat } \sigma$$

where P is a (parametric) recursive term and the set of assumptions Γ is extended in the obvious way to contain assumptions of the form X **sat** σ.

The proof system contains all the rules collected in Table 1, which were used for nonparametric processes, but this time we assume that S and T range over nonparametric specifications, P and Q range over parametric recursive terms, Γ is extended as explained above, and the side conditions concerning alphabets refer to hyper-alphabets, i.e. α is replaced with α^*. The side condition of communication guardedness in Recursion rule refers now to the modified version of this notion.

Moreover, the proof system of Table 1 is extended with the following rules, which are essentially λ-introduction and λ-elimination rules of the proof system for Church typed λ calculus (see for example [1]), plus a modified Assumption rule, which replaces the Assumption rule from Table 1.

$$\text{(Assumption)} \qquad \Gamma,\; X \textbf{ sat } \sigma \vdash X \textbf{ sat } \sigma\,, \qquad \text{where } \alpha^*(X) = \alpha^*(\sigma)$$

$$\text{(Abstraction)} \qquad \frac{\Gamma,\; X \textbf{ sat } \sigma_1 \;\vdash\; P \textbf{ sat } \sigma_2}{\Gamma \;\vdash\; \lambda X : \sigma_1 \,.\, P \textbf{ sat } \sigma_1 \to \sigma_2}\,, \qquad \text{where } \alpha^*(X) = \alpha^*(\sigma_1)$$

$$\text{(Application)} \qquad \frac{\Gamma \;\vdash\; P \textbf{ sat } \sigma_1 \to \sigma_2 \qquad\qquad \Gamma \;\vdash\; Q \textbf{ sat } \sigma_1}{\Gamma \;\vdash\; PQ \textbf{ sat } \sigma_2}$$

Definition 6. The set of *parametric processes PProc* consists of those recursive terms P, for which $\Gamma \;\vdash\; P$ **sat** σ can be derived for some Γ and σ. *CPProc*, the set of *closed parametric processes*, is defined as the set of those parametric processes that have no free variables.

Note that the above definitions and Proposition 5 imply that $Proc \subseteq PProc$ and $CProc \subseteq CPProc$.

We have just defined all components of a parametric specification system

$$(CPProc, PSpec, \mathbf{sat})$$

In this report we do not provide semantical account of parametric processes. This can be done by extending the readiness denotational semantics so that parametric processes are interpreted as process-valued functions on processes. Such a construction, which is an adaptation of a general procedure described in [8], will be reported separately together with soundness proof of the proof system for **sat** . Here, Theorem 7 below, provides syntactic arguments indicating that the proposed formalism is sound. In Theorem 7, we use the arrow $\xrightarrow{\beta^*}$ to denote β-reducibility in a context and in possibly many steps, where the notion of β-reduction is defined in a standard manner: $(\lambda X : \sigma \,.\, P)\, Q \xrightarrow{\beta} P\{Q/X\}$.

Theorem 7.
(a) If $\vdash P \,\mathbf{sat}\, S$ *then there exists* $P' \in Proc$ *such that* $P \xrightarrow{\beta^*} P'$*. For every such* P' $\mathcal{R}^*[\![P']\!] \sqsupseteq \mathcal{R}^*[\![S]\!]$ *, i.e.* $\vdash P' \,\mathbf{sat}\, S$ *is justified semantically.*
(b) If $\vdash P \,\mathbf{sat}\, \sigma_1 \rightarrow \sigma_2$ *then for every* Q *, whenever* $\vdash Q \,\mathbf{sat}\, \sigma_1$ *then* $\vdash PQ \,\mathbf{sat}\, \sigma_2$*.*

Point (a) of Theorem 7 ensures that if a parametric process P can be proved to satisfy a nonparametric specification S then, up to β equivalence, this assertion is semantically justified. Point (b) relates satisfaction of higher order terms to lower order ones, which is a natural requirement if parametric processes are interpreted as functions.

The next proposition shows that there is an exact match between the proof systems for $\equiv>$ on mixed terms and the proof system for **sat** on a restricted class of processes of the form

$$\lambda X_1 : S_1 \ldots \lambda X_n : S_n \,.\, P \tag{6}$$

This class of parametric processes with no higher-order parameters corresponds to mixed terms of [5] as (6) can be syntactically translated into a mixed term $P\{S_1/X_1, \ldots, S_n/X_n\}$ and, conversely, a mixed term P can be translated into a process having the form (6) by replacing every occurrence of any specification S_i appearing in P with a fresh variable X_i.

Proposition 8. $\vdash \lambda X_1{:}S_1 \ldots \lambda X_n{:}S_n \,.\, P \,\mathbf{sat}\, S_1 \rightarrow \cdots \rightarrow S_n \rightarrow S$ *if and only if* $\vdash P\{S_1/X_1, \ldots, S_n/X_n\} \equiv> S$.

Proof. It can be shown by induction that

$$\vdash \lambda X_1 : S_1 \ldots \lambda X_n : S_n \,.\, P \,\mathbf{sat}\, S_1 \rightarrow \cdots \rightarrow S_n \rightarrow S$$

if and only if

$$S_1 \,\mathbf{sat}\, X_1, \ldots, S_n \,\mathbf{sat}\, X_n \vdash P \,\mathbf{sat}\, S.$$

By Theorems 2 and 4 this is equivalent to $\vdash P\{S_1/X_1, \ldots, S_n/X_n\} \equiv> S$.

5 Examples

Now we are in a position to fill in the details of the example that was sketched in Introduction. Part of the work has been already done in Examples 1 and 2. The processes $double_1$ and cnt_2 can be defined as follows:

$$\begin{aligned} double_1 &\equiv \lambda X : CNT_1 \,.\, (X[link/dn] \parallel X[link/up]) \backslash link \\ cnt_2 &\equiv double_1 \; cnt_1 \end{aligned}$$

Using the proof system for parametric processes we may:

(i) derive the satisfaction assertion $\vdash cnt_1$ **sat** CNT_1,
(ii) independently derive the assertion $\vdash double_1$ **sat** $CNT_1 \to CNT_2$.

By virtue of Application rule, these imply $\vdash cnt_2$ **sat** CNT_2.

As another example we give a process qdr_1 which can be shown to satisfy (3).

$$qdr_1 \;\equiv\; \lambda Y{:}CNT_1 \to CNT_2 \,.\, \lambda X{:}CNT_1 \,.\, ((Y X)[link/up] \parallel (Y X)[link/dn]) \backslash link$$

A larger example of digital gates modelling, is developed in [7].

Acknowledgements. We would like to thank Thorsten Altenkirch, Kārlis Černās and Uno Holmer for helpful comments on this work.

References

1. H.P. Barendregt, *Lambda Calculi with Types*, in: Handbook of Logic in Computer Science, vol. 2 (S. Abramsky, Dov M. Gabbay, T. S. E. Maibaum, Eds.) pp. 117–309, Calderon Press, Oxford 1992.
2. S. Holmström, A refinement calculus for specification in Hennessy-Milner logic with recursion, *Formal Aspects of Computing*, vol. 1 (3). pp. 242–272, 1989.
3. K. Larsen and R. Milner, A Complete Protocol Verification using Relativised Bisimulations, R 86-12, Institute of Electrical Systems, Aalborg University Centre, 1986.
4. R. Milner, *Communication and Concurency*, Prentice Hall, 1989.
5. E.-R. Olderog, *Nets, Terms and Formulas*, Cambridge University Press, Cambridge 1991.
6. E.-R. Olderog, Towards a design calculus for communicating programs, in: *Proc. CONCUR '91*, LNCS, Springer, 1991.
7. W. Pawłowski, P. Pączkowski, S. Sokołowski, A specification system for parametric concurrent processes, electronically available as `ftp://ipipan.gda.pl/pub/local_users/Pawlowski_W./parametric.ps.gz`.
8. S. Sokołowski, Requirement specifications and their realizations: toward a unified framework. Submitted to *Theoretical Computer Science*. A preliminary version has appeared as: The GDM approach to specifications and their realizations. Part I: Specification systems. ICS PAS Report 797, Warszawa 1995.

This article was typeset using the LaTeX macro package with the LLNCS2E class.