Specifying and Verifying Parametric Processes

Wiesław Pawłowski* Paweł Pączkowski† Stefan Sokołowski*

December 1994

Abstract

A framework in which processes parametrized with other processes can be specified, defined and verified is introduced. 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. The decomposition principle, when formalized in terms of a *specification system*, by which, following [Sokołowski 94], we mean a triple

$$(Constr, Spec, \mathbf{sat})$$

consisting of a set of constructions *Constr* (programs, processes, etc.), a set of specifications *Spec*, and satisfaction relation **sat** relating constructions to specifications, amounts to the requirement that a task of providing a construction c that satisfies a given specification S (denoted c **sat** S) can be decomposed as follows:

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 c that c **sat** S.

In [Sokołowski 94], a point of view was taken, that the method mentioned in point 2 should be a *construction* itself, i.e. belong to *Constr*. Adopting this point of view we can rephrase 2 as

*Institute of Computer Science, Polish Academy of Sciences, ul. Abrahama 18, 81-825 Sopot, Poland, e-mail: {W.Pawlowski, S.Sokolowski}@IPIPAN.Gda.PL; supported by ICS PAN, KBN grant PB-1312/P3/92/02 and CRIT IC 1010/II.

†Department of Computing Science, Chalmers University of Technology, 412-96 Göteborg, Sweden, and Institute of Mathematics, University of Gdańsk, ul. Wita Stwosza 57, 80-952 Gdańsk, Poland, e-mail: pawel@cs.chalmers.se, matpmp@halina.univ.gda.pl; supported by KBN grant PB-1312/P3/92/02, CRIT IC 1010/II and 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 which 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 c we were looking for.

When the sub-constructions are again decomposed into smaller bits the parameters become parametric themselves and we have to deal with higher order parameters. In [Sokołowski 94] a more elaborate decomposition schema than the one presented above is considered: dependent products rather than functional specifications are allowed, but here we will confine ourselves to a simple setup.

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 logic for specifying processes proposed in [Olderog 91]. Olderog extends a process algebra by introducing mixed terms, where a specification can become a part of a process term. This enables transformation based development of processes [Olderog 91, Olderog 91a]. We consider a different option and introduce parametric processes that contain process variables. We allow higher order parametricity: process variables can represent also parametric processes. Technically speaking, Olderogs'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 "a (parametric) process meets a specification". To this end, we needed to reformulate the proof system for nonparametric processes that was given in [Olderog 91]. 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 [Olderog 91]. The direct proof system is extended to the parametric case.

The concept of parametric processes (but only first order ones) has already appeared in [Milner 89]. 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. [Larsen and Milner 86]).

The formalism we develop allows decomposition steps of the form 1 and $2'$ that involve higher-order processes as 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 using transformation [Olderog 91] or refinement [Holmström 89] techniques.

The paper is organized as follows. In Section 2 we present a process algebra of nonparametric processes and a logic for specifying them that appear in

[Olderog 91]. 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 an example.

2 Nonparametric processes and specifications

The class of processes that we consider and a logic for specifying their properties are taken from [Olderog 91]. We briefly present the main notions, referring to [Olderog 91] for a comprehensive exposition.

2.1 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 *actions*.

In order to define a syntactic representation of processes we start with a description of the set of *recursive terms Rec*. Let $\mathcal{X}$ be a (countable) set of (process) *identifiers*. We assume that $\mathcal{X}$ is partitioned into sets $\mathcal{X}_A$ of *identifiers with alphabet* A, where $A \subseteq Comm$ is finite. In the definition below P and Q range over recursive terms, X ranges over identifiers, and A over finite sets of communications. The set *Rec* is defined as follows:

$$
\begin{array}{lcll}
P & ::= & \mathrm{stop}_A & \text{(deadlock)} \\
 & | & a.P & \text{(prefix)} \\
 & | & P + Q & \text{(choice)} \\
 & | & P \parallel Q & \text{(parallel composition)} \\
 & | & P[b/a] & \text{(renaming)} \\
 & | & P\backslash a & \text{(hiding)} \\
 & | & X & \text{(identifier)} \\
 & | & \mu X.P & \text{(recursion)}
\end{array}
$$

Iterated applications of renaming and hiding will be abbreviated as follows:

- $P[\bar{b}/\bar{a}]$ will stand for $P[b_1/a_1]\ldots[b_n/a_n]$, where $\bar{b} = b_1, \ldots, b_n, \bar{a} = a_1, \ldots, a_n$
- $P\backslash A$ will denote $P\backslash a_1 \ldots \backslash a_n$, where $A = \{a_1, \ldots a_n\}$.

Definition 1 A recursive term P is *communication guarded* if there exists a set of communications A such that for every recursive subterm $\mu X.Q$ of P, for every free occurrence of X in Q

(1) X lies within a subterm of Q of the form $a.R$

(2) X does not lie within a subterm of Q of the form $R[b/a]$ or $R\backslash a$, where $a \in A$ and $b \notin A$.

If the condition (2) above is dropped, we say that P is *action guarded* (this notion is usually given a separate, slightly simpler definition).

To define process terms one more notion is needed — a notion of a *(communication) alphabet*. An alphabet of a recursive term P (denoted by $\alpha(P)$) is a set of communications defined inductively as follows:

$$\begin{array}{lcl}
\alpha(\mathrm{stop}_A) & = & A \\
\alpha(a.P) & = & \{a\} \cup \alpha(P) \\
\alpha(P + Q) & = & \alpha(P) \cup \alpha(Q) \\
\alpha(P \parallel Q) & = & \alpha(P) \cup \alpha(Q) \\
\alpha(P[b/a]) & = & (\alpha(P) - \{a\}) \cup \{b\} \\
\alpha(P\backslash a) & = & (\alpha(P) - \{a\}) \\
\alpha(X) & = & A \text{ if } X \in \mathcal{X}_A \\
\alpha(\mu X.P) & = & \alpha(X) \cup \alpha(P)
\end{array}$$

Definition 2 We say that a recursive term P is a *process term* (or simply, a process) if it satisfies the following conditions:

- P is action guarded
- every subterm $a.Q$ of P satisfies $a \in \alpha(Q)$
- every subterm $Q + R$ of P satisfies $\alpha(Q) = \alpha(R)$
- 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*.

The intuitive meaning of process terms is as follows:

- stop_A denotes a process which neither engages in any communication nor in any internal action.
- $a.P$ denotes a process which first communicates a and then behaves like P.
- $P + Q$ denotes a process which behaves like P or like Q depending on whether its first action was an action of P or of Q. If the first action could be performed by P as well as by Q the choice made is nondeterministic.
- $P \parallel Q$ denotes a parallel composition of P and Q which behaves as P and Q working independently except for synchronization on actions belonging to the intersection of their communication alphabets.
- $P[b/a]$ denotes a process which behaves like P, but with all actions a changed to b.
- $P\backslash a$ denotes a process which behaves like P, but with all actions a hidden, i.e. changed to τ.
- $\mu X.P$ behaves like P, but with every occurrence of X inside P denoting a "recursive call" to $\mu X.P$.

The above intuition can be made precise in several different ways. One possible choice is so called *readiness semantics*. It describes process behaviour in terms of pairs $(tr, \mathcal{F})$ where tr is finite sequence of communications called *trace*, and $\mathcal{F}$, called *ready set*, is a set of actions in which a process is ready to engage after "performing" the trace tr. Besides the ready pairs the semantic domain contains a distinguished element τ representing initial *unstability* of a process, and "divergent" ready pairs of the form $(tr, \uparrow)$ which represent behaviours where a process after performing trace tr enters an infinite loop.

Thus, the set of *process information* over A is defined as:

$$\mathrm{Info}_{\mathcal{R},A} = \{\tau\} \cup A^* \times \mathcal{P}(A) \cup A^* \times \{\uparrow\}$$

and the semantic domain of the readiness semantics is defined as follows:

$$\mathrm{DOM}_{\mathcal{R}} = \bigcup_{A \subseteq Comm} \mathrm{DOM}_{\mathcal{R},A}$$

where A is a finite subset of $Comm$ and

$$\mathrm{DOM}_{\mathcal{R},A} = \{(A, \Gamma) | \Gamma \subseteq \mathrm{Info}_{\mathcal{R},A}\}.$$

The readiness semantics for closed process terms is a function:

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

which can be given an elegant denotational definition.

There is a natural partial ordering on the semantic domain $\mathrm{DOM}_{\mathcal{R}}$:

$$(A_1, R_1) \sqsubseteq (A_2, R_2) \text{ if } A_1 = A_2 \text{ and } R_1 \supseteq R_2$$

Intuitively, if $\mathcal{R}^*[\![P]\!] \sqsubseteq \mathcal{R}^*[\![Q]\!]$ then the process denoted by Q is more *controllable* than the one denoted by P, i.e. it is more deterministic, less divergent and more stable.

2.2 Readiness specifications

Processes are specified by formulas of many-sorted predicate logic, called *readiness logic*, which has sorts of communications, traces (finite sequences of communications), ready sets (finite subsets of communications), natural numbers and logical values. In this report we give just a minimal necessary information on readiness logic. For our purposes it is sufficient to remark that readiness logic allows one to define sets of ready pairs in a direct manner and 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 set variable F are free, and which satisfies some natural syntactic restrictions that guarantee a monotonicity property explained below. Since a specification has just two free variables h and F, the notion of truth of a specification with respect to the evaluation $(tr, \mathcal{F})$ of h and F can be defined and will be denoted

$$(tr, \mathcal{F}) \models S \tag{1}$$

If (1) holds we say that S is satisfied by a ready pair $(tr, \mathcal{F})$. Thus, a specification S determines a set of ready pairs which satisfy it.

Moreover, with every specification S its alphabet, denoted by $\alpha(S)$, can be associated. This is a consequence of yet another syntactic restriction on readiness formulas that every occurrence of the free variable h lies in the scope of *projection* $\downarrow_A$ onto some finite set of communications A. What it means semantically is that S defines a set of ready pairs $(tr, \mathcal{F})$, where only the projection of tr onto $\alpha(S)$ is defined and the actions outside $\alpha(S)$ do not affect satisfiability of S by $(tr, \mathcal{F})$.

The *monotonicity* property of readiness specifications that was mentioned above is defined as follows: for an arbitrary readiness specification S and for all traces tr and ready sets $\mathcal{F}$ and $\mathcal{G}$

$$(tr, \mathcal{F}) \models S \text{ and } \mathcal{F} \subseteq \mathcal{G} \text{ imply } (tr, \mathcal{G}) \models S$$

The set of specifications will be denoted by *Spec*. The *readiness semantics of readiness specifications* is defined as follows:

$$\mathcal{R}^*[\![S]\!] = (\alpha(S), \{(tr, \mathcal{F}) | tr \in \alpha(S)^* \wedge \mathcal{F} \subseteq \alpha(S) \wedge (tr, \mathcal{F}) \models S\}).$$

2.3 Satisfaction relation

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

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

Intuitively, it means that

- The process P and the specification S have the same alphabets i.e. $\alpha(P) = \alpha(S)$.
- *Safety condition.* If P may engage in a trace tr then there must exist a ready set $\mathcal{F}$ such that $(tr, \mathcal{F}) \models S$.
- *Liveness condition.* After performing a trace tr the process P must engage in one of the ready sets $\mathcal{F}$ such that $(tr, \mathcal{F}) \models S$.
- *Stability condition.* The process P is stable immediately.

2.4 Proof system for sat via mixed terms

The inference system for **sat** defined in [Olderog 91] makes use of a notion of *mixed terms*. Mixed terms are, as the name suggests, a mixture of readiness specifications and process terms and are formally defined by allowing specifications in recursive terms

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

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

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

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 [Olderog 91] 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. The inference rules of the proof system for $\equiv>$ are presented in Table 1. In these rules, P, Q, R range over closed mixed terms, S and T — over specifications. Curly braces are used to denote syntactic substitutions.

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 2 we propose such a proof system, whose rules bear a clear resemblance to Olderog's proof system for $\equiv>$. (From now on, the symbol $\equiv>$ will refer to the syntactic notion of derivability in the proof system of Table 1.)

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

$$\Gamma \vdash P \textbf{ sat } S, \tag{2}$$

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

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

The sets of assumptions will be needed to handle recursion. The special case of (2), with $\Gamma = \emptyset$ will be shown to coincide with Olderog's notion of $P \equiv> S$.

The theorems below show the relative soundness and completeness of the proposed proof system for **sat** with respect to the proof system for $\equiv>$.

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 a 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 $S_1, \ldots, S_n$.

To distinguish the rules of the proof systems for $\equiv>$ and for **sat**, we add index $_{\equiv>}$ to the names of the respective rules of the proof system for $\equiv>$.

Theorem 3 (Relative soundness of proof system for **sat**) *If* $\Gamma \vdash P$ **sat** S *then* $P\{\Gamma\} \equiv> S$.

Proof. Follows by by induction on the length of the derivation of $\Gamma \vdash X$ **sat** S. Note, that some of the assumptions of proof rules in Table 1 are replaced by different but equivalent versions in Table 2. Such equivalent pairs of assumptions are collected in Lemma 4 below. □

(Reflexivity) $P \equiv\!> P$

(Transitivity) $$\frac{P \equiv\!> Q \qquad Q \equiv\!> R}{P \equiv\!> R}$$

(Context) $$\frac{Q \equiv\!> R}{M\{Q/X\} \equiv\!> M\{R/X\}}$$
where M is a mixed term, $\mathrm{M}\{Q/X\}$ and $\mathrm{M}\{R/X\}$ are closed mixed terms

(Consequence) $$\frac{\models S \;\Rightarrow\; T}{P \equiv\!> T}\,, \qquad \text{where } \alpha(S) = \alpha(T)$$

(Deadlock) $stop_A \equiv\!> h{\downarrow_A} \leq \varepsilon$

(Prefix) $$\frac{(\varepsilon, \{a\}) \models S}{a.S\{a.h/h\} \equiv\!> S}\,, \qquad \text{where } a \in \alpha(S)$$

(Choice) $S + T \equiv\!> \varphi_+(S,T)$
where $\varphi_+(S,T) \equiv (h{\downarrow_A} = \varepsilon \;\Rightarrow\; S \wedge T) \;\wedge\; (h{\downarrow_A} \neq \varepsilon \;\Rightarrow\; S \vee T)$
and $\alpha(S) = \alpha(T)$

(Parallel) $S \parallel T \equiv\!> \varphi_{\parallel}(S,T)$
where $\varphi_{\parallel}(S,T) \equiv \exists G\; \exists H\; (S\{G/F\} \wedge T\{H/F\} \wedge$
$G \cap H \subseteq F \wedge (G \cup H) - A \subseteq F)$
and $A = \alpha(S) = \alpha(T)$

(Renaming) $S[\bar{b}/\bar{a}] \equiv\!> \varphi_{[\,]}(S, \bar{b}, \bar{a})$
where $\varphi_{[\,]}(S, \bar{b}, \bar{a}) \equiv \exists t\; \exists G\; (S\{t/h, G/F\} \wedge t[\bar{b}/\bar{a}] = h{\downarrow_A} \wedge$
$G[\bar{b}/\bar{a}] \subseteq F)$
and $A = \alpha(S)\{\bar{b}/\bar{a}\}$

(Hiding) $$\frac{\begin{array}{l}\models S\{F - B/F\} \;\Rightarrow\; T \\ \forall b \in B\;\; b \models \neg\exists F\; S \\ \forall tr \in \alpha(S)^*\;\; \exists n \geq 0\; \forall b_1, \ldots, b_n \in B\;\; tr\, b_1 \ldots b_n \models \neg\exists F\; S\end{array}}{S \setminus B \equiv\!> T}$$
where $\alpha(T) = \alpha(S) - B$ and $\alpha(S) = A$

(Recursion) $$\frac{P\{S/X\} \equiv\!> S}{\mu X.P \equiv\!> S}\,,$$
where $\mu X.P$ is a communication guarded closed mixed term
and $\alpha(S) = \alpha(X)$

Table 1.

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

(Deadlock)
$$\Gamma \vdash stop_A\ \mathbf{sat}\ h{\downarrow_A} \leq \varepsilon$$

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

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

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

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

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

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

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

Table 2.

Lemma 4

(a) $\models S\{\varepsilon/h, \{a\}/F\}$ *if and only if* $(\varepsilon, \{a\}) \models S$

(b) $\models S \;\Rightarrow\; \forall b \in B\ h{\downarrow_A} \neq b$ *if and only if* $\forall b \in B\ b \models \neg\exists F\ S$

(c) $\models \exists n \forall b_1 \ldots b_n \in B^*\ \neg\exists F\ S\{hb_1 \ldots b_n/h\}$
if and only if
$\forall tr \in \alpha(S)^*\ \exists n \geq 0\ \forall b_1, \ldots, b_n \in B\ trb_1 \ldots b_n \models \neg\exists F\ S$

Corollary 5 *Proof system for* **sat** *is sound, that is,*

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

Proof. Follows from Theorem 3 and soundness of proof system for $\equiv>$. □

The converse of Theorem 3 also holds.

Theorem 6 (Relative Completeness) *Let P be a process and Γ a set of assumptions such that $P\{\Gamma\}$ is a closed term. If $P\{\Gamma\} \equiv> S$ for a specification S then $\Gamma \vdash P$ **sat** S.*

Proof. The proof is long and is omitted due to space limitations. □

We end this section with the following useful observation

Proposition 7 *If for a recursive term P a judgement $\Gamma \vdash P$ **sat** S can be derived for some S and Γ, then $P \in Proc$, that is, P satisfies the restrictions of Definition 2.*

Proof. Follows by induction on the derivation of $\Gamma \vdash P$ **sat** S. □

4 Parametric Processes and Specifications

The triple

$$(CProc, Spec, \mathbf{sat})$$

can be viewed as a specification system, in the sense of [Sokołowski 94]. We lift it 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 given a process satisfying (possibly parametric) specification σ_1 as an argument yields a process that satisfies specification σ_2.

The syntax of recursive terms is extended by adding two new clauses

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

We consider the recursive terms up to α conversion, where λ as well as μ are variable binding operators.

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 2 were used to distinguish the set *Proc* of processes in the set of (nonparametric) recursive terms. However, as it has been noted in Proposition 7, these restrictions are encoded also in proof system for **sat**. In the parametric case, it is more convenient to adopt a counterpart of Proposition 7 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.

The notions of alphabet of a process and alphabet of a specification have to be extended to the parametric case. So far the alphabets 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:

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

Now, 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 nonparametric case we assume that every process variable X has an associated hyper-alphabet $\alpha^*(X)$.

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

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 $+$ s 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 an application PQ, where P is a higher order term, can belong to the former subset and process algebra operators can be applied to it.

The syntactic notion of communication guardedness also needs to be re-defined. Apart from conditions (1) and (2), we add the following point to Definition 1:

(3) X does not lie within a subterm of Q of the form RQ

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

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

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 2, 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 2 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 [Barendregt 92]), plus a modified Assumption rule, which replaces the Assumption rule from Table 2.

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

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

$$\text{(Application)} \qquad \frac{\Gamma \;\vdash\; P \;\mathbf{sat}\; \sigma \to \rho \qquad \Gamma \;\vdash\; Q \;\mathbf{sat}\; \sigma}{\Gamma \;\vdash\; PQ \;\mathbf{sat}\; \rho}$$

Definition 8 The set of *parametric processes PProc* consists of those recursive terms P, for which $\Gamma \;\vdash\; P \;\mathbf{sat}\; \sigma$ 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 7 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 [Sokołowski 94], will be reported separately together with soundness proof of the proof system for **sat** . Here, we just show that the proof system respects β reductions.

Since parametric processes can be seen as lambda terms built on top of non-parametric processes, the notion of β-reduction can be defined in a standard manner:

$$(\lambda X : \sigma \,.\, P)\, Q \xrightarrow{\beta} P\{Q/X\}$$

We will use the same arrow $\xrightarrow{\beta}$ to denote β-reducibility in possibly many steps.

The next proposition ensures that **sat** is preserved by β reductions.

Proposition 9 *If* $\Gamma \vdash P$ **sat** σ *and* $P \xrightarrow{\beta} P'$ *then* $\Gamma \vdash P'$ **sat** σ.

Proof. By induction on the derivation of $\Gamma \vdash P$ **sat** σ. □

Corollary 10 *If* $P \in PProc$ *and* $P \xrightarrow{\beta} P'$ *then* $P' \in PProc$.

In [Sokołowski 94] a special case of *shallow* parametricity is distinguished. A shallow n-parameter process has the form

$$\lambda X_1 : S_1 \ldots \lambda X_n : S_n \,.\, P \tag{3}$$

This class of parametric processes corresponds to mixed terms of [Olderog 91]. The process term shown above can be syntactically translated into a mixed term

$$P\{S_1/X_1\,, \ldots,\, S_n/X_n\}$$

Conversely, a mixed term P can be translated into a process having the form (3) by replacing every occurrence of any specification S_i appearing in P with a fresh variable X_i.

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

Proposition 11 $\vdash \lambda X_1 : S_1 \ldots \lambda X_n : S_n \,.\, P$ **sat** $S_1 \to \cdots \to S_n \to S$ *if and only if* $P\{S_1/X_1\,, \ldots,\, S_n/X_n\} \equiv> S$.

Proof. It can be show by induction that

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

if and only if

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

By Theorems 3 and 6 this is equivalent to $P\{S_1/X_1\,, \ldots,\, S_n/X_n\} \equiv> S$. □

5 Example

We present here small example to illustrate the use of the proposed formalism: a task of constructing a counter of capacity 2. *CNT2* below is a readiness specification of such a counter. Upon communication *up* (*dn*) the value stored in the counter is increased (decreased) by 1. The symbol $up\#h$ denotes the number of *up* communications in trace h.

$$\begin{array}{rcl} CNT2 & \equiv & (0 \leq up\#h - dn\#h \leq 2) \wedge (up\#h - dn\#h < 2 \;\Rightarrow\; up \in F) \wedge \\ & & (up\#h - dn\#h > 0 \;\Rightarrow\; dn \in F) \end{array}$$

The construction of process *cnt2* such that

$$\vdash \; cnt2 \;\textbf{sat}\; CNT2$$

can be decomposed into constructing a counter of capacity 1, *cnt1*, and a parametric process *mk-cnt2* that constructs a counter of capacity 2 given a counter of capacity 1:

$$\vdash \; mk\text{-}cnt2 \;\textbf{sat}\; CNT1 \rightarrow CNT2$$

$$\vdash \; cnt1 \;\textbf{sat}\; CNT1$$

CNT1 is a specification of a counter of capacity 1:

$$\begin{array}{rcl} CNT1 & \equiv & (0 \leq up\#h - dn\#h \leq 1) \wedge (up\#h - dn\#h = 0 \;\Rightarrow\; up \in F) \wedge \\ & & (up\#h - dn\#h = 1 \;\Rightarrow\; dn \in F) \end{array}$$

Finally, the processes that realize the specifications can be defined as follows:

$$\begin{array}{lcl} mk\text{-}cnt2 & \equiv & \lambda X : CNT1 \,.\, (X[lk/up] \| X[lk/dn]) \backslash lk \\ cnt1 & \equiv & \mu X \,.\, up.dn.X \\ cnt2 & \equiv & mk\text{-}cnt2 \; cnt1 \end{array}$$

Using the proof system for parametric processes we can derive

$$\vdash \; cnt2 \;\textbf{sat}\; CNT2$$

as well as the remaining satisfaction assertions above.

Acknowledgement

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

References

[Barendregt 92] 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.

[Holmström 89] S. Holmström, A refinement calculus for specification in Hennessy-Milner logic with recursion, Formal Aspects of Computing, vol. 1 (3). pp. 242–272, 1989.

[Larsen and Milner 86] K. Larsen and R. Milner, A Complete Protocol Verification using Relativised Bisimulations, R 86-12, Institute of Electrical Systems, Aalborg University Centre, 1986.

[Milner 89] R. Milner, *Communication and Concurency*, Prentice Hall, 1989.

[Olderog 91] E.-R. Olderog, *Nets, Terms and Formulas*, Cambridge University Press, Cambridge 1991.

[Olderog 91a] E.-R. Olderog, Towards a design calculus for communicating programs, in: Proc. CONCUR '91, LNCS, Springer, 1991.

[Sokołowski 94] S. Sokołowski, The GDM approach to specifications and their realizations. Part I: Specification systems. Technical Report, Gdańsk, 1994.