Taxonomy and Expressiveness of Preemption: A syntactic approach ⋆

Sophie Pinchinat[1], Éric Rutten[1], and R.K. Shyamasundar[2]

[1] EP-ATR Group, IRISA, F-35042, Rennes, France, e-mail: {pinchina, rutten}@irisa.fr

[2] Computer Science Group, Tata Institute of Fundamental Research, Bombay 400 005, India, e-mail: shyam@tcs.tifr.res.in

Abstract. We propose a taxonomy of preemptive (*suspensive* and *abortive*) operators capturing the intuition of such operators that exist in the various synchronous languages. Some of the main contributions of the paper are: a precise notion of *preemption* is established at a structural level ; we show that the class of *suspensive* operators is strictly more expressive than *abortive operators*, and we show that *suspension* is primitive while *abortion* is not.

The proof techniques relies on a syntactic approach, based on SOS-specification formats, to categorize the different preemption features ; also an equivalence criterion between operators specifications is proposed to provide us with expressive power measurement.

1 Introduction

Process preemption is the notion of controlling the life and death of an activity. Various forms of preemption such as *interrupts* have been in use in the context of operating systems for a long time. Well-known occurrences are suspension and abortion of processes (also known as, in the case of the UNIX operating system, respectively `^z` and `^c`). However, it is only recently that the vital role of preemption operators is coming to light in the context of synchronous programming languages. Berry [Ber93] has argued for the need and importance of *process preemption* operators for programming reactive and real-time systems that have inherently a time-dependent model. A variety of preemption operators can be seen in synchronous languages or formalisms such as ARGOS, ESTEREL, LUSTRE, STATECHARTS, SIGNAL etc, [pro91] (Some of these operators are described in Example 4 of this paper).

In spite of the wide usage of the notion of preemption, there have been no attempts in characterizing these operators at structural level or transformational level. The advantage of such a characterization is that it is possible to look at existing notions like expressiveness or completeness in the context of synchronous calculi such as Meije [Sim85] and SCCS [Mil83], this time, in the context of

⋆ Work supported by IFCPAR (Indo-French Center for the Promotion of Advanced Research), New Delhi.

preemption. Such a characterization will also make it possible to characterize classes of operators with *good behaviors* based on syntactic formats such as *GSOS* [BIM95], *tyft* [GV92] etc. Furthermore such frameworks would also lead to the study of canonical models from the operational models.

In this paper, we provide a characterization of preemption operators within such a spirit. First, we propose an SOS-specification of preemption operators through a sub-fragment of the *tyft* format [GV92] called the *xyfg/xyfz* format. This framework leads to the natural categorization of the *suspensive* and *abortive* operators capturing the intuition of such operators that exist in the various synchronous languages. Some of the main contributions of the paper are:

- A precise notion of *preemption* at a structural level is established.
- It is shown that the class of *suspensive* operators is strictly more expressive than *abortive operators*, and
- From the class of operators we consider (see Section 2.2) it is further established that *suspension* is primitive while *abortion* is not.

The paper is organized as follows : Section 2.1 introduces the framework of SOS-specifications for describing operators and proposes a criterion of equivalence between operators (and specifications). Classes of preemption operators will be described in so-called *xyfg/xyfz* specifications, introduced in Section 2.2. In Section 3, the taxonomy of preemption operators is presented ; it uses syntactic criteria over the specification rules of the operators. Finally, Section 4 answers the expressiveness issues and Section 5 concludes this work.

2 Preliminary notions

2.1 General notions

Notation: We use set V to denote a denumerable set of *term variables*, whose typical elements denoted by $v, v_1, v_2 ...$ and X to denote a denumerable set of meta-variables, whose typical elements are denoted by $x, x_1, ..., y, y_1 ....$

The distinction between meta-variables and variables is made for pedagogical purposes: meta-variables like $x, x_1, ...$ will be used to describe clauses of the operational rules schemes, whereas variables $v, v_1, ...$ will be used to refer to contexts as open terms of the language.

Definition 1. (Signature) A *signature* is a pair $\Sigma = (F, \nabla)$ where $F = \{f, g, ...\}$ is a denumerable set of *function names* disjoint from $V \cup X$, and $\nabla : F \to \mathbb{N}$ is the *arity function*. We write $(f, n) \in \Sigma$ to express that $f \in F$ and $\nabla(f) = n$.

$T(\Sigma \cup V \cup X)$ will denote the set of terms over Σ that might possibly contain variables in $V \cup X$, and will be ranged over by $T, T_1, T_2, ..., S, U, ...$ For $T \in T(\Sigma \cup V \cup X)$, $Var(T) \subseteq V \cup X$ denotes the set of variables occurring in T. If $Var(T) \neq \emptyset$ then T is called an *open term (over Σ)*; otherwise it is a *closed term*. $T(\Sigma)$ will denote the set of terms over Σ, with typical elements $t, t', t_1, t_2, \cdots$.

The *height* of a term $T \in T(\Sigma \cup V \cup X)$, written $|T|$, is defined by induction over the structure of T by $|T|$ is 1 if T is a variable, otherwise T is some $f(T_1, ..., T_n)$, then $|T|$ is $1 + max\{0, |T_1|, ..., |T_n|\}$

A *substitution* is any partial function $V \cup X \rightarrow T(\Sigma \cup V \cup X)$. Typical elements for substitutions will be written $\sigma, \sigma', \rho, \xi, ...$. Write $Dom(\sigma)$ the domain of partial functions σ, and $v\sigma$ the image of variable v by σ.

Any substitution σ can be extended to the domain of open terms over Σ by taking $T\sigma$ as the term obtained by replacing in T any $v \in Dom(\sigma)$ by $v\sigma$.

Classically, given two open terms T, T' over Σ and $v \in Var(T)$, $T[T'/v]$ is the term $T\sigma$ where $Dom(\sigma) = \{v\}$ and $v\sigma = T'$.

Example 2. Consider signature Σ_0 composed of $\{(nil, 0), (a\supset, 1), (a>, 1), (a., 1), (b., 1), (>, 2), (||, 2), (\times, 2), (+, 2)\}$. Using infix notations for operators, examples of terms are $a.(nil), || \ (a.(nil), b.(nil)), a> (a.(nil)) \in T(\Sigma_0)$, and $a \supset (v), > (v_1, v_2) \in T(\Sigma_0 \cup V)$.

Classically (see [Plo81], [GV92]), a formal system is associated to the signature in order to define the operational semantics of closed terms. This system is composed of deductive rules that are used to prove (labeled) transitions between terms. Such a set of rules together with the signature, is called an *SOS specification*.

Throughout the paper, we assume a denumerable set of action names, A disjoint from $\Sigma \cup V \cup X$.

Definition 3. (SOS specification) An *SOS specification* (over A) is a tuple $P = (\Sigma, R)$ where Σ is a signature and R is a set of rules of the form

$$\frac{\{T_i \xrightarrow{a_i} T_i' \mid i \in I\}}{T \xrightarrow{a} T'} \qquad \text{where } I \text{ is finite, } T_i, T_i', T, T' \in T(\Sigma \cup X),\ a, a_i \in A.$$

Expressions $T_i \xrightarrow{a_i} T_i'$ are called the *premises* of the rule.

A specification (Σ, R) is *elementary* if $\nabla_\Sigma(f) = 0$ for all $f \in F_\Sigma$.

We assume the reader is familiar with the notion of *proof* of a transition in the formal system given by the set of rules. However, a formal definition is given in the Appendix. We write $\vdash^P_{\mathcal{P}} t \xrightarrow{a} t'$ if $\mathcal{P}$ is a proof of $t \xrightarrow{a} t'$ (in P), and $\vdash^P t \xrightarrow{a} t'$ if there is some proof of this transition.

We say that two specifications P_1 and P_2 *agree* if for all $f \in \Sigma_1 \cap \Sigma_2$, $\nabla_{\Sigma_1}(f) = \nabla_{\Sigma_2}(f)$. Given two agreeing specifications P_1 and we define the *union* of P_1 and P_2, $P_1 \oplus P_2$ by $(\Sigma_1 \cup \Sigma_2, R_1 \cup R_2)$, where $\Sigma_1 \cup \Sigma_2$ is standardly defined. Two specifications P_1 and P_2 are *disjoint* if $\Sigma_1 \cap \Sigma_2 = \emptyset$ (and therefore $R_1 \cap R_2 = \emptyset$).

Example 4. **(Typical SOS Rules)** Using Σ_0 defined above, SOS-specification P_0 over $\{a, b\}$ is given below; P_0 is based on several operators well known from the literature (with prefix notation instead of usual infix one).

Pure Esterel ("a suspend" operator)

$$\frac{}{a\supset(x_1) \xrightarrow{a} a\supset(x_1)} \quad (1) \qquad \frac{x_1 \xrightarrow{b} y_1}{a\supset(x_1) \xrightarrow{b} a\supset(y_1)} \quad (2)$$

Pure Esterel ("do watching a" operator)

$$\frac{x_1 \xrightarrow{a} y_1}{a{>}(x_1) \xrightarrow{a} nil} \quad (3) \qquad \frac{x_1 \xrightarrow{b} y_1}{a{>}(x_1) \xrightarrow{b} a{>}(y_1)} \quad (4)$$

Let $\mu \in \{a, b\}$,
Operator ∇ of [SP95]

$$\frac{x_2 \xrightarrow{\mu} y_2}{\nabla(x_1, x_2) \xrightarrow{\mu} y_2} \quad (5) \qquad \frac{x_1 \xrightarrow{\mu} y_1}{\nabla(x_1, x_2) \xrightarrow{\mu} \nabla(y_1, x_2)} \quad (6)$$

SCCS, Meije,...

$$\frac{}{\mu.(x_1) \xrightarrow{\mu} x_1} \quad (7) \qquad \frac{x_1 \xrightarrow{\mu} y_1}{+(x_1, x_2) \xrightarrow{\mu} y_1} \quad (8)$$

$$\frac{x_2 \xrightarrow{\mu} y_2}{+(x_1, x_2) \xrightarrow{\mu} y_2} \quad (9) \qquad \frac{x_1 \xrightarrow{\mu} y_1}{||\ (x_1, x_2) \xrightarrow{\mu} ||\ (y_1, x_2)} \quad (10)$$

$$\frac{x_2 \xrightarrow{\mu} y_2}{||\ (x_1, x_2) \xrightarrow{\mu} ||\ (x_1, y_2)} \quad (11) \qquad \frac{x_1 \xrightarrow{\mu} y_1 \quad x_2 \xrightarrow{\mu} y_2}{||\ (x_1, x_2) \xrightarrow{\mu} ||\ (y_1, y_2)} \quad (12)$$

$$\frac{x_1 \xrightarrow{\mu} y_1 \quad x_2 \xrightarrow{\mu} y_2}{\times(x_1, x_2) \xrightarrow{\mu} \times(y_1, y_2)} \quad (13)$$

SOS-specifications naturally describe a set of programs defined by their operational semantics, i.e. their corresponding state-transitions graphs also called *Labeled Transition Systems*.

Definition 5. (Labeled Transition System, $TS(P)$, executions, traces)
A *labeled transition system (over A)* is a structure $S = (Q, \rightarrow)$ where Q is a set of states, and $\rightarrow \subseteq Q \times A \times Q$ is the transition relation. $(q, a, q') \in \rightarrow$ will be written $q \xrightarrow{a} q'$.

Given $q \in Q$, we write $Ex_S(q)$ for the set of all *executions starting from q* and defined by all the maximal (finite or infinite) sequences of the form $q =$

$q_0 \overset{a_0}{\to} q_1 \overset{a_1}{\to} q_2$.... A *trace starting from* q is a word a_0a_1... s.t. there exists an execution from q of the form $q = q_0 \overset{a_0}{\to} q_1 \overset{a_1}{\to} q_2$.... We write $Tr(q)$ for the set of traces starting from q.

Given a specification $P = (\Sigma, R)$ (over A), we write $TS(P)$ for the labeled transition system over A associated to P and defined by $TS(P) = (T(\Sigma), \to)$ where $t \overset{a}{\to} t'$ whenever $\vdash^P t \xrightarrow{a} t'$.

Definition 6. (Bisimulation over labeled transition systems) [**Par81,Mil89**] Let $S_1 = (Q_1, \to_1)$ and $S_2 = (Q_2, \to_2)$ be two labeled transition systems over A. A *bisimulation* relation between S_1 and S_2 is a total[1] relation $\varrho \subseteq Q_1 \times Q_2$ s.t. $(q_1, q_2) \in \varrho$ implies for all $q_1 \overset{a}{\to}_1 q_1'$ there exists $q_2 \overset{a}{\to}_2 q_2'$ with $(q_1', q_2') \in \varrho$, and vice versa, for all $q_2 \overset{a}{\to}_2 q_2'$ there exists $q_1 \overset{a}{\to}_1 q_1'$ with $(q_1', q_2') \in \varrho$.

Since bisimulation relations between S_1 and S_2 are closed under arbitrary unions, there exists a greatest bisimulation, which we write $\sim_{S_1,S_2}$. We simply write $\sim_S$ whenever $S_1 = S_2 = S$ and we omit the subscripts when they are clear from the context.

Specifications also describe program combinators, called *contexts* here; this is formalized in the following.

Definition 7. (Contexts) Given a specification $P = (\Sigma, R)$, a Σ-*context* is an open term $T \in T(\Sigma \cup V)$ where all variables occurring in T are distinct. We write $C(\Sigma)$ the set of Σ-contexts, and $C^h(\Sigma)$ the set of contexts of height $h \in \mathbb{N}$. We call an n-*hole context* a context with n variables. Zero-hole contexts are closed terms; they represent programs.

Among Σ-contexts, those of the form $f(v_1, ..., v_n)$, with $n > 0$ and $v_1, ..., v_n \in V$, will be called Σ-*operators*, and we shall write $O(\Sigma) \subseteq C(\Sigma)$ for the set of Σ-operators.

Clearly, any mapping ϕ from $O(\Sigma)$ to some domain of terms (maybe based on another signature Σ'), can be structurally extended to $C(\Sigma)$ according to $\phi(v) = v$ for all $v \in V$ and $\phi(f(T_1, ..., T_n)) = \phi(f(v_1, ..., v_n))[\phi(T_i)/v_i]$.

In order to compare the expressive power of operators, we need a notion of operational equivalence between specifications. To do so, we introduce a notion of *open bisimulation* over *contexts* which coincides with the standard notion of bisimulation over closed terms (see Definition 6) when equivalent contexts are applied to bisimilar programs. The "programs" will be given separately by an elementary specification (0-arity operators) describing their operational semantics.

Definition 8. (Open bisimulation) Let $P_i = (\Sigma_i, R_i)$ $(i = 1, 2)$ be two specifications. An *open bisimulation* between P_1 and P_2 is a total relation $\chi \subseteq C(\Sigma_1) \times C(\Sigma_2)$ s.t. for any elementary specification $B = (\Sigma, R)$ (disjoint from P_1 and P_2)

[1] $\forall q_1 \in Q_1, \exists q_2 \in Q_2, (q_1, q_2) \in \varrho$ and vice versa.

the relation $\{(T_1\sigma, T_2\sigma')|\ (T_1, T_2) \in \chi, \sigma, \sigma' : V \to T(\Sigma)$ and $v\sigma \sim_{TS(B)} v\sigma'\}$ is a bisimulation between $TS(P_1 \oplus B)$ and $TS(P_2 \oplus B)$. In the following, we shall write $T_1\ \chi\ T_2$ instead of $(T_1, T_2) \in \chi$.

Definition 8 of *open bisimulation* can be related to proposals in the literature: it generalizes the *context bisimulation* of [Lar89] and coincides with the *instance closed bisimulation* of [Ren97]. However, we do not enter more details since an accurate comparison between the variants of "open bisimulations" is a topic in its own and would be out of the scope of this paper.

We say that two specifications P_1 and P_2 are *equivalent*, written $P_1 \equiv P_2$, if there exists an open bisimulation between P_1 and P_2; $\equiv$ is indeed an equivalence relation. Notice that we do not require any congruence property for $\equiv$: indeed, open bisimilar contexts do not necessarily remain equivalent when instantiated by bisimilar sub-contexts. The congruence property can be achieved by two-ways explicit syntactic mappings between contexts, leading to a notion of *effectively equivalent* specifications.

Definition 9. (Effectively equivalent specifications) We say that P_1 and P_2 are *effectively equivalent* if there exist two mappings $\phi_{12} : O(\Sigma_1) \to C(\Sigma_2)$ and $\phi_{21} : O(\Sigma_2) \to C(\Sigma_1)$ such that the relation $\chi = \{(T, \phi_{12}(T))|T \in C(\Sigma_1)\} \cup \{(\phi_{21}(S), S)|S \in C(\Sigma_2)\}$ is an open bisimulation between P_1 and P_2

Effective equivalence of specifications is a strong notion with concrete applications as it provides us with a translation between programs (i.e. closed terms) in a compositional way.

2.2 The format

Confining to particular rule formats enables to characterize precisely the class of operators we are able to treat. In this paper, we consider a subclass of so-called *tyft* format operators of [GV88,GV92]. This subclass is called the *xyfg/xyfz* format.
We assume given a specification $P = (\Sigma, R)$ whose rules have the general following form:

$$\frac{\{T_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_1, ..., x_n) \xrightarrow{a} T} \tag{14}$$

where I is a finite set of indexes, $f \in \Sigma$, x_j $(1 \leq j \leq n)$ and y_i $(i \in I)$ are all different variables from X, $a_i, a \in A$, $T_i, T \in T(\Sigma \cup X)$. We then say that r is *a rule for* f, and we define $op(r) = f$, $lht(r) = f(x_1, ..., x_n)$ (for "left-hand term"), $rht(r) = T$ (for "right-hand term"). Also, we let $Var(r)$ denote the set of all variables occurring in $lht(r)$.

Definition 10. (xyfg/xyfz format) An SOS rule is in the *xyfg/xyfz* format if it is of the form (15) or (16) below (where $I \subseteq \{1, ..., n\}$, $(f, n) \in \Sigma$ and the x_j's and the y_i's are distinct variables):

Basic rules

$$\frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_1, ..., x_n) \xrightarrow{a} g(z_1, ..., z_m)} \tag{15}$$

where $(g, m) \in \Sigma$, and $\{z_1, ..., z_m\} \subseteq \{x_j\}_{1 \leq j \leq n} \cup \{y_i\}_{i \in I}$ are distinct variables s.t. $\forall i \in I$, $\{x_i, y_i\} \not\subseteq \{z_1, ..., z_m\}$. Therefore $m \leq n$.

Projective rule

$$\frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_1, ..., x_n) \xrightarrow{a} z} \tag{16}$$

where $z \in \{x_j\}_{1 \leq j \leq n} \cup \{y_i\}_{i \in I}$

All the rules of Example 4 are xyfg/xyfz : Rules (3), (5), (7), (8) and (9) are projective, the others are basic. In (3), operator $(nil, 0) \in \Sigma_0$ corresponds to (g, m) in (15).

The xyfg/xyfz format is at the crossing of several formats in the literature : first, it is a fragment of the pure tyft format of [GV92]. However, in tyft rules as in (14), term T can be any open term whereas in an xyfg/xyfz rule, term $T \in O(\Sigma)$. Also, our format respects the definition of [Pin91] referred to as "without copy" format. Copying allows several occurrences of the same variable x (or occurrences of derived variables y in the premises) to occur in the right hand side term of the rule. Finally, xyfg/xyfz format strictly generalizes the *basic* format of [BD92]. As far as we know, most of the well-known description languages of the literature (e.g. Meije, SCCS, Esterel, Signal, ...) bear an SOS-specification description in this format (see Example 4).

An operator of a given specification is an *xyfg/xyfz operator* if all the rules for it are xyfg/xyfz, and we call an *xyfg/xyfz specification* (or simply "specification" for short in the following) any SOS-specification $P = (\Sigma, R)$ s.t. every $f \in \Sigma$ is xyfg/xyfz.

3 A taxonomy for preemption

The notion *preemption* has been widely associated with interrupts, process scheduling and operating systems. For instance, killing and suspending processes (for example `^c` or `^z` in the UNIX operating systems) have been used widely in operating systems and hardware description languages. However, it is only in the works of G. Berry on ESTEREL, one finds the notion of *preemption* elevated to the status of an operator explicitly. In fact, [Ber93] argues the need of considering preemption and concurrency as orthogonal features in reactive language specifications. The two primary categories of preemptive operators are the *suspensive* and *abortive* operators. In fact, these operators can be seen in the new look-ahead architectures abundantly (look-ahead or IA-64 architecture). In terms of the evolution of the state processes, besides execution (transition from one state to the next, different one), we would have suspension (transition while remaining in the same state) and abortion (transition with loss of any state information about the process). From this point of view, operators intuitively considered not preemptive can nevertheless be regarded as performing suspension or abortion. For example, the sequence operator, as in `P1 ; P2`, can indeed be understood as an operator that "installs" both processes, `P2` in a suspended state initially, suspends `P2` as long as `P1` executes, and aborts `P1` when it terminates, while `P2` is resumed. Of course it might not at all be implemented that way, and this point might be seen as counter-intuitive (in classical architectures). But if one considers a system where installing a process would take more time than resuming it, then this way of achieving sequence might be in favour of reaction fastness. And anyway our concern here is that the behaviour defined is the one where first that of `P1` is observed, and then that of `P2`.

The intuition behind the use of a syntactic criteria lies in the consideration that abortion is characterized by the definitive loss of information

In this section, preemption is characterized in a syntactic way: this characterization is based on the syntax of the rule schemes describing the operators.

According to the intuition, a unary operator, like operator $a\supset(v)$ in Example 4, has a suspensive feature if there exists a rule for it, namely Rule (1), in which operator leaves its argument unchanged by the transition. This can be generalized to higher arity operators, see below. On the other hand, abortive operators would be characterized by SOS-rules in which no reference to the argument, or to any of its derivatives, will occur in the target term. Definitions below also introduce "relax" operators as a natural complementary notion of the suspensive and abortive ones.

Definition 11. (Suspensive, abortive and evolutive rules) A rule is said to be *suspensive with respect to the j-th argument* (or *w.r.t. j* for short) if $x_j \in Var(rht(r))$. It is *abortive w.r.t. the j-th argument* (or *w.r.t. j*) if $x_j \notin Var(rht(r))$ and (if y_j is defined) $y_j \notin rht(r)$. Finally, it is *evolutive w.r.t. the j-th argument* (or *w.r.t. j*) if $j \in I$ and $y_j \in rht(r)$. A rule is *relax* if it is evolutive w.r.t. all its arguments.

In Example (4), Rule(1) (resp. (2)) is suspensive (resp. evolutive) w.r.t. 1. Rule (3) is abortive w.r.t. 1, and Rule (4) is relax. Rule (5) is abortive w.r.t. 1 and evolutive w.r.t. 2, and Rule (6) is evolutive w.r.t. 1 and suspensive w.r.t. 2. Rule (7) is suspensive w.r.t. 1. Rule (8) (resp. (9)) is abortive w.r.t. 2 (resp. 1). Etc...

The taxonomy of Definition 11 extends to (Σ-)operators. We obtain definitions for *preemptive* (more precisely *suspensive* or *abortive*) and *non-preemptive* (also called *relax*) ones:

Definition 12. (Suspensive, (pure) abortive and relax operators)
Let f be an xyfg/xyfz operator. We say that f is

- *suspensive* if it has a suspensive rule w.r.t. some j and no abortive rule.
- *abortive* if there exists a rule for f abortive w.r.t. some j
- *relax* if all the rules for f are relax.

We write $\mathcal{S}$ (resp. $\mathcal{A}$, $\mathcal{R}$) the sub-class of suspensive (resp. abortive, relax) operators. Sub-classes $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$ form a partition of xyfg/xyfz operators. An operator is *preemptive* if it is suspensive or abortive. Among abortive operators, we distinguish so-called *pure abortive* ones defined as abortive operators with no suspension feature, i.e. with no suspensive rule for it. In the following, let $\mathcal{A}_p$ denote the class of pure abortive operators.

Refering to our example: $a\supset$, $\mu.$ and $||$ are suspensive, $a>$ and $+$ are pure abortive. $>$ is abortive but not pure abortive as it possesses a suspensive rule (see Rule (6)). Finally, $\times$ is relax. The sequence operator mentioned above would have a suspensive rule, as well as an abortive rule, and hence, as an operator, it is to be considered abortive. In fact, such a flexibility allows the interpretation in various architectures (including, lookahead architectures).

In the literature, mostly classes $\mathcal{S}$ and $\mathcal{A}_p$ have been considered. In the Esterel language [Ber93], the "**a suspend**" operator belongs to $\mathcal{S}$. It describes the suspension of a process at a signal occurrence (here for signal **a**). The various versions of **do watching** as well as the **trap** belong to $\mathcal{A}_p$. However, non pure abortive operators arise very naturally : in [SP95], operator $\triangledown$ describes the abortion of a process by the starting of another process.

4 Expressiveness issues

We now address expressiveness issues for the classes of operators $\mathcal{S}$ and $\mathcal{A}$. The main result of this paper shows that the class $\mathcal{S}$ of suspensive operators is strictly more expressive than the class $\mathcal{A}$ of abortive ones.

Basically, we show how to replace abortive operators by suspensive ones. To do so, we replace every abortive rule by a suspensive one, by keeping track of the absorbed variables in the right hand term of the rule. Therefore, the arity of the right hand term operator has to be increased. However, increasing the arity cannot be made locally for each rule, but has to take into account all the rules in which the concerned operator is involved. To do so, we give the specification a canonical decomposition into sub-specifications, each of them being associated an increase of the arity for each operator. By transforming uniformly all the abortive operators into new suspensive ones with greater arity and new rules, we show that the obtained specification is effectively equivalent to the original one.

Let P be a specification. We first explain how to decompose P: define the *operators graph of* P, written $G(P)$, as the non-directed graph whose vertices are all the function names $f \in \Sigma_P$ and whose edges are the pairs (f, g) s.t. there exists a rule r with $lht(r)$ of the form $f(...)$ and $rht(r)$ of the form $g(...)$, or vice versa.

P is *connected* if $G(P)$ is connected. Otherwise, decompose $G(P)$ into connected components $\{\Sigma_1, ...\}$ (Note that there might be infinitely many Σ_k's.) and partition R according to the following equivalence relation: "*$op(r_1)$ and $op(r_2)$ belong to the same connected component*". Denote the partitions of the equivalence relations by $\{R_1, R_2, ...\}$ where the indices for R_k's have been chosen in a such way that $r \in R_k$ iff $op(r) \in \Sigma_k$. Clearly, $P = \bigoplus_k P_k$, where $P_k = (\Sigma_k, R_k)$. Such a decomposition is unique and is referred to as *the canonical decomposition of* P.

Let $\Sigma = (F, \nabla)$ be a signature with bounded arity function, and define N as the maximal element of the set $\{\nabla(f) \mid f \in \Sigma\}$. We define $\Sigma_{\uparrow N}$ by $F_{\Sigma_{\uparrow N}} = \{f_{\uparrow N} \mid f \in F_\Sigma\}$ and $\nabla_{\Sigma_{\uparrow N}} : F_{\Sigma_{\uparrow N}} \to \mathbb{N}$ delivering the constant value N.

Finally, for a rule r, we define $abs(r) \subseteq X$ the set of *absorbed variables* in r, to denote all the variables x_k that appear in $lht(r)$, s.t. neither x_k nor y_k (corresponding to a premise if any) occur in $rht(r)$. For example, in Rule (5), the set of absorbed variables is $\{x_1\}$.

We now define an extension of a connected specification:

Definition 13. (The N-extension of P) let $P = (\Sigma, R)$ be a connected specification and assume the set $\{\nabla(f) \mid f \in \Sigma\}$ has a maximal element N.
We define the *N-extension of* P, written $P_{\uparrow N}$, by

- **case (1)** If for all $f \in F_\Sigma$, $\nabla_\Sigma(f) = N$ then $\Sigma_{P_{\uparrow N}} \stackrel{\text{def}}{=} \Sigma_{\uparrow N}$,
  case (2) otherwise, there exists $f \in F_\Sigma$ s.t. $\nabla_\Sigma(f) < N$ (and therefore $N \geq 1$), then $\Sigma_{P_{\uparrow N}} \stackrel{\text{def}}{=} \Sigma_{\uparrow N} \cup \{(\Pi, N)\}$, where Π is a new function name.
- $R_{P_{\uparrow N}}$ contains the rules (we write $\boldsymbol{abs(r)}$ to denote the tuple of variables in $abs(r)$ ordered by increasing indexes) :
  Extended basic rules : if r is like (15) then $r_{\uparrow N}$ is ($\mathbf{x_{n+1}}, ..., \mathbf{x_N}$ are fresh meta-variables)

$$\frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I\}}{f_{\uparrow N}(x_1, ..., x_n, \mathbf{x_{n+1}}, ..., \mathbf{x_N}) \xrightarrow{a} g_{\uparrow N}(z_1, ..., z_m, \boldsymbol{abs(r)}, \mathbf{x_{n+1}}, ..., \mathbf{x_N})} \quad (17)$$

  Extended projective rules : if r is like (16) then $r_{\uparrow N}$ is ($\mathbf{x_{n+1}}, ..., \mathbf{x_N}$ are fresh meta-variables)

$$\frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I\}}{f_{\uparrow N}(x_1, ..., x_n, \mathbf{x_{n+1}}, ..., \mathbf{x_N}) \xrightarrow{a} \mathbf{\Pi}(z, \boldsymbol{abs(r)}, \mathbf{x_{n+1}}, ..., \mathbf{x_N})} \quad (18)$$

  If R contains a projective rule, then also add the following rule (referred to as *-rule):

$$\frac{\mathbf{x_1} \xrightarrow{a} \mathbf{y_1}}{\mathbf{\Pi}(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_N}) \xrightarrow{a} \mathbf{\Pi}(\mathbf{y_1}, \mathbf{x_2}, ..., \mathbf{x_N})} \quad (19)$$

For example, consider P, composed of rules (5) and (6) for operator ∇. Then the maximal arity being 2, and Rule (5) being projective, $R_{\uparrow 2}$ is then

$$\frac{x_2 \xrightarrow{\mu} y_2}{\nabla(x_1, x_2) \xrightarrow{\mu} \Pi(y_2, y_1)} \quad (20) \qquad \frac{x_1 \xrightarrow{\mu} y_1}{\Pi(x_1, x_2) \xrightarrow{\mu} \Pi(y_1, x_2)} \quad (21)$$

$$\frac{x_1 \xrightarrow{\mu} y_1}{\nabla(x_1, x_2) \xrightarrow{\mu} \nabla(y_1, x_2)} \quad (22)$$

Extending connected specifications leads to only suspensive or relax operators (By construction $abs(r_{\uparrow N}) = \emptyset$ for all $r \in R$. Noting that $abs(r) = \emptyset$ iff for every j, r is suspensive or evolutive w.r.t. j, concludes the proof.). Also, P is connected and has no abortive rule entails that P and $P_{\uparrow N}$ are isomorphic (see definition in the Appendix).

Theorem 14. *Let P be a connected specification with bounded arity (call N the maximum), then P and $P_{\uparrow N}$ are effectively equivalent (See Appendix for the full proof.).*

Notice that specification $P_{\uparrow N}$ fits the basic format of [BD92]. Theorem 14, but mostly its generalization in Theorem 15, shows that, assuming the $\equiv$-equivalence criterion is accepted, the xyfg/xyfz format is not more expressive than the basic format. The translation from P to $P_{\uparrow N}$ would precisely deliver the same basic format presentation of CCS proposed in [BD92] with moreover a proof that it is correct with regard to the original specification of CCS in [Mil81,Mil89].

4.1 Abortion can be simulated by "for ever" suspension

Let *$\mathcal{SR}$-specification* be a specification that contains only suspensive or relax operators. We show how to transform any specification into an equivalent $\mathcal{SR}$-specification. This is formalized by Theorem 15. Note that the result also holds for infinite specifications provided connected sub-specifications have bounded arity.

Theorem 15. *Let P be a specification (with bounded arity for each connected sub-specification of P). Then there exists an $\mathcal{SR}$-specification, $\hat{P}$, which is effectively equivalent to P.*

Proof (sketch): We use Theorem 14 for each component of the canonical decomposition of P; see Appendix for the full proof.

Theorem 15 is powerful because it delivers a constructive way to translate any abortive operator into suspensive and relax ones. Moreover, extending the arity of operators w.r.t. the connected sub-specification they belong to minimizes the arities in the extended specifications. This technique also applies to infinitely many connected specifications even if no global maximal arity in the signature exists.

4.2 Suspension is primitive

We prove here that class $\mathcal{S}$ is strictly more expressive than class $\mathcal{A}_p$ in the sense that it is not possible to express suspension by means of pure abortive and/or relax operators (the question for non pure abortive operators is irrelevant because non-pure abortive operators have necessarily some suspensive feature, which trivially can be used to encode suspension). We show the result in the framework of "terminating" programs, - i.e. programs with only finite executions. In this framework, we first prove that abortive and relax operators preserve termination.

Proposition 16. *Let $P = (\Sigma, R)$ be an $\mathcal{A}_p\mathcal{R}$-specification s.t. for any program δ, i.e. any $(\delta, 0) \in \Sigma$, all executions from δ are finite. Then, for all $t \in T(\Sigma)$, any execution $\pi \in Ex_{TS(P)}(t)$ is also finite (See Appendix for the proof.).*

We now build an $\mathcal{SR}$-specification P s.t. there is no $\mathcal{A}_p\mathcal{R}$-specification equivalent to P, not even "trace" equivalent: consider P_{susp} the $\mathcal{SR}$-specification which signature is $\{(nil, 0), (\supset, 1)\}$ and which rules are (1) and (2) of Example 4. Clearly, program $a \supset (nil)$ possesses an infinite trace, namely a^ω. Thanks to Proposition 16, no ending $\mathcal{A}_p\mathcal{R}$-specification can be used to build a program which is trace equivalent to $a \supset (nil)$, as all its executions would be finite. This concludes the proof.

5 Conclusion - Debate

In this paper, we have characterized the notion of preemption widely used in reactive and real-time programming languages in the context of SOS specifications. As far as our knowledge goes, this is the first attempt of characterization of preemption in a formal setting rather than invoking virtual/real operating system dependent intuitions. The proposed framework has led to the natural categorization of the *suspensive* and *abortive* operators that exist in the various synchronous languages.

An interesting improvement of this work would be to consider semantic arguments in between the intuition underlying the preemptive concepts and the syntactic categorization proposed here; we believe that a notion of control attached to processes can achieve this issue. Such solution would give a formal rationale to the Definition 12. The latter nevertheless already contains semantic motivations, briefly presented in the introduction of Section 3. We conjecture that any attempt in defining the preemption concepts at a more semantic level would agree with the syntactic taxonomy we defend here, when considered in an SOS setting.

At least, the syntactic categorization provide us with a clear mathematical framework to tackle expressiveness issues: we have shown that *suspensive* operators are strictly more expressive than *abortive operators*, and shown *suspensive* operators to be primitive and the *abortive* to be non-primitive.

The translation of abortive operators into suspensive ones has been made possible through the notion of *effective equivalence* between specifications. We argue that *effective equivalence* between specifications, see Definition 9, is the good notion to achieve both constructive and compositional translations of various operators. Indeed, this makes it possible to reason about operators independently of their context of use.

Finally, it should be remarked that we did not say anything about complexity issues, as only expressiveness aspects are treated here. It should be noted that the present contribution gives a clear comparisons between general infinite classes of operators but also between particular (finite) sets of operators. Theorem 15 is general enough to show first, that the xyfg/xyfz format is in fact not more expressive than the basic format of Badouel and Darondeau, by means of the transformation of a given specification P into $\hat{P}$.

Expressiveness questions still remain to be answered. For example, since suspensive (and relax) operators are now proved to be expressive enough, one can think of a strict minimal expressive complete sub-class of suspensive operators from which any kind of preemption could be derived. In particular, is there any "primitive" suspensive operator s.t. any other operator would be captured, using simple constructions? If so, then this would be another very strong argument in favor of Definition 12.

Acknowledgments The authors would like to thank Philippe Darondeau for the original idea of handling preemption syntactically, Vincent Schmitt for carefully reading the draft and Philippe Schnoebelen for helpful discussion, as well as the anonymous referees for their suggestions and critical remarks.

References

[BD92] E. Badouel and Ph. Darondeau. Structural operational specifications and trace automata. In *Proc. CONCUR'92, Stony Brook, NY, LNCS 630*, pages 302–316. Springer-Verlag, August 1992.

[Ber93] G. Berry. Preemption in concurrent systems. In *Proc. FSTTCS'93, Bombay, India, LNCS 761*, pages 72–93. Springer-Verlag, 1993.

[BIM95] B. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, January 1995.

[GV88] J. F. Groote and F. W. Vaandrager. Structured operational semantics and bisimulation as a congruence. Research Report CS-R8845, CWI, November 1988.

[GV92] J. F. Groote and F. W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, October 1992.

[Lar89] K. G. Larsen. Compositional theories based on an operational semantics of contexts. In *Stepwise Refinement of Distributed Systems. Models, Formalisms, Correctness, Mook, LNCS 430*, pages 487–518. Springer-Verlag, May 1989.

[Mil81] R. Milner. A modal characterisation of observable machine-behaviour. In *Proc. CAAP'81, Genoa, LNCS 112*, pages 25–34. Springer-Verlag, March 1981.

[Mil83] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 23(3):267–310, 1983.

[Mil89] R. Milner. A complete axiomatisation for observational congruence of finite-state behaviours. *Information and Computation*, 81(2):227–247, 1989.

[Par81] D. Park. Concurrency and automata on infinite sequences. In *Proc. 5th GI Conf. on Th. Comp. Sci., LNCS 104*, pages 167–183. Springer-Verlag, March 1981.

[Pin91] S. Pinchinat. Ordinal processes in comparative concurrency semantics. In *Proc. 5th Workshop on Computer Science Logic, Bern, LNCS 626*, pages 293–305. Springer-Verlag, October 1991.

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Lect. Notes, Aarhus University, Aarhus, DK, 1981.

[pro91] Special issue on *Real-time systems designs and programming*, *Proceedings of the IEEE*, volume 79, September 1991.

[Ren97] A. Rensink. Bisimilarity of open terms. In Catuscia Palamidessi and Joachim Parrow, editors, *Proc. EXPRESS '97: Expressiveness in Concurrency (Santa Margherita Ligure, Italy, September 8–12, 1997)*, volume 7 of *entcs*. Elesevier Science Publishers, 1997.

[Sim85] R. De Simone. Higher-level synchronising devices in MEIJE-SCCS. *Theoretical Computer Science*, 37:245–267, 1985.

[SP95] R.K. Shyamasundar S. Pinchinat, E. Rutten. Preemption in reactive laguages (a preliminary report). In *In Algorithms, Concurrency and Knowledge, Proc. of the Asian Computing Science Conference, ACSC'95, Pathumthani, Thailand, LNCS 1023*, pages 111–125. Springer-Verlag, December 1995.

Appendix

Definition of Proofs

Let $P = (\Sigma, R)$ be a specification. We use $\Theta, \Theta_1, ...$ to denote transitions in P, and $\Upsilon, \Upsilon_1, ...$ to denote expressions in rules. A *proof* of a transition Θ *in* P is a finite tree, $\mathcal{P}$, whose edges are ordered and whose vertices are labeled by transitions in P s.t. :

- the root is labeled by Θ,
- if Θ' is the label of some vertex and $\Theta_1, ..., \Theta_m$ are the labels of its children (no children when $m = 0$), then there is a rule $r \in R$ of the form

  $$\frac{\Upsilon_1 \; ... \; \Upsilon_m}{\Upsilon}$$

  and a substitution σ s.t. $\Theta_i = \Upsilon_i \sigma$ and $\Theta' = \Upsilon \sigma$.

  We write $\vdash^P_{\mathcal{P}} \Theta$ if $\mathcal{P}$ is a proof of Θ (in P), and $\vdash^P \Theta$ if $\vdash^P_{\mathcal{P}} \Theta$ for some proof $\mathcal{P}$. We omit the subscripts when they are clear from the context. Sub-proofs are sub-proof-trees.

Isomorphic specifications

Two specifications $P_i = (\Sigma_i, R_i)$ $(i = 1, 2)$ are said to be *isomorphic* if there exists a bijection $\theta : \Sigma_1 \to \Sigma_2$ s.t. for each rule $r_1 \in R_1$, the rule obtained by substituting every rule of R_1; that is, isomorphic specifications are the same modulo renaming of operators.

Proof of Theorem 14

If P contains no abortive rule the result is trivial since P and $P_{\uparrow N}$ are isomorphic. We can now assume that P has at least one abortive rule, entailing $N \geq 1$.

Let define ϕ and ϕ' by $\phi(v) = v$ for all $v \in V$, $\phi(f(v_1, ..., v_n)) = f_{\uparrow N}(v_1, ..., v_n, v_{n+1}, ..., v_n)$ for all $(f, n) \in \Sigma$ where $v_{n+1}, ..., v_N$ are new variables, and $\phi'(v) = v$ for all $v \in V$, $\phi'(f_{\uparrow N}(v_1, ..., v_n, v_{n+1}, ..., v_N) = f(v_1, ..., v_n)$ for all $(f, n) \in \Sigma$. Finally, define $\phi'(\Pi(v_1, ..., v_N) = v_1$. Now extend ϕ (resp. ϕ') to $C(\Sigma)$ (resp. $C(\Sigma_{P_{\uparrow N}})$ by :

- $\phi(f(T_1, ..., T_n) = \phi(f(v_1, ..., v_N))[\phi(T_j)/v_j]_{1 \leq j \leq \nabla(f)}$
- $\phi'(f_{\uparrow N}(S_1, ..., S_N) = f(\phi'(S_1), ..., \phi'(S_{\nabla(f)}))$

To prove that ϕ and ϕ' provide us with the desired result, we first establish the intermediate following result:

Proposition 17. *Let $\chi \subseteq C(\Sigma_P) \times C(\Sigma_{P_{\uparrow N}})$ be the least relation s.t.*

- *$v \;\chi\; v$ for all $v \in V$,*
- *$f(T_1, ..., T_n) \;\chi\; f_{\uparrow N}(S_1, ..., S_N)$, for all $(f, n) \in \Sigma$, all $T_j \;\chi\; S_j$ $(j = 1, ..., n)$ and all $S_{n+1}, ..., S_N$,*
- *$T \;\chi\; \Pi(S, S_2, ..., S_N)$ for all $T \;\chi\; S$ and all $S_2, ..., S_N$.*

Then relation χ is an open bisimulation between P and $P_{\uparrow N}$.

Proof. Let B be an elementary specification, disjoint from P and $P_{\uparrow N}$.

We define $\sim \subseteq T(\Sigma_P \cup \Sigma_B) \times T(\Sigma_{P_{\uparrow N}} \cup \Sigma_B)$ as the least relation such that $\sim \supseteq \sim_{TS(B)}$, and $f(t_1, ..., t_n) \sim f_{\uparrow N}(s_1, ..., s_n, s_{n+1}, ..., s_N)$ whenever $(f, n) \in \Sigma_P$, $t_j \sim s_j$, for all $j = 1, ..., n$, and $t \sim \Pi(s, s_2, ..., s_N)$, whenever $t \sim s$.

It is enough to prove that $\sim$ is a bisimulation between $TS(P \oplus B)$ and $TS(P_{\uparrow N} \oplus B)$.

Let $t \sim s$, and let $\vdash_{\mathcal{P}}^{P \oplus B} t \xrightarrow{a} t'$. We show that there exists $\vdash_{\mathcal{P}'}^{P_{\uparrow N} \oplus B} s \xrightarrow{a} s'$ s.t. $t' \sim s'$. The proof is done by induction over the height of the proof $\mathcal{P}$, written $h(\mathcal{P})$.

$h(\mathcal{P}) = 0$ **:** The only rule r in $\mathcal{P}$ is an axiom. If $r \in R_B$, then $t = \beta \in T(\Sigma_B)$ and $s = \beta' \in T(\Sigma_B)$ for some $\beta \sim_{TS(B)} \beta'$ and it is trivial. Otherwise, $r \in R_P$ is of the form

$$\frac{}{f(x_1, ..., x_n) \xrightarrow{a} rht(r)}$$

and t is some $f(t_1, ..., t_n)$ and s is some $f_{\uparrow N}(s_1, ..., s_N)$ with all $t_j \sim s_j$ $(j = 1, ..., n)$. $\mathcal{P}$ is obtained by applying rule r to t with substitution ν where $x_j \nu = t_j$ and $t' = rht(r)\nu$.

Define ν' by $x_k \nu' = s_k$ for all $k = 1, ..., N$. Rule $r_{\uparrow N}$ together with ν' gives a proof $\vdash_{\mathcal{P}'}^{P_{\uparrow N} \oplus B} s \xrightarrow{a} rht(r_{\uparrow N})\nu'$. By construction of ν', $t' \sim rht(r_{\uparrow N})\nu'$ (Check the two cases when rule r is basic or projective, i.e. where $rht(r)$ is some $g(z_1, ..., z_m)$ or some z).

$h(\mathcal{P}) > 0$ **:** This case is very similar to the previous one, with an additional induction argument for sub-proofs in $\mathcal{P}$. We do not give the proof here.

Reciprocally, let $\vdash_{\mathcal{P}'}^{P_{\uparrow N} \oplus B} s \xrightarrow{a} s'$. We show that there exists some $\vdash_{\mathcal{P}}^{P \oplus B} t \xrightarrow{a} t'$ with $t' \sim s'$. Again we use an induction over the height of $\mathcal{P}'$, $h(\mathcal{P}')$.

$h(\mathcal{P}') = 0$ **:** Proof $\mathcal{P}'$ is based on some axiom. If this axiom is in R_B then $s = \beta' \in T(\Sigma_B)$ and $t = \beta \in T(\Sigma_B)$ with $\beta \sim_{TS(B)} \beta'$ and it is trivial. Otherwise, the axiom belongs to R_P and is necessarily some $r_{\uparrow N}$ (because Rule(*) is not an axiom). Proof $\mathcal{P}'$ is obtained from $r_{\uparrow N}$ and a substitution ν'. W.l.o.g. assume s has the form $f_{\uparrow N}(s_1, ..., s_N)$. Define the substitution $\nu : X \rightarrow T(\Sigma_P \cup \Sigma_B)$ by $x_j \nu = t_j$ for all $j = 1, ..., n$. Applying ν to rule r gives a proof $\mathcal{P}$ s.t. $\vdash_{\mathcal{P}}^{P \oplus B} t \xrightarrow{a} t'$. By construction of ν, $t' \sim s'$ (consider the two cases where $r_{\uparrow N}$ is extended basic or extended projective).

$h(\mathcal{P}') > 0$ **:** W.l.o.g. we can avoid the cases where s is some $\Pi(s_1, s_2, ..., s_N)$ since
$\vdash_{\mathcal{P}'} \Pi(s_1, s_2, ..., s_N) \xrightarrow{a} s'$ iff $s' = \Pi(s'_1, s_2, ..., s_N)$ and $\vdash_{\mathcal{P}''} s_1 \xrightarrow{a} s'_1$ with $h(\mathcal{P}'') < h(\mathcal{P}')$, and $t' \sim \Pi(s'_1, s_2, ..., s_N)$ iff $t' \sim s'_1$.
The case $s = f_{\uparrow N}(s_1, ..., s_N)$ is very similar to the previous one, with an additional induction argument for sub-proofs in $\mathcal{P}'$. It is straight forward.

Now, we can build a proof by induction over the structure of open terms that $T \; \chi \; \phi(T)$ and $\phi'(S) \; \chi \; S$, for all $T \in C(\Sigma_P)$, $S \in C(\Sigma_{P_{\uparrow N}})$, which concludes.

Proof of Theorem 15

Let $P = (\Sigma, R)$ be a (xyfg/xyfz) specification with bounded arity and write $N \stackrel{\text{def}}{=} max\{\nabla_\Sigma(f) \mid f \in \Sigma\}$. Let $P = \bigoplus_k P_k$ be the canonical decomposition of P and let N_k be the maximal arity of P_k.

Definition 18. We define $\hat{P} \stackrel{\text{def}}{=} \bigoplus_k (P_{k \uparrow N_k})$, where in each $P_{k \uparrow N_k}$ the possible additional symbol Π is indexed by k.

Write $\bigoplus_k P_k$ for the canonical decomposition of P, each P_k with maximal arity N_k.

Call ϕ_k and ϕ'_k the mappings in the proof of Theorem 14 when applied to P_k and define $\phi : C(\Sigma_P) \rightarrow C(\Sigma_{\hat{P}})$ by $\phi(T) = T$ if $T \in V$ and $\phi(T) = \phi_k(f(v_1, ..., v_{N_k}))[\phi(T_j)/v_j]_{1 \leq j \leq n}$ if $(f, n) \in \Sigma_k$ and $T = f(T_1, ..., T_n)$.

Notice that ϕ and ϕ_k coincide over $C(\Sigma_k)$. (Also we can define a mapping ϕ' to come back from $P_{\uparrow N}$ in a similar way). It remains to prove that T and $\phi(T)$ are open bisimilar. We reason by induction over the maximal number of alternations of function names belonging to different Σ_k in T : for $T \in C(\Sigma)$, define $alt(T)$, the *maximal number of alternations* in T by (write $T(\epsilon)$ for the topmost symbol of T) as follows:

- $alt(T) = 0$ if $T \in C(\Sigma_k)$,
- $alt(f(T_1, ..., T_n)) = max\{alt(T_j) \mid 1 \leq j \leq n\}$ if $f, T_j(\epsilon) \in \Sigma_k$ for some k, otherwise $1 + max\{alt(T_j) | T_j(\epsilon) \in \Sigma_{k'}$ and $f \notin \Sigma_{k'}\}$

For example, $alt(|| \, (a.(nil), v)) = 2$, $alt(+(v_1, +(v_2, v_3)) = 0$.

If $alt(T) = 0$, then necessarily $T \in C(\Sigma_k)$ for some k and by Theorem 14 T and $\phi_k(T)$ are open bisimilar. Otherwise $alt(T) = h + 1$ with $h \geq 0$. Then T is some l-holes Σ_k-context (with, say, l variables $v_1, ..., v_l$) applied to contexts $T_1, ..., T_l$ with less number of alternations. Formally $T = T^k \rho$ where $T^k \in C(\Sigma_k)$ and $v_j \rho = T_j \in C^{h'}(\Sigma)$ with $h' < h$.

Then by definition of ϕ, $\phi(T) = \phi_k(T^k)[\phi(T_j)/v_j]$.

Let B be an elementary specification. To show that $T\sigma$ and $\phi(T)\sigma'$ are bisimilar for all substitution $\sigma, \sigma' : V \to T(\Sigma_B)$ with $v\sigma \sim_{TS(B)} v\sigma'$, we use the induction hypothesis over T_j and $\phi(T_j)$, that is T_j and $\phi(T_j)$ are open bisimilar. Then $T_j \sigma \sim \phi(T_j)\sigma'$.

Consider the new elementary specification B' composed of programs $T_j \sigma \in T(\Sigma \cup \Sigma_B)$ and $\phi(T_j)\sigma' \in T(\Sigma_{\hat{P}} \cup \Sigma_B)$. By applying Theorem 14 to T^k, we have $T^k \xi \sim \phi_k(T^k)\xi'$ for substitutions $\xi, \xi' : V \to T(\Sigma \cup \Sigma_B) \bigcup T(\Sigma_{\hat{P}} \cup \Sigma_B)$ s.t. $v_j \xi = T_j \sigma \sim \phi(T_j)\sigma' = v_j \xi'$.

Because $T^k \xi = T\sigma$ and $\phi_k(T^k)\xi = \phi(T)\sigma'$ we conclude that $T\sigma \sim \phi(T)\sigma'$.

For lack of space we do not give the proof that $\phi'(S)$ and S are open bisimilar for some suitable definition of mapping $\phi' : O(\Sigma_{P \uparrow N}) \to C(\Sigma_P)$.

Proof of Proposition 16

By induction over $|t|$, the height of t.

If $|t| = 1$, then t is a constant, and we are done by assumption.

Otherwise, let t be s.t. $|t| \geq 2$ and let $\pi \in Ex(t)$. Assume π is infinite, of the form $t = t_0 \overset{a_1}{\to} t_1 \overset{a_2}{\to} t_2....$ The xyfg/xyfz format ensures that $|t_j| \leq |t|$ for all $j \in \mathbb{N}$.

Now because P is an $\mathcal{A}_p\mathcal{R}$-specification, only abortive or relax rules apply along a proof of π, among which only a finite number of abortive ones (as we cannot infinitively decrease the height of (finite) terms.

Then there exists k s.t. a proof of $\pi' \overset{\text{def}}{=} t_k \overset{a_{k+1}}{\to} t_{k+1}..$ only uses relax rules. Clearly π' is an infinite execution if π is.

By assumption t_k cannot be a constant, it is then of the form $f_l(u_1^k, ..., u_n^k)$ with $n \geq 1$, and because only relax rules are applied, all the following t_l's are some $f_l(u_1^l, ..., u^n)$.

Now a proof of $t_k \overset{a_{k+1}}{\to} t_{k+1}$ relies on proofs that $u_i^k \overset{b_i^{k+1}}{\to} u_i^{k+1}$ (for all $i = 1, ..., n$ and some b_i^{k+1}), as well as a proof of $t_{k+1} \overset{a_{k+2}}{\to} t_{k+2}$ relies on proofs that $u_i^{k+1} \overset{b_i^{k+2}}{\to} u_i^{k+2}$ (for all $i = 1, ..., n$ and some b_i^{k+2}), and so on and so worth. Therefore we can build a proof of some infinite execution from, for example, u_1^k, but because $|u_1^k| \leq |t_k| < |t|$, it contradicts the induction hypothesis.