A Process Model for Life Cycles

Maarten M. Fokkinga and Rick van Rein
Dept. of Computer Science, University of Twente, NL
{*fokkinga,vanrein*}*@cs.utwente.nl*

Version of March 12, 2001

Abstract. Life cycles (discussed elsewhere regarding the informal and pragmatic aspects) can be formally defined as a process model with some un-conventional features, namely: multiplicity of events, a restricted visibility of parameters, and 'negative prefixes' to block an event set from happening. We show how to translate these aspects in terms of conventional process constructs.

Several times we manage to present the proofs of an implication and its converse that use *different* induction hypotheses as *one* series of equivalences. This presentation technique seems more generally applicable to and useful for process models.

Contents

A Introduction **1**

B Syntax **2**

C Semantics **6**

D Eliminating parameters **9**

E Eliminating negative prefixes **14**

F Conclusion **19**

A Introduction

1 Life cycles. In this paper we discuss a process model that has been designed for the purpose of giving a formal semantics to *life cycles*. Life cycles, in turn, are being developed by Van Rein[8, 9] in order to close the gap between 'theory' on the one hand (process models like ACP [1], CSP [5], and the π-calculus [6]) and 'practice' on the other hand (state diagrams and state charts [2, 4], and so on). In order to be close to life cycles, our process model exhibits the following un-conventional features: multiplicity of events (event *sets*), a restricted visibility of parameters, and a construct ('negative prefix') to block an event from happening. We show how to translate these aspects in terms of conventional process constructs.

Life cycles are intended to be closer to pragmatic applications such as object oriented process models than conventional process models from theory, but keeping the possibility to perform formal verifications. We shall resist the temptation to describe life cycles in detail, including the arguments why and how life cycles solve the defects or inpracticalities of the 'theoretical' and 'practical' process models. Let us confine attention here to merely motivating the distinguishing features of our process model.

2 The process features. The distinguishing features of our process model are: multiplicity of events, a restricted visibility of parameters, and negative prefixes. Let us briefly motivate these features (for those readers that are already familiar with process models).

Regarding multiplicity of events, our viewpoint is that a life cycle (formalised here as *process*) is a *perspective* on an object or component. There may be many perspectives on an object or component. Different perspectives describe the behaviour of an object with respect to possibly different events. Therefore, an object is described by several life cycles at the same time, and a *single* transition of the object may involve *several* events. This leads to a design in which a single transition in general involves several events.

Regarding the peculiar visibility of parameters, we remark that "transition $a(?x)$" in a process specifies that action $a(v)$ occurs for *some* v from a fixed set of values; in the resulting residual process, variable x then stands for that v. Furthermore, any mentioning of $b(x)$ in the residual process implies, in conventional process models, that the process has $b(w)$ in its alphabet for *all* values w, since the alphabet of events with which communication occurs is static (the same for a process and all its residuals). This fact sometimes leads to a deadlock that we consider undesirable. Therefore, we stipulate that event specification '$b(x)$' is not visible as long as transition step $a(?x)$ has not been passed; as a consequence, in our model the "current alphabet of communication" is called *sensitivity set* and it changes dynamically.

Regarding negative prefixing (or: explicit blocking), the motivation is that this construct is just a slight generalisation of the dynamic character of the sensitivity set, and that it provides a much appreciated practical expressiveness.

3 Overview and results. Section B explains our process notion and notation. Section C gives the semantics of our processes in terms of a transition step system. Each transition step maintains some *environment* information to reflect change of state. After elimination of parameters, in Section D, the transition steps manipulate the process expressions *themselves* to reflect change of state. After elimination of negative prefixes, in Section E, a conventional process is obtained with conventional transition steps, in which the *atomic* actions are *sets* of events. The peculiar visibility of parameters exhibits itself in the notion of *sensibility set* (which might be called the "dynamic alphabet"), and is eliminated in the combination of both steps.

Each of the process models in this sequence of translation steps has it advantages and disadvantages. The first transition step semantics, manipulating environments, is close to real efficient implementations. The second semantics, after parameter elimination, has the shorter and clearer definition, but gives an enormous blow-up of the process expressions. After the final translation conventional processes result, and so existing tools can, in principle, be applied.

Apart from the process model itself, we stress that also the proof presentation deserves attention. There are several equivalences ($\Leftrightarrow$) for which the proofs need different induction hypotheses for the $\Rightarrow$ and the $\Leftarrow$ direction. We have succeeded in *presenting* each of those proofs as one sequence of equivalences, thus being concise and precise at the same time. The technique seems more generally applicable for translations between process models.

When reading this paper, the proofs can be skipped without loss of continuity. The same applies to the Notes; they give supplementary information or recall 'well-known' theory and conventions.

Skip this remark about Skipping. In case you *did* skip this remark, we congratulate you with your self-control; it is a promise that you will succeed in finishing this paper. In case you did *not* skip this remark, we urge you to skip this kind of remarks in the sequel; they are not meant to be read by you! The remarks to be skipped are left out in the abbreviated version(s) of this paper.

B Syntax

4 Basic concepts. The following sets and function are postulated, with conventions regarding meta-variables within parentheses:

action names	$Actn$	$(a, b, c : Actn)$,	$arity : Actn \longrightarrow \mathbb{N}$
values	Val	$(v, w : Val)$	
variables	Var	$(x, y, z : Var)$	

process names $\quad Name \quad (X, Y : Name)$

An environment says what variables stand for what values:

environments $\quad Env = Var \pfun Val \quad (\rho, \sigma, \tau : Env)$

We shall use the Z notation for functions [7]; thus *Env* is a set of *partial* functions, and $\rho = \{x \mapsto v,\ y \mapsto w, \ldots\}$ is a partial function with domain $\dom \rho = \{x, y, \ldots\}$ mapping x to v, y to w, and so on. Operator $\oplus$ denotes *overriding*: $(\rho \oplus \sigma)(x) = \sigma\, x$ if $x \in \dom \sigma$ else $\rho\, x$.

5 Process concept. A *process* P is "something" whose main property is that it may make a *transition step*, thereby *participating* in an *event set* e and *evolving* to another process Q, denoted:

$$P \quad \xrightarrow{e}_{\rho,\sigma} \quad Q$$

In the notation of a process, there will occur variables, and those marked with a '?' are called *choice* parameters: they may get a value assigned in a transition step. Environment ρ gives the way in which the variables in P need to be interpreted (due to preceding steps in which choice parameters have got a value); environment σ denotes the assignment for the choice parameters in this very step (and which will be used in later steps to interpret, together with ρ, the variables in Q).

Another property of a process P is that, at any moment in time, it is *sensitive* for a particular set of events; we call such a set the *sensitivity set* of P, and denote it by βP, by analogy and similarity with the conventional *alphabet* αP. The sensitivity set of a process has influence on the transition steps in the following way. In a transition step with event set e of a composite process $P_1 \| P_2$, each constituent P_i is required to participate for that part of e for which it is sensitive; if some P_i cannot fulfill this requirement, it *blocks* the step; and each P_i is said to *ignore* the events for which it is not sensitive.

6 Process notation. We now define the *abstract* syntax of various sets of *expressions*. The syntax is abstract in the sense that the linear notation is ambiguous. There is no ambiguity, however, when expressions are considered to be trees rather than symbol strings. In the grammar (and in subsequent definitions) we use ",..." to denote a possibly empty, comma-separated repetition of the entire preceding part of the clause. An explanation follows below:

parameter list	(p)	$p ::= x \mid ?x \mid v \mid p, \ldots$
event set	(e)	$e ::= \emptyset \mid \{a(p)\} \mid e \cup e'$
process	(P, Q, R)	$P ::= X \mid \frac{e}{e'}{\to}P \mid P{+}P \mid P \| P \mid STOP_e$
process definitions	$(Defs)$	$Defs ::= X \mathrel{\widehat{=}} P, \ldots$

In $a(p)$ the length of p must be equal to $arity(a)$. As suggested by the syntactic symbols, an 'event set' e will always be interpreted as really a *set*; however, in concrete examples we will just enumerate the elements and not use the abstract syntax.

The notation $\frac{e}{e'}{\to}P$ ("prefix") is the basic means to express the steps that processes can make. Process $\frac{e}{e'}{\to}P$ is sensitive to, roughly said, $e \cup e'$ (irrespective of P); it is ready to participate in an event set that at least contains e and does not contain the events from e'; when it makes such a step, it evolves to P. In $P = \frac{e}{e'}{\to}P'$ we call event set e the *positive prefix* of P, and e' the *negative prefix*. During the step, the choice parameters in e get a value assigned, and this assignment permeates the evolving process. However, the choice parameters in e' stand for an arbitrary value and provide a shorthand for what otherwise would require a kind of set comprehension (or a huge enumeration).

Process $P{+}Q$ ("choice") is ready to make steps that P or Q can do, and evolves then to the process that P or Q, respectively, evolves to, the choice being nondeterministic. Process $P \| Q$ ("communication") makes the steps of P and Q simultaneously; but if P is insensitive to the

event set of the step, then P stays the same and only Q makes the step and evolves; and similarly with the roles of P and Q interchanged. So if P, say, *is* sensitive for e but cannot make a step with e, then the entire composition $P \| Q$ cannot make an e-step: component P blocks e-steps of the composition.

Process $STOP_e$ ("stop") cannot make a step at all; it is, however, only sensitive to events in e, and therefore will not block everything. Process $\rho \bullet P$ ("let") behaves the same as P except that the variables are assigned the values as prescribed by ρ; it behaves the same as P in which x is *substituted* by $\rho\, x$, for every x in $\mathrm{dom}\, \rho$. An alternative, less abstract and more verbose syntax for $\rho \bullet P$ is the let-notation **let** ρ **in** P. This form of process is only needed to express life cycles in our process model (which we won't do here).

Throughout the sequel, when we discuss a process, we assume a set *Defs* of process definitions be given, which we refer to as *the* process definitions. We define function *body* by putting $body\ X = P$ for each definition $X \mathrel{\widehat{=}} P$ in *Defs*.

We let $\otimes$ vary over $+$ and $\|$: for most syntactic definitions these two operators behave the same. We only formally discuss the binary composition $+$, although we actually intend to treat the n-ary and infinite version of the operation as well: $\sum$.

Skip this remark about Notation. In order to keep the notation and proof manipulations light-weight, we have not introduced an explicit notation for recursion, for example something like $\mu\, X \bullet P$. In several manipulations (like saturation, §18), there arises a potentially large number of mutually recursive process definitions, and writing the recursion explicitly with something like $\mu\, X \bullet P$ is not easy nor useful.

Skip this remark about Life Cycles. Life cycles are special versions of these concepts:

Val	the life identities
life, state	process in which no values occur, and
	no $?this$ but only $this$ (postulated to be a variable), and
	no $\xrightarrow{e}$ but only $\xrightarrow{\emptyset}$, and also
	no $\|$, neither directly nor indirectly via the process definitions
life cycle system	$(\{this \mapsto v_0\} \bullet P) \ \| \ (\{this \mapsto v_1\} \bullet Q) \ \| \ldots$ where
	$v_0, v_1, \ldots$ are arbitrary distinct values, and $P, Q, \ldots$ are lifes.

7 Free variables and binding. Systematic renaming does not change the semantics of expressions. For example, $\frac{a(?x)}{b(x)} \rightarrow P(x)$ will get the same semantics assigned as $\frac{a(?y)}{b(y)} \rightarrow P(y)$. In order to formalise this phenomenon we need the notions of *binding*, and *free* and *choice* variables.

The *choice parameters* $Ch(_)$ of a parameter list and event set are the variables labelled with a question mark. In an expression $\frac{e}{e'} \rightarrow P$, the occurrences of $?x$ within e jointly *bind* each other as well as the occurrences of x in e' and P (and *not* the x in e itself), and the occurrences of $?x$ in e' only bind each other (and *not* the x in e' itself or in P). So, these two process expressions get the same semantics:

$$\frac{a(?x), b(x), c(?x)}{d(x), e(?x)} \rightarrow P(x) \qquad \text{and} \qquad \frac{a(?y), b(x), c(?y)}{d(y), e(?z)} \rightarrow P(y)$$

Similarly, the parameters $?x$ in the e of $STOP_e$ jointly bind each other. In $\rho \bullet P$, the variables x in $\mathrm{dom}\, \rho$ bind the occurrences of x in P. The *free* variable occurrences are those that are not bound. Formalised this yields the following definitions:

p	$Ch\ p$	$Fr\ p$
x	$\emptyset$	$\{x\}$
$?x$	$\{x\}$	$\emptyset$
v	$\emptyset$	$\emptyset$
$p, \ldots$	$Ch\ p \cup \ldots$	$Fr\ p \cup \ldots$

e	$Ch\ e$	$Fr\ e$
$\emptyset$	$\emptyset$	$\emptyset$
$\{a(p)\}$	$Ch\ p$	$Fr\ p$
$e \cup e'$	$Ch\ e \cup Ch\ e'$	$Fr\ e \cup Fr\ e'$

P	$Fr\,P$
X	$Fr(body\,X)$
$\frac{e}{e'}\!\rightarrow\! P$	$Fr\,e \cup (Fr\,e' \cup Fr\,P) \setminus Ch\,e$
$P \otimes Q$	$Fr\,P \cup Fr\,Q$
$STOP_e$	$Fr\,e$
$\rho \bullet P$	$Fr\,P \setminus \mathrm{dom}\,\rho$

Note that $Ch(_)$ and $Fr(_)$ yield disjoint sets.

We claim, but do not formally prove, that consistent renaming of the bound variable occurrences will not change processes in an essential way: most syntactic and semantic manipulations commute with consistent renaming.

8 Note on Recursion. Since X may occur in $body\,X$, the definition $Fr\,X = Fr(body\,X)$ is circular, or rather, it is merely a fixed point equation. There may be several solutions for Fr that satisfy the equations for Fr. To fix just one particular solution, we stipulate that, for any P, the set $Fr\,P$ is least with respect to set inclusion. This means that Fr itself is least with respect to the ordering $\preceq$ defined by: $f \preceq g \Leftrightarrow \forall\, x \bullet f\,x \subseteq g\,x$. The least solution may be *defined* without circularity as follows:

> Take $Fr_0\,P = \varnothing$ for all P, and define $Fr_{i+1}\,P$ precisely as above but with 'Fr' (in the right-hand side) replaced by 'Fr_i', and then define $Fr\,P = \bigcup_i Fr_i\,P$.

Roughly said, the least solution may be *computed* by taking $Fr\,X$ to be "$\varnothing$ within the body and $Fr(body\,X)$ elsewhere". All this can be formalised and proved by standard fixed point theory, as explained, for instance, by Winskel [10].

The same remark applies to the coming definitions of the set-valued functions α (§9) and β (§11).

Skip this remark about Syntax. An expression that has no free variables is called *closed*. As a syntactic constraint, we require that the entire, *root*, process is closed. For example, in the definitions below, the free x of $STOP_{c(x)}$ must be bound in every path to $STOP_{c(x)}$. This is the case for P, but not for Q:

$$P \;=\; \tfrac{a(?x)}{\varnothing}\!\rightarrow\! STOP_{c(x)} + \tfrac{b(?x)}{\varnothing}\!\rightarrow\! STOP_{c(x)} \qquad \text{O.K., since } Fr\,P = \varnothing$$

$$Q \;=\; \tfrac{a(?x)}{\varnothing}\!\rightarrow\! STOP_{c(x)} + \tfrac{b'(\ )}{\varnothing}\!\rightarrow\! STOP_{c(x)} \qquad \text{K.O., since } Fr\,Q = \{x\} \neq \varnothing$$

Skip this remark about Definition. In the clause for $\frac{e}{e'}\!\rightarrow\! P$, the choice variables of e' do *not* bind variables in P; they are only used to indicate a set of possibilities, as apparent from the saturation of events (§10). If for some reason it would be required that the choice variables in e' bind variables in P, then we should replace, in that clause, part $Fr\,P$ by $(Fr\,P \setminus Ch\,e')$.

9 Alphabet. The alphabet αP of a process P is the set of events that might turn up in some evolution of the process. Formally, for each P the set αP is the least set such that the following equations hold:

P	$\alpha\,P$
X	$\alpha(body\,X)$
$\frac{e}{e'}\!\rightarrow\! P$	$e \cup e' \cup \alpha P$
$P \otimes Q$	$\alpha P \cup \alpha Q$
$STOP_e$	e
$\rho \bullet P$	$(\alpha P)[\rho]$

In the latter line the notation $e[\rho]$ means: e in which x is replaced by ρx, for every x in $\mathrm{dom}\,\rho$.

Skip this remark about Substitution. We have not introduced a notation for substitution in processes since "$P[\rho]$" would misleadingly suggest that only P is affected whereas a proper treatment would also affect the process definitions. In the sequel, we can do without a notion of substitution on processes.

Skip this remark about the Role of the Alphabet. For the time being, the alphabet plays no role at all. We have presented it here only to contrast this well-known syntactic concept from conventional process models with our concept of sensitivity set.

10 Saturation of events. An expression is called *saturated* if it has no occurrences of variables, neither x nor $?x$. So, a saturated expression is closed. Here we define the saturation $[\![_]\!]_\rho$ that transforms events to saturated events, in an environment ρ; later on we deal with processes (§18). In essence, saturation is nothing more than the substitution of value $\rho\, x$ for the free occurrences of x, for each x in $\mathrm{dom}\,\rho$. However, choice parameters $?x$ may get an arbitrary value, but the same value for all occurrences of the same choice parameter. To express this, we use an another saturation function that depends also on the choice for the new environment σ, called a *supplementary* environment, which is written as a superscript: $[\![_]\!]^\sigma_\rho$. The main saturation $[\![_]\!]_\rho$, then, takes all possible σ into account:

p	$[\![p]\!]^\sigma_\rho$
x	$\rho\, x$
$?x$	$\sigma\, x$
v	v
$p, \ldots$	$[\![p]\!]^\sigma_\rho, \ldots$

e	$[\![e]\!]^\sigma_\rho$
$\varnothing$	$\varnothing$
$\{a(p)\}$	$\{a([\![p]\!]^\sigma_\rho)\}$
$e \cup e'$	$[\![e]\!]^\sigma_\rho \cup [\![e']\!]^\sigma_\rho$

e	$[\![e]\!]_\rho$
e	$\bigcup_{\sigma \mid \mathrm{dom}\,\sigma = Ch\ e} [\![e]\!]^\sigma_\rho$

Terms $[\![p]\!]_\rho$ and $[\![p]\!]^\sigma_\rho$ only make sense if $Fr\ p \subseteq \mathrm{dom}\,\rho$ and $Ch\ p \subseteq \mathrm{dom}\,\sigma$, and similarly for $[\![e]\!]_\rho$ and $[\![e]\!]^\sigma_\rho$. In the sequel we shall take care that these requirements are met. Saturation $[\![_]\!]^\sigma_\rho$ is not only a locally used auxiliary function; in transition rule $[\rightarrow]$ (§14) it is used as an independent function to interpret the positive prefixes (thereby establishing a supplementary environment σ for an inner part of a process).

Here is an example of saturation. Assuming $Val = \{v_0, v_1, \ldots\}$, then:

$$[\![a(x), b(?x), c(x), d(?y)]\!]_{\{x \mapsto v\}} \quad = \quad \{a(v), b(v_0), b(v_1), \ldots\, c(v), d(v_0), d(v_1), \ldots\}$$

Notice that difference in choice variables does matter within a single event, but not in distinct events of an event set:

$$[\![a(?x), b(?y)]\!]_\rho = \{v, w : Val \bullet a(v), b(w)\} = [\![a(?x), b(?x)]\!]_\rho$$
$$[\![a(?x, ?y)]\!]_\rho = \{v, w : Val \bullet a(v, w)\} \neq \{v : Val \bullet a(v, v)\} = [\![a(?x, ?x)]\!]_\rho$$

C Semantics

11 Sensitivity set. The *sensitivity set* of a process is the set of events for which the process is sensitive: for each event in the sensitivity set, the process either can participate in the event or it blocks the event. (This is formalised later in the transition steps: §14.) The sensitivity set may vary as the process evolves; this is distinctive for the life cycle approach in comparison to conventional process theories (where only a fixed alphabet plays a role). The set is defined by straightforward induction for composite processes. For process $\frac{e}{e'}\!\rightarrow P$, the sensitivity set consists, roughly said, of $e \cup e'$, independently of P. Formally, for each P and ρ the set $\beta_\rho P$ is the least one such that the following equations hold:

P	$\beta_\rho P$
X	$\beta_\rho(body\ X)$
$\frac{e}{e'}\!\rightarrow P$	$\bigcup_{\sigma \mid \mathrm{dom}\,\sigma = Ch\ e} [\![e]\!]^\sigma_\rho \cup [\![e']\!]_{\rho \oplus \sigma}$
$P \otimes Q$	$\beta_\rho P \cup \beta_\rho Q$
$STOP_e$	$[\![e]\!]_\rho$
$\sigma \bullet P$	$\beta_{\rho \oplus \sigma} P$

In the clause for $\frac{e}{e'}\!\rightarrow P$, the parameters $?x$ in e stand for an arbitrary value but the same one for all occurrences of the same $?x$, and the x in e' also denotes that value; this intention is taken care of by the use of the supplementary environment σ.

12 Transition steps. The main semantics of a process P is given by the transition steps it can make, denoted:

$$P \quad \xrightarrow{e}_{\rho,\sigma} \quad Q$$

Process Q is the residual process that results after the step. Set e is the event set that makes the step happen. We deliberately stipulate that such an event set is saturated, that is, it contains no variables (neither x nor $?x$). Environment ρ says what variables denote what values; it should at least do so for all free variables of P. Environment σ says what the choices are for the choice variables; the step thus establishes a supplementary environment for the residual process Q, and $\mathrm{dom}(\rho \oplus \sigma)$ will contain all free variables of Q. (The actual usage of the supplementary environment occurs in the combination of one step after the other: §16.)

We present the definition of the transition steps as an inference system.

Skip this remark about Sensefulness. In order that (by induction) the terms $[\![\ldots]\!]_\rho$ make sense in the side-conditions of the rules, it is required that in each rule, with conclusion $P \xrightarrow{e}_{\rho,\sigma} Q$ say, we have $Fr\, P \subseteq \mathrm{dom}\, \rho$. The reader may observe that if this requirement for the conclusion is true, then so is the corresponding property for the premise(s) of the rule.

In rule $[\rightarrow]$ in §14, one immediately sees that $[\![e]\!]^\sigma_{\cdots}$ in the side-condition makes sense, since $\mathrm{dom}\, \sigma = Ch\, e$.

13 Note on Data Flow. (For those who know the concepts, environment ρ is an *inherited* attribute of the rules, whereas σ is a *synthesised* attribute: the ρ in the conclusion determines the ρ in the premise, and for σ it is just the other way around.)

14 Single steps. The steps that a name X can make are those of its body:

$$[\widehat{=}]\frac{body\ X \xrightarrow{e}_{\rho,\sigma} P}{X \xrightarrow{e}_{\rho,\sigma} P}$$

Process $\frac{e}{e'}{\rightarrow}P$ can make a step with event set e'' if that set contains at least all the events of the positive prefix e of the process and, at the same time, does not contain the events that are blocked according to negative prefix e':

$$[\rightarrow]\frac{}{\frac{e}{e'}{\rightarrow}P \quad \xrightarrow{e''}_{\rho,\sigma} \quad P} \begin{cases} \mathrm{dom}\, \sigma = Ch\, e \\ e'' \cap [\![e']\!]_{\rho\oplus\sigma} = \varnothing \wedge [\![e]\!]^\sigma_\rho \subseteq e'' \end{cases}$$

For example, consider:

$$\frac{e}{e'}{\rightarrow}\cdots \qquad \text{where} \quad e = a(?x), \quad e' = a(w), b(x)$$

Suppose the process participates in some e'' with the choice $\sigma = \{x \mapsto v\}$ for variable x. Then, firstly, the condition $[\![e]\!]^\sigma_\rho \subseteq e''$ says that e'' contains at least $a(v)$: the positive prefix e with the substitution $x \mapsto v$. And secondly, the condition $e'' \cap [\![e']\!]_{\rho\oplus\sigma} = \varnothing$ says that e'' contains nothing from $[\![\{a(w), b(x)\}]\!]_{\rho\oplus\sigma}$, that is, e'' does neither contain $a(w)$ nor $b(v)$. So, the step can only take place if $v \neq w$. Note also that process $\frac{a(?x)}{a(x)}{\rightarrow}P$ cannot make a step at all.

A choice can do a step if either operand can do:

$$[+_{left}]\frac{P \xrightarrow{e}_{\rho,\sigma} P'}{P{+}Q \xrightarrow{e}_{\rho,\sigma} P'} \qquad [+_{right}]\frac{Q \xrightarrow{e}_{\rho,\sigma} Q'}{P{+}Q \xrightarrow{e}_{\rho,\sigma} Q'}$$

A communication can do a step if both components can do the step. In order that the separately established supplementary environments σ and τ work as expected on the residual composite, we assume a suitable naming convention. This convention and its effect on the rule here, is explained in detail below in §15.

$$[\|_{both}]\frac{P \xrightarrow{e}_{\rho,\sigma} P' \qquad Q \xrightarrow{e}_{\rho,\tau} Q'}{P \| Q \xrightarrow{e}_{\rho,\ \sigma\cup\tau} P' \| Q'} \qquad \text{(Naming convention: see below)}$$

Alternatively, a communication can do a step if one component can do the step and the other is insensitive to the event set (and *ignores* it):

$$[\|_{left}]\frac{P \xrightarrow{e}_{\rho,\sigma} P'}{P\|Q \xrightarrow{e}_{\rho,\sigma} P'\|Q}\Big\{ e \text{ disjoint-from } \beta_\rho Q \qquad \text{(Naming convention: see below)}$$

$$[\|_{right}]\frac{Q \xrightarrow{e}_{\rho,\tau} Q'}{P\|Q \xrightarrow{e}_{\rho,\tau} P\|Q'}\Big\{ e \text{ disjoint-from } \beta_\rho P \qquad \text{(Naming convention: see below)}$$

Process $\sigma \bullet P$ behaves as P, except that the variables of $\operatorname{dom}\sigma$ are to be interpreted as the values as given by σ, both for P and for the residual process after the step:

$$[let]\frac{P \xrightarrow{e}_{\rho\oplus\sigma,\,\tau} P'}{\sigma \bullet P \xrightarrow{e}_{\rho,\,\sigma\oplus\tau} P'}$$

Process $STOP_e$ cannot do a step at all; so, there is no inference rule for it.

15 Naming convention. In order to avoid awkward syntactic transformations, we have assumed in the rule for communication that no name clash occurs. This can always be achieved by suitable renaming.

More precisely, in rule $[\|_{both}]$ we have assumed that $Fr\,P \cup Bnd\,P$ is disjoint from $Bnd\,Q$, and conversely, $Bnd\,P$ is disjoint from $Fr\,Q \cup Bnd\,Q$. Here $Bnd\,P$ is the set of bound variables of P, or more economically, a subset containing at least those variables that may get a value assigned in the very first transition step of P (that is, $\operatorname{dom}\sigma$). As a consequence of this assumption we have that $\operatorname{dom}\sigma$ is disjoint from $\operatorname{dom}\tau$ so that $\sigma\cup\tau$ is a well defined function, and $Fr\,P$ is disjoint from $\operatorname{dom}\tau$, and $\operatorname{dom}\sigma$ from $Fr\,Q$, so that later on (in a sequence of steps, or in a saturation) the free variables of P are not interpreted by τ, and the free variables of Q not by σ. These consequences are also put into use in the formal proof (§28, case $P_1\|P_2$).

Similarly for rules $[\|_{left}]$ and $[\|_{right}]$.

16 Sequences of steps. A sequence of event sets may result in several steps in succession. We let s, t vary over *sequences* of event sets, and denote a sequence of steps by the same arrow symbol as a single step.

No step at all can always be observed, and makes no choices:

$$[none]\frac{}{P \xrightarrow{\langle\rangle}_{\rho,\emptyset} P}$$

A sequence with just one event set is the same as a single step:

$$[single]\frac{P \xrightarrow{e}_{\rho,\sigma} Q}{P \xrightarrow{\langle e\rangle}_{\rho,\sigma} Q}$$

In a sequence of steps, each newly established supplementary environment is in effect for the subsequent steps, and the accumulation of these environments is the newly established supplementary environment of the whole sequence:

$$[comp]\frac{P \xrightarrow{s}_{\rho,\sigma} Q \qquad Q \xrightarrow{t}_{\rho\oplus\sigma,\,\tau} R}{P \xrightarrow{st}_{\rho,\,\sigma\oplus\tau} R}$$

D Eliminating parameters

17 The usefulness of saturation. In this section we define the saturation $[\![_]\!]_\rho$ of processes, which eliminates all variables from a process, and we show that saturation can be factored out from other definitions on processes. Roughly said, it means that "other interesting functions" f (or relations) on processes, can be written as $f' \circ [\![_]\!]_\rho$, where f' is a variant of f with a simpler definition (simpler because its arguments are saturated and no longer arbitrary). More precisely, sometimes it is not the very f that equals $f' \circ [\![_]\!]_\rho$, but "f followed by saturating its result", i.e., $[\![_]\!]_{\rho'} \circ f$. This property is also known as *commutativity* of the following diagram:

$$\begin{array}{ccc} . & \xrightarrow{f} & . \\ {\scriptstyle [\![_]\!]_\rho}\downarrow & & \downarrow{\scriptstyle id} \\ . & \xrightarrow{f'} & . \end{array} \qquad \text{respectively} \qquad \begin{array}{ccc} . & \xrightarrow{f} & . \\ {\scriptstyle [\![_]\!]_\rho}\downarrow & & \downarrow{\scriptstyle [\![_]\!]_{\rho'}} \\ . & \xrightarrow{f'} & . \end{array}$$

Alternatively, we call this property the *correctness* of saturation with respect to f and f'.

In addition to showing that the factorisation is *possible*, we also claim that the factorisation is *easy* (that is, the definition of $[\![_]\!]_\rho$ is at most as complicated as the definitions of the other interesting functions f) and that the resulting simplification of the other interesting functions is *substantial* (the definition of f' is simpler than f). These two claims cannot be proved formally, but are subjective judgements. We do *not* claim that the factorisation is helpful to achieve an efficient implementation.

Thanks to the possibility of factorisation, it may be beneficial to begin a discussion of the "interesting f" by assuming that "parameters have been eliminated in favour of sums", so that f' can be dealt with instead.

18 Saturation of processes. The saturation for events has been given earlier, in §10. Here is the saturation of processes (for ρ satisfying $Fr\ P \subseteq \operatorname{dom}\rho$):

P	$[\![P]\!]_\rho$
X	X_ρ
$\frac{e}{e'}\!\rightarrow\! P$	$\sum_{\sigma \mid \operatorname{dom}\sigma = Ch\ e} \frac{[\![e]\!]^\sigma_\rho}{[\![e']\!]_{\rho\oplus\sigma}}\!\rightarrow\![\![P]\!]_{\rho\oplus\sigma}$
$P \otimes Q$	$[\![P]\!]_\rho \otimes [\![Q]\!]_\rho$
$STOP_e$	$STOP_{[\![e]\!]_\rho}$
$\sigma \bullet P$	$[\![P]\!]_{\rho\oplus\sigma}$

Name X_ρ is a fresh name; and the definition $X_\rho \mathrel{\widehat{=}} [\![body\ X]\!]_\rho$ has to be added to the definitions of the saturated process. Different occurrences of X give rise to X_ρ for several ρ, and thus to several process definitions.

19 Note on Recursion. Recursion is *not* problematic here: after saturation infinitely many equally shaped definitions $X_{\rho'} \mathrel{\widehat{=}} [\![body\ X]\!]_{\rho'}$ exist, but all ρ' have a finite domain (provided the initial ρ is finite).

20 Sensitivity set. For saturated processes, the definition of the sensitivity set simplifies a little bit:

P	βP
X	$\beta(body\ X)$
$\frac{e}{e'}\!\rightarrow\! P$	$e \cup e'$
$P \otimes Q$	$\beta P \cup \beta Q$
$STOP_e$	e

The use of the simple definition after saturation yields the same result as the complex definition followed by saturation:

21 Theorem (Correctness of saturation I) *Let $Fr\, P \subseteq \mathrm{dom}\, \rho$. Then:*

$$\beta_\rho P \quad = \quad \beta([\![P]\!]_\rho)$$

The left-hand β_ρ is the original one from §11; the right-hand β is the new, simpler one from §20. In other words, the following diagram commutes:

$$\begin{array}{ccc} P & \xrightarrow{\beta_\rho} & . \\ [\![_]\!]_\rho \downarrow & & \downarrow id \\ . & \xrightarrow{\beta} & . \end{array}$$

□

Proof

The proof of the theorem is by fixed point induction.

22 Note on Scott's Fixed Point Induction. Let $\mathcal{B}\beta$ be the right-hand side of the definition of β, so that the definition of β reads: $\beta = \mathcal{B}\beta$. As remarked at the definition of *Fr* in §8, the least fixed point of the equation $\beta = \mathcal{B}\beta$ equals $\bigcup_i \beta_i$, where $\beta_0 = \lambda\, P.\emptyset$, and $\beta_{i+1} = \mathcal{B}\beta_i$. For such a function, the following *fixed point induction* principle holds (consult for example Winskel [10] for the underlying theory):

> Suppose ('Base') property $\mathcal{P}$ holds of function $\beta := \lambda\, P.\emptyset$ and
> suppose ('Step') $\forall\, \beta \bullet \mathcal{P}(\beta) \Rightarrow \mathcal{P}(\mathcal{B}\beta)$.
> Then ('Conclusion') $\mathcal{P}$ holds of $\beta := \bigcup_i \beta_i$.

This is only true of 'admissible' predicates $\mathcal{P}$. Equality of functions is admissible.

The principle generalises to two functions β instead of one; symbolically:

$$\mathcal{P}((\lambda\, P.\emptyset), (\lambda\, P.\emptyset)) \wedge (\forall\, \beta_1, \beta_2 \bullet \mathcal{P}(\beta_1, \beta_2) \Rightarrow \mathcal{P}(\mathcal{B}_1\beta_1, \mathcal{B}_2, \beta_2)) \quad \Rightarrow \quad \mathcal{P}(\textstyle\bigcup_i \beta_{1,i}, \bigcup_i \beta_{2,i})$$

We shall apply this principle, with the old and new function β. In order not to clutter the formulas with distracting squiggles, we do not distinguish the two functions β notationally and instead write a dotted line to separate each calculation into two parts: above the dots the *old* function β is used and below the dots the *new* β is used.

The reader may observe that induction on the structure of the process expression will not succeed, since *body X* is not a sub-expression of process name X.

We first consider the 'Base' where each β is defined to be $\lambda\, P.\emptyset$:

$$\begin{array}{ll} \beta_\rho P & \\ = \emptyset & \text{[base case assumption on } \beta_\rho\text{]} \\ \cdots\cdots\cdots & \cdots\cdots\cdots \\ = \beta([\![P]\!]_\rho) & \text{[base case assumption on } \beta\text{]} \end{array}$$

Next the 'Step': for arbitrary β_ρ and β (an old and a new one) with $\forall\, P \bullet \beta_\rho P = \beta([\![P]\!]_\rho)$ (the *induction hypothesis*), we put $\beta'_\rho = \mathcal{B}\beta_\rho$ and $\beta' = \mathcal{B}\beta$, and show $\beta'_\rho\, P = \beta'\, ([\![P]\!]_\rho)$. We distinguish the forms that the process expression P can take. In case the expression is X, then:

$$\begin{array}{ll} \beta'_\rho\, X & \\ = \beta_\rho(body\ X) & \text{[defn } \beta'_\rho\text{]} \\ \cdots\cdots\cdots & \cdots\cdots\cdots \\ = \beta([\![body\ X]\!]_\rho) & \text{[induction hypothesis, taking } P := body\ X\text{]} \\ = \beta'(X_\rho) & \text{[defn } \beta'\text{, defn } X_\rho \mathrel{\widehat{=}} [\![body\ X]\!]_\rho\text{]} \end{array}$$

$$= \beta'([\![X]\!]_\rho) \qquad \text{[defn } [\![_]\!]_\rho\text{]}$$

In case the expression is $\frac{e}{e'}{\rightarrow}P$, then (without using the induction hypothesis):

$$\begin{array}{ll}
\beta'_\rho(\frac{e}{e'}{\rightarrow}P) & \\
= \bigcup_{\sigma|\mathrm{dom}\,\sigma=Ch\,e}([\![e]\!]^\sigma_\rho \cup [\![e']\!]_{\rho\oplus\sigma}) & \text{[defn } \beta'_\rho\text{]} \\
\cdots\cdots\cdots\cdots & \cdots\cdots\cdots\cdots \\
= \bigcup_{\sigma|\mathrm{dom}\,\sigma=Ch\,e}\beta'(\frac{[\![e]\!]^\sigma_\rho}{[\![e']\!]_{\rho\oplus\sigma}}{\rightarrow}P) & \text{[defn } \beta'\text{]} \\
= \beta'(\sum_{\sigma|\mathrm{dom}\,\sigma=Ch\,e}\frac{[\![e]\!]^\sigma_\rho}{[\![e']\!]_{\rho\oplus\sigma}}{\rightarrow}P) & \text{[defn } \beta' \text{ (on generalised } +)\text{]} \\
= \beta'([\![\frac{e}{e'}{\rightarrow}P]\!]_\rho) & \text{[defn } [\![_]\!]_\rho\text{]}
\end{array}$$

In case the expression is $P \otimes Q$, then:

$$\begin{array}{ll}
\beta'_\rho(P \otimes Q) & \\
= \beta_\rho P \cup \beta_\rho Q & \text{[defn } \beta'_\rho\text{]} \\
\cdots\cdots\cdots\cdots & \cdots\cdots\cdots\cdots \\
= \beta([\![P]\!]_\rho) \cup \beta([\![Q]\!]_\rho) & \text{[induction hypothesis]} \\
= \beta'([\![P]\!]_\rho \otimes [\![Q]\!]_\rho) & \text{[defn } \beta'\text{]} \\
= \beta'([\![P \otimes Q]\!]_\rho) & \text{[defn } [\![_]\!]_\rho\text{]}
\end{array}$$

In case the expression is $\sigma \bullet P$ or $STOP_e$, then the argument is similar.

End of Proof

23 Transition steps. Due to the absence of environments and the work that saturation has done in the clause for $\frac{e}{e'}{\rightarrow}P$, the inference rules simplify considerably, in particular for $\frac{e}{e'}{\rightarrow}P$. First, the steps that a name X can make are those of its body:

$$[\widehat{=}]\frac{body\,X \xrightarrow{e} P}{X \xrightarrow{e} P}$$

Next, process $\frac{e}{e'}{\rightarrow}P$ can make a step with event set e'' if that set contains at least all the events of the positive prefix e of the process and does not contain events from the negative prefix e':

$$[\rightarrow]\frac{}{\frac{e}{e'}{\rightarrow}P \xrightarrow{e''} P}\left\{\ e'' \cap e' = \varnothing \wedge e \subseteq e''\right.$$

A choice can do a step if either operand can:

$$[+_{left}]\frac{P \xrightarrow{e} P'}{P{+}Q \xrightarrow{e} P'} \qquad [+_{right}]\frac{Q \xrightarrow{e} Q'}{P{+}Q \xrightarrow{e} Q'}$$

A communication can do a step if both components can do the step, or one can do so and the other is insensitive to the event:

$$[\|_{both}]\frac{P \xrightarrow{e} P' \qquad Q \xrightarrow{e} Q'}{P\|Q \xrightarrow{e} P'\|Q'}$$

$$[\|_{left}]\frac{P \xrightarrow{e} P'}{P\|Q \xrightarrow{e} P'\|Q}\left\{e \text{ disjoint-from } \beta Q \qquad [\|_{right}]\frac{Q \xrightarrow{e} Q'}{P\|Q \xrightarrow{e} P\|Q'}\right\{e \text{ disjoint-from } \beta P$$

Process $STOP_e$ cannot do a step at all; so, there is no inference rule for it. After saturation, the let-form of processes has disappeared; so, there is no inference rule for that form, either.

Successive steps combine as in relational composition:

$$[none]\frac{}{P \xrightarrow{\langle\rangle} P} \qquad [single]\frac{P \xrightarrow{e} Q}{P \xrightarrow{\langle e\rangle} Q} \qquad [comp]\frac{P \xrightarrow{s} Q \qquad Q \xrightarrow{t} R}{P \xrightarrow{st} R}$$

In spite of the simplicity of the inference rules for saturated processes, the step relation is equivalent to the one with the complicated inference rules:

24 Theorem (Correctness of saturation II) *The following diagram commutes (reading "for some σ" along the top&right path):*

$$\begin{array}{ccc} P & \xrightarrow{s}_{\rho,\sigma} & \cdot \\ {\scriptstyle [\![-]\!]_\rho}\downarrow & & \downarrow{\scriptstyle [\![-]\!]_{\rho\oplus\sigma}} \\ \cdot & \xrightarrow{s} & Q \end{array}$$

Formally and more precisely, let $Fr\ P \subseteq \operatorname{dom}\rho$. Then:

$$(\exists\, \sigma, P' \bullet\ P \xrightarrow{s}_{\rho,\sigma} P' \xmapsto{[\![-]\!]_{\rho\oplus\sigma}} Q) \quad \Leftrightarrow \quad [\![P]\!]_\rho \xrightarrow{s} Q$$ □

The proof is by induction on the structure of the derivations, using Theorem 21 along the way.

Proof

The theorem is easily proved by induction on the length of sequence s, provided that the same claim holds with event set sequence s replaced by event set e:

25 $(\exists\, \sigma, P' \bullet\ P \xrightarrow{e}_{\rho,\sigma} P' \xmapsto{[\![-]\!]_{\rho\oplus\sigma}} Q) \quad \Leftrightarrow \quad [\![P]\!]_\rho \xrightarrow{e} Q$

In order to prove this claim we need the following lemma (for P, ρ_1, ρ_2 satisfying $Fr\ P \subseteq \operatorname{dom}\rho_1 \cap \operatorname{dom}\rho_2$):

26 $[\![P]\!]_{\rho_1} \quad = \quad [\![P]\!]_{\rho_2} \quad$ if $\forall\, x : Fr\ P \bullet \rho_1 x = \rho_2 x$

The lemma is proved by induction on the structure of P. We omit further details.

Now we prove claim 25 by induction on the structure of the derivations, using Theorem 21 along the way.

27 Note on the Proof Presentation. Formally, what we are going to prove is this:

> For each derivation Δ (whether in the old or new inference system) and for each e, ρ, P, Q:
>
> *if*, for some σ and P', derivation Δ ends with the conclusion $P \xrightarrow{e}_{\rho,\sigma} P'$ and $[\![P']\!]_{\rho\oplus\sigma} = Q$,
> *then* (there exists a derivation of) $[\![P]\!]_\rho \xrightarrow{e} Q$
> (thus establishing part $\Rightarrow$ of claim 25)
>
> and: *if* derivation Δ ends with the conclusion $[\![P]\!]_\rho \xrightarrow{e} Q$,
> *then*, for some σ and P', (there exists a derivation of) $P \xrightarrow{e}_{\rho,\sigma} P'$ and $[\![P']\!]_{\rho\oplus\sigma} = Q$
> (thus establishing part $\Leftarrow$ of claim 25)

We use induction on the structure of derivation Δ. In order not to introduce a lot of notation and concepts, we will be rather implicit about the derivations, and only mention explicitly the formulas derived. Nevertheless, when we make use of the induction hypothesis, the reader may check that it is applied to a sub-derivation of Δ.

In order to avoid duplication of several proof steps, we present the proofs of an implication and its converse as a chain of equivalence steps. The reader should be aware that the formula referred to by a justification 'induction hypothesis', depends on the direction ('$\Rightarrow$' or '$\Leftarrow$') in which the proof is read.

In our proof, induction on the structure of the process expression does not succeed since *body* X is not a sub-expression of process name X.

Before we delve into the details, let us remark that the rules of the inference system for the step relation are *implications*. For example:

$$P_1 \xrightarrow{e} Q \quad \Rightarrow \quad P_1 + P_2 \xrightarrow{e} Q \qquad \text{captures rule} \qquad [+_{left}]\frac{P_1 \xrightarrow{e} Q}{P_1+P_2 \xrightarrow{e} Q}$$

$$P_2 \xrightarrow{e} Q \quad \Rightarrow \quad P_1 + P_2 \xrightarrow{e} Q \qquad \text{captures rule} \qquad [+_{right}]\frac{P_2 \xrightarrow{e} Q}{P_1+P_2 \xrightarrow{e} Q}$$

These implications are referred to by 'rule $[+_{left}]$' and 'rule $[+_{right}]$', respectively. However, since $P \xrightarrow{e} Q$ holds if, and only if, there is a *finite* derivation of that formula, it so happens that the inference system determines several *equivalences*. For example:

$$P_1 \xrightarrow{e} Q \vee P_2 \xrightarrow{e} Q \quad \Leftrightarrow \quad P_1 + P_2 \xrightarrow{e} Q$$

We shall refer to such an equivalence by 'rule [+-iff]'. In order not to clutter the rule names with distracting squiggles, we do not notationally distinguish the rules of the old and new inference system, and instead write a dotted line to separate each calculation into two parts: above the dots the *new* inference rules are used and below the dots the *old* rules.

28 Proof of claim 25. We distinguish the form that process expression P can take. In case P is process name X, then:

$$\begin{array}{ll}
[\![X]\!]_\rho \xrightarrow{e} Q & \\
\Leftrightarrow X_\rho \xrightarrow{e} Q & [\text{defn } [\![_]\!]_\rho] \\
\Leftrightarrow [\![\mathit{body}\ X]\!]_\rho \xrightarrow{e} Q & [\text{rule } [\widehat{=}\text{-iff}], \text{property } [\![_]\!]_\rho] \\
\cdots\cdots\cdots & \cdots\cdots\cdots \\
\Leftrightarrow \exists\, \sigma, P' \bullet\ \mathit{body}\ X \xrightarrow[\rho,\sigma]{e} P' \xmapsto{[\![_]\!]_{\rho\oplus\sigma}} Q & [\text{induction hypothesis}] \\
\Leftrightarrow \exists\, \sigma, P' \bullet\ X \xrightarrow[\rho,\sigma]{e} P' \xmapsto{[\![_]\!]_{\rho\oplus\sigma}} Q & [\text{rule } [\widehat{=}\text{-iff}]]
\end{array}$$

In case P is $\frac{e_1}{e_2}{\rightarrow}P_0$, then (without using the induction hypothesis):

$$\begin{array}{ll}
[\![\frac{e_1}{e_2}{\rightarrow}P_0]\!]_\rho \xrightarrow{e} Q & \\
\Leftrightarrow (\sum_{\sigma \mid \mathrm{dom}\,\sigma = Ch\ e_1} \frac{[\![e_1]\!]^\sigma_\rho}{[\![e_2]\!]_{\rho\oplus\sigma}}{\rightarrow}[\![P_0]\!]_{\rho\oplus\sigma}) \xrightarrow{e} Q & [\text{defn } [\![_]\!]_\rho] \\
\Leftrightarrow \exists\, \sigma \mid \mathrm{dom}\,\sigma = Ch\ e_1 \bullet\ \frac{[\![e_1]\!]^\sigma_\rho}{[\![e_2]\!]_{\rho\oplus\sigma}}{\rightarrow}[\![P_0]\!]_{\rho\oplus\sigma} \xrightarrow{e} Q & [\text{rule } [+\text{-iff}]] \\
\Leftrightarrow \exists\, \sigma \mid \mathrm{dom}\,\sigma = Ch\ e_1 \bullet\ e \cap [\![e_2]\!]_{\rho\oplus\sigma} = \varnothing \wedge [\![e_1]\!]^\sigma_\rho \subseteq e \wedge Q = [\![P_0]\!]_{\rho\oplus\sigma} & [\text{rule } [\rightarrow\text{-iff}]] \\
\cdots\cdots\cdots & \cdots\cdots\cdots \\
\Leftrightarrow \exists\, \sigma \bullet\ \mathrm{dom}\,\sigma = Ch\ e_1 \ \wedge\ e \cap [\![e_2]\!]_{\rho\oplus\sigma} = \varnothing \wedge [\![e_1]\!]^\sigma_\rho \subseteq e \wedge Q = [\![P_0]\!]_{\rho\oplus\sigma} & [\text{pred. logic}] \\
\Leftrightarrow \exists\, \sigma, P' \bullet\ (\frac{e_1}{e_2}{\rightarrow}P_0) \xrightarrow[\rho,\sigma]{e} P' \xmapsto{[\![_]\!]_{\rho\oplus\sigma}} Q & [\text{one-point rule } (P' = P_0), \text{ rule } [\rightarrow\text{-iff}]]
\end{array}$$

In case P is $P_1 + P_2$, then:

$$\begin{array}{ll}
[\![P_1 + P_2]\!]_\rho \xrightarrow{e} Q & \\
\Leftrightarrow [\![P_1]\!]_\rho + [\![P_2]\!]_\rho \xrightarrow{e} Q & [\text{defn } [\![_]\!]_\rho]
\end{array}$$

$\Leftrightarrow [\![P_1]\!] \xrightarrow{e} Q \ \vee \ [\![P_2]\!]_\rho \xrightarrow{e} Q$ [rule [+-iff]]

. . . .

29 Usefulness of the translation. For conventional processes there exist several tools. In principle, these can now be used for our processes (and our life cycles!) as well. However, since the expressions that result from saturation and translation are quite large in the number of '$e{\rightarrow}P$' parts, the use of conventional tools will only succeed for *toy* processes.

30 Syntax of conventional processes. Let E be a set of so-called *action*s; we let e vary over E, and E' over sub-sets of E. The abstract syntax for conventional processes reads as follows:

process	(P, Q, R)	$P ::= X \mid e{\rightarrow}P \mid P{+}P \mid P \| P \mid \mathit{STOP}_{E'}$
process definitions	(Defs)	$\mathit{Defs} ::= X \mathrel{\widehat{=}} P, \ldots$

The alphabet αP of a conventional process is defined thus:

P	αP
$e{\rightarrow}P$	$\{e\} \cup \alpha P$
$P \otimes Q$	$\alpha P \cup \alpha Q$
$\mathit{STOP}_{E'}$	E'
X	$\alpha(\mathit{body}\ X)$

31 Note on Syntax. In Hoare's CSP, it is a syntactic requirement that, except for a communication, the constituents of a process have the same alphabet as the composite itself (so that $\alpha(e{\rightarrow}P) = \alpha P$ and $\alpha(P + Q) = \alpha P = \alpha Q$). This will be trivially true for the processes that result from the translation, since they all have E as their alphabet.

32 Transition steps of conventional processes. The inference system for conventional processes is, again, a little simpler than what we had before. First, the steps that a name X can make are those of its body, exactly as before:

$$[\widehat{=}]\frac{\mathit{body}\ X \xrightarrow{e} P}{X \xrightarrow{e} P}$$

Next, process $e{\rightarrow}P$ can (and can only) make a step with action e:

$$[\rightarrow]\frac{}{e{\rightarrow}P \xrightarrow{e} P}$$

A choice can do a step if either operand can do, exactly as before:

$$[+_{\mathit{left}}]\frac{P \xrightarrow{e} P'}{P{+}Q \xrightarrow{e} P'} \qquad [+_{\mathit{right}}]\frac{Q \xrightarrow{e} Q'}{P{+}Q \xrightarrow{e} Q'}$$

A communication can do a step with some action if both components can do the step:

$$[\|]\frac{P \xrightarrow{e} P' \qquad Q \xrightarrow{e} Q'}{P \| Q \xrightarrow{e} P' \| Q'}$$

Alternatively, a communication can do the step if one component can do so and the other has the action not in its alphabet. But since all processes that will result from our translation, have E (the set of all event sets) as their alphabet, this case never applies in the sequel.

Process $\mathit{STOP}_{E'}$ cannot do a step at all; so, there is no inference rule for it. Successive steps combine exactly as before, see §23.

33 Translation. The translation function is denoted $\mathcal{T}$; it eliminates negative prefixes from saturated processes, thus mapping saturated processes to conventional ones. Translation $\mathcal{T}$ will be defined together with an auxiliary function $\mathcal{S}$ by mutual recursion.

Recall that in the grammar for conventional processes a set E has been postulated, being the set of *action*s. Now we take E to be the set of *all saturated event sets*; so, a saturated event *set* is a *single* action. The alphabet of a translated process will be E.

The effect of $\mathcal{T}$ on a process P is twofold. First, it adds "a step to itself" with action e for every e for which P is insensitive. Second, a prefix form $\frac{e_1}{e_2}\!\!\rightarrow\!...$ is replaced by a summation over all events sets (actions!) e in which the process may participate: all e for which $e \cap e_2 = \varnothing \wedge e_1 \subseteq e$. The first of these effects is immediate from the right hand side of the definition of $\mathcal{T}$: by induction for a communication and explicitly written out for all other forms. The second effect of the translation is delegated to auxiliary function $\mathcal{S}$ (which, in turn, calls $\mathcal{T}$ recursively). The translation even has a third and fourth effect. The third effect is a syntactic trick to make *each* outcome having alphabet E: just add a semantically uninteresting summand $STOP_E$ at appropriate places. The fourth effect is a syntactic trick to make the (syntactic!) translation *injective* (which we need in the proof below), by adding semantically uninteresting summands $STOP_e$ at appropriate places.

Here is the definition:

$$\begin{array}{lcll} \mathcal{T}P & = & \mathcal{T}P_0 \| \mathcal{T}P_1, & \text{if } P = P_0 \| P_1 \text{ for some } P_0, P_1 \\ & = & X \text{ with } X \mathrel{\widehat{=}} (\sum\limits_{e:E \mid e \cap \beta P = \varnothing} e\!\rightarrow\! X) + \mathcal{S}P, & \text{otherwise (where } X \text{ is a fresh name)} \end{array}$$

and

P	$\mathcal{S}P$
$\frac{e_1}{e_2}\!\!\rightarrow\! P$	$(\sum_{e:E \mid e \cap e_2 = \varnothing \wedge e_1 \subseteq e} e\!\rightarrow\! \mathcal{T}P) + STOP_E + STOP_{e1} + STOP_{e2}$
$P + Q$	$\mathcal{S}P + \mathcal{S}Q$
$P \| Q$	$\mathcal{T}P \| \mathcal{S}Q + \mathcal{S}P \| \mathcal{T}Q$
$STOP_e$	$STOP_E + STOP_e$
X	X

In the $\mathcal{T}$-translation of a system, the process definitions $X \mathrel{\widehat{=}} body\ X$ have to be transformed into $X \mathrel{\widehat{=}} \mathcal{S}(body\ X)$. Note that if the right hand side of $\mathcal{S}(P \| Q)$ is replaced by $\mathcal{T}P \| \mathcal{T}Q$ (making the definition much more elegant), then an $\mathcal{S}$-result can also make "a step to itself" so that lemma 35 below doesn't longer hold, thus complicating the proof of correctness.

Skip this remark about the Definition. Although the *idea* of the translation (namely, the "recursion" step), and its accompanying Theorem 34 below, was readily born, it took us quite some time and several attempts before we had found the form above. In particular the treatment of communication $\|$ was eventually inspired by analysis of the reasons why previous proof attempts failed.

34 Theorem (Correctness of negative prefix elimination I) *The following diagram commutes:*

$$\begin{array}{ccc} P & \stackrel{e}{\Longrightarrow} & Q \\ {\scriptstyle\mathcal{T}}\downarrow & & \downarrow{\scriptstyle\mathcal{T}} \\ \cdot & \stackrel{e}{\longrightarrow} & \cdot \end{array}$$

where $P \stackrel{e}{\Rightarrow} Q$ means that either P makes an e-step to Q ($P \stackrel{e}{\rightarrow} Q$) or P is insensitive to e and $P = Q$. Formally, and more precisely:

$$(e \cap \beta P = \varnothing \wedge P = Q) \vee P \stackrel{e}{\rightarrow} Q \quad \Leftrightarrow \quad \mathcal{T}P \stackrel{e}{\rightarrow} \mathcal{T}Q$$

In the right-hand side the step relation $\stackrel{e}{\rightarrow}$ is the new step relation for conventional processes. In the left-hand side it is the old step relation for our saturated processes. The proof is by induction on the derivation of the step-relations.

Proof

We will use the following lemma (which itself is a nice equivalence):

35 $P \xrightarrow{e} Q \quad \Leftrightarrow \quad \mathcal{S}P \xrightarrow{e} \mathcal{T}Q$

The claims of the theorem and the lemma are shown simultaneously by induction on the structure of the derivation of the premise.

36 Note on the Proof Presentation. Formally, what we are going to prove is this:

> For each derivation Δ (whether in the old or new inference system) and for each e, P, Q:
>
> *if* $e \cap \beta P = \emptyset \wedge P = Q$ or derivation Δ ends with the conclusion $P \xrightarrow{e} Q$,
> *then* (there exists a derivation of) $\mathcal{T}P \xrightarrow{e} \mathcal{T}Q$
> (thus establishing part $\Rightarrow$ of claim 34)
>
> and: *if* derivation Δ ends with the conclusion $\mathcal{T}P \xrightarrow{e} \mathcal{T}Q$,
> *then* $e \cap \beta P = \emptyset \wedge P = Q$ or (there exists a derivation of) $P \xrightarrow{e} Q$,
> (thus establishing part $\Leftarrow$ of claim 34)
>
> and: ..... similarly for claim 35 instead of 34 .....

Since claims 34 and 35 are proved simultaneously, we may also use claim 35 as an induction hypothesis on sub-derivations in the proof of claim 34, and conversely.

We use induction on the structure of derivation Δ — with the same conventions as in the proof of Theorem 24. In particular, we will be rather implicit about the derivations, and present the proofs of an implication and its converse as a chain of equivalence steps.

We use the same convention regarding references to the rules and properties of the inference systems for the step relations as in the proof of Theorem 24.

Here is the proof of claim 35: $P \xrightarrow{e} Q \Leftrightarrow \mathcal{S}P \xrightarrow{e} \mathcal{T}Q$ (treated "from right to left" in the calculations below, namely "from the more complex form to the more simple form"). We distinguish the forms that P can take. In case P is $\frac{e_1}{e_2}{\rightarrow}P_0$, then:

$$
\begin{array}{ll}
\mathcal{S}(\frac{e_1}{e_2}{\rightarrow}P_0) \xrightarrow{e} \mathcal{T}Q & \\
\Leftrightarrow ((\sum_{e:E \mid e\cap e_2=\emptyset \wedge e_1 \subseteq e} e{\rightarrow}\mathcal{T}P_0) + \mathit{STOP}_E + \mathit{STOP}_{e1} + \mathit{STOP}_{e2}) \xrightarrow{e} \mathcal{T}Q & \text{[defn } \mathcal{S}\text{]} \\
\Leftrightarrow (\sum_{e:E \mid e\cap e_2=\emptyset \wedge e_1 \subseteq e} e{\rightarrow}\mathcal{T}P_0) \xrightarrow{e} \mathcal{T}Q & \text{[property + and } \mathit{STOP}\text{]} \\
\Leftrightarrow e \cap e_2 = \emptyset \wedge e_1 \subseteq e \ \wedge \ \mathcal{T}P_0 = \mathcal{T}Q & \text{[rule [+-iff] and [}{\rightarrow}\text{-iff]]} \\
\Leftrightarrow e \cap e_2 = \emptyset \wedge e_1 \subseteq e \ \wedge \ P_0 = Q & [\mathcal{T} \text{ is an injective function}] \\
\Leftrightarrow (\frac{e_1}{e_2}{\rightarrow}P_0) \xrightarrow{e} Q & \text{[rule [}{\rightarrow}\text{-iff]]}
\end{array}
$$

In case P is $P_1 + P_2$, then:

$$
\begin{array}{ll}
\mathcal{S}(P_1 + P_2) \xrightarrow{e} \mathcal{T}Q & \\
\Leftrightarrow \mathcal{S}P_1 + \mathcal{S}P_2 \xrightarrow{e} \mathcal{T}Q & \text{[defn } \mathcal{S}\text{]} \\
\Leftrightarrow \mathcal{S}P_1 \xrightarrow{e} \mathcal{T}Q \ \vee \ \mathcal{S}P_2 \xrightarrow{e} \mathcal{T}Q & \text{[rule [+-iff]]} \\
\Leftrightarrow P_1 \xrightarrow{e} Q \ \vee \ P_2 \xrightarrow{e} Q & \text{[induction hypothesis 35]} \\
\Leftrightarrow P_1 + P_2 \xrightarrow{e} Q & \text{[rule [+-iff]]}
\end{array}
$$

In case P is $P_1 \| P_2$, then:

$$
\begin{array}{ll}
\mathcal{S}(P_1 \| P_2) \xrightarrow{e} \mathcal{T}Q & \\
\Leftrightarrow (\mathcal{T}P_1 \| \mathcal{S}P_2 + \mathcal{S}P_1 \| \mathcal{T}P_2) \xrightarrow{e} \mathcal{T}Q & \text{[defn } \mathcal{S}\text{]} \\
\Leftrightarrow (\mathcal{T}P_1 \| \mathcal{S}P_2 \xrightarrow{e} \mathcal{T}Q) \ \vee \ (\ldots \textit{ symmetrically } \ldots) & \text{[rule [+-iff]]} \\
\Leftrightarrow \exists\, Q_1, Q_2 \mid \mathcal{T}Q_1 \| \mathcal{T}Q_2 = \mathcal{T}Q \bullet & \text{[rule [}\|\text{-iff]]}
\end{array}
$$

$$
\begin{array}{ll}
& (\mathcal{T}P_1 \xrightarrow{e} \mathcal{T}Q_1 \wedge \mathcal{S}P_2 \xrightarrow{e} \mathcal{T}Q_2) \ \vee \ (\ldots \ \textit{symmetrically} \ldots) \\
\Leftrightarrow & \exists\, Q_1, Q_2 \mid \mathcal{T}Q_1 \| \mathcal{T}Q_2 = \mathcal{T}Q \bullet & \text{[induction hypotheses 34 and 35, respectively]} \\
& ((e \cap \beta P_1 = \varnothing \wedge P_1 = Q_1 \ \vee \ P_1 \xrightarrow{e} Q_1) \wedge P_2 \xrightarrow{e} Q_2) \ \vee \ (\ldots \ \textit{symmetrically} \ldots) \\
\Leftrightarrow & \exists\, Q_1, Q_2 \mid Q_1 \| Q_2 = Q \bullet & \text{[property } \mathcal{T}\text{]} \\
& ((e \cap \beta P_1 = \varnothing \wedge P_1 = Q_1 \ \vee \ P_1 \xrightarrow{e} Q_1) \wedge P_2 \xrightarrow{e} Q_2) \ \vee \ (\ldots \ \textit{symmetrically} \ldots) \\
\Leftrightarrow & P_1 \| P_2 \xrightarrow{e} Q & \text{[rule [}\|\text{-iff]]}
\end{array}
$$

In case P is process name X, then

$$
\begin{array}{ll}
& \mathcal{S}X \xrightarrow{e} \mathcal{T}Q \\
\Leftrightarrow & X \xrightarrow{e} \mathcal{T}Q & \text{[defn } \mathcal{S}\text{]} \\
\Leftrightarrow & \mathcal{S}(\textit{body}\ X) \xrightarrow{e} \mathcal{T}Q & \text{[rule [}\widehat{=}\text{-iff], translation gives: } X \mathrel{\widehat{=}} \mathcal{S}(\textit{body}\ X)\text{]} \\
\Leftrightarrow & \textit{body}\ X \xrightarrow{e} Q & \text{[induction hypothesis 35]} \\
\Leftrightarrow & X \xrightarrow{e} Q & \text{[rule [}\widehat{=}\text{-iff]]}
\end{array}
$$

In case P is STOP_{e_0}, then:

$$
\begin{array}{ll}
& \mathcal{S}\mathit{STOP}_{e_0} \xrightarrow{e} \mathcal{T}Q \\
\Leftrightarrow & \mathit{STOP}_E + \mathit{STOP}_{e0} \xrightarrow{e} \mathcal{T}Q & \text{[defn } \mathcal{S}\text{]} \\
\Leftrightarrow & \mathit{false} & \text{[no rule for } \mathit{STOP}_E \text{ and } \mathit{STOP}_{e0}\text{]} \\
\Leftrightarrow & \mathit{STOP}_{e_0} \xrightarrow{e} \mathcal{T}Q & \text{[no rule for } \mathit{STOP}_{e_0}\text{]}
\end{array}
$$

For the proof of claim 34 we distinguish two cases according to the form that P can take. In case P is not a communication, then:

$$
\begin{array}{ll}
& \mathcal{T}P \xrightarrow{e} \mathcal{T}Q \\
\Leftrightarrow & X \xrightarrow{e} \mathcal{T}Q \quad \text{with } \mathcal{T}P = X \mathrel{\widehat{=}} (\sum_{e:E \mid e \cap \beta P = \varnothing} e{\rightarrow}X) + \mathcal{S}P & \text{[defn } \mathcal{T}\text{]} \\
\Leftrightarrow & (\sum_{e:E \mid e \cap \beta P = \varnothing} e{\rightarrow}X) \xrightarrow{e} \mathcal{T}Q \vee \mathcal{S}P \xrightarrow{e} \mathcal{T}Q \quad \text{with } \mathcal{T}P = X \mathrel{\widehat{=}} \ldots & \text{[rules [}\widehat{=}, +\text{-iff]]} \\
\Leftrightarrow & (e \cap \beta P = \varnothing \wedge X = \mathcal{T}Q) \vee (\mathcal{S}P \xrightarrow{e} \mathcal{T}Q) \quad \text{with } \mathcal{T}P = X \mathrel{\widehat{=}} \ldots & \text{[rule [}+\text{-iff]]} \\
\Leftrightarrow & (e \cap \beta P = \varnothing \wedge P = Q) \vee \mathcal{S}P \xrightarrow{e} \mathcal{T}Q & [\mathcal{T} \text{ is an injective function}] \\
\Leftrightarrow & (e \cap \beta P = \varnothing \wedge P = Q) \vee P \xrightarrow{e} Q & \text{[just shown above: claim 35 for this case]}
\end{array}
$$

We could eliminate the reference to 'claim 35 for this case' by copying here the referenced proof, so that only proper induction hypotheses of claim 34 and 35 remain.
Finally, in case $P = P_1 \| P_2$, then:

$$
\begin{array}{ll}
& \mathcal{T}P \xrightarrow{e} \mathcal{T}Q \\
\Leftrightarrow & \mathcal{T}(P_1 \| P_2) \xrightarrow{e} \mathcal{T}Q & \text{[case assumption on } P\text{]} \\
\Leftrightarrow & \mathcal{T}P_1 \| \mathcal{T}P_2 \xrightarrow{e} \mathcal{T}Q & \text{[defn } \mathcal{T}\text{]} \\
\Leftrightarrow & \mathcal{T}P_1 \| \mathcal{T}P_2 \xrightarrow{e} Q' \| Q'' \quad \text{with } Q' \| Q'' = \mathcal{T}Q & \text{[inference system]} \\
\Leftrightarrow & \mathcal{T}P_1 \| \mathcal{T}P_2 \xrightarrow{e} \mathcal{T}Q_1 \| \mathcal{T}Q_2 \quad \text{with } Q_1 \| Q_2 = Q & \text{[property } \mathcal{T}\text{]} \\
\Leftrightarrow & \mathcal{T}P_1 \xrightarrow{e} \mathcal{T}Q_1 \wedge \mathcal{T}P_2 \xrightarrow{e} \mathcal{T}Q_2 & \text{[rule [}\|\text{-iff]]} \\
\Leftrightarrow & ((e \cap \beta P_1 = \varnothing \wedge P_1 = Q_1) \vee P_1 \xrightarrow{e} Q_1) \wedge (\ldots \ 2 \ \ldots) & \text{[induction hypothesis 34]} \\
\Leftrightarrow & (e \cap \beta P_1 = \varnothing \wedge P_1 = Q_1 \wedge e \cap \beta P_2 = \varnothing \wedge P_2 = Q_2) \vee & \text{[proposition logic]} \\
& (e \cap \beta P_1 = \varnothing \wedge P_1 = Q_1 \wedge P_2 \xrightarrow{e} Q_2) \vee \\
& (P_1 \xrightarrow{e} Q_1 \wedge e \cap \beta P_2 = \varnothing \wedge P_2 = Q_2) \vee \\
& (P_1 \xrightarrow{e} Q_1 \wedge P_2 \xrightarrow{e} Q_2) \\
\Leftrightarrow & (e \cap \beta(P_1 \| P_2) = \varnothing \wedge P_1 \| P_2 = Q) \vee P_1 \| P_2 \xrightarrow{e} Q & \text{[rules [}\|_{both,left,right}\text{-iff]]}
\end{array}
$$

$\Leftrightarrow (e \cap \beta P = \varnothing \wedge P = Q) \vee P \xrightarrow{e} Q$ [case assumption on P]

End of Proof

Theorem 34 relates the inference systems on the level of "single" steps. We will now express the relation on the level of sequences of steps. First define $P \stackrel{e}{\Rightarrow} Q$ to means that 'either P ignores event set e and $P = Q$, or P participates in event set e and evolves to Q', and then extend this to sequences of event sets:

$$\frac{(e \cap \beta P{=}\varnothing \wedge P{=}Q) \vee P \xrightarrow{e} Q}{P \stackrel{e}{\Rightarrow} Q} \qquad \frac{}{P \stackrel{\langle\rangle}{\Longrightarrow} P} \qquad \frac{P \stackrel{e}{\Rightarrow} Q}{P \stackrel{\langle e \rangle}{\Longrightarrow} Q} \qquad \frac{P \stackrel{s}{\Rightarrow} Q \qquad Q \stackrel{t}{\Rightarrow} R}{P \stackrel{st}{\Longrightarrow} R}$$

37 Theorem (Correctness of negative prefix elimination II)

$$P \stackrel{s}{\Rightarrow} Q \quad \Leftrightarrow \quad \mathcal{T}P \xrightarrow{s} \mathcal{T}Q$$

The proof is by straightforward induction on the structure of s, using Theorem 34 along the way.

F Conclusion

This paper introduced a process model that is intended for giving a semantics to *life cycles*. Therefore, the process model has language concepts not usually worked out in standard process models. The most noticeable concepts were the restricted visibility of parameters, multiplicity of events, and 'negative prefixes' to block events. A translation of these concepts to standard process models was provided and proved correct.

The main correctness theorems were formulated as an equivalence ($\Leftrightarrow$). We have succeeded to present the proofs of the two implications ($\Leftarrow$ and $\Rightarrow$) that use *different* induction hypotheses, as one series of equivalences, thus being concise and precise at the same time. This presentation technique seems more generally applicable and useful.

Although the process model, and thus life cycles, can therefore be called a mere syntactical sugar coating for conventional process models, we think that the resulting language is closer to pragmatic applications such as object oriented process models, but keeping the possibility to perform formal verifications. The proof of this conjecture has to be given yet, and is planned as future work.

Acknowledgement. We thank Arend Rensink for a critical reading of a draft version of this paper.

References

[1] J.C.M. Baeten and W.P. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1990.

[2] Rational Software Corporation. *UML Semantics*. Rational Software Corporation, 1997.

[3] Maarten M. Fokkinga and Rick van Rein. A process model for life cycles. http://www.cs.utwente.nl/~fokkinga/mmf2001b.ps, 2001.

[4] OMG (Object Management Group). Uml 1.1 specification. Obtainable by http://www.omg.org/, in particular: http://www.omg.org/cgi-bin/doc?ad/99-06-08.

[5] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[6] R. Milner. The polyadic π-calculus: a tutorial. In *Proceedings of the International Summer School on Logic and Algebra of Specification*. Springer-Verlag, Aug 1991.

[7] J.M. Spivey. *The Z notation: a reference manual (2nd edition)*. Prentice Hall International, UK, 1992.

[8] Rick van Rein. http://www.cs.utwente/~vanrein.

[9] Rick van Rein. Specifying processes with dynamic life cycles. In *Proceedings of the CAiSE*00 conference*, pages 190–208, 2000.

[10] G. Winskel. *The Formal Semantics of Programming Languages: An Introduction*. Foundations of Computing series. MIT Press, Cambridge MA, 1993.