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

Abbreviated 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.

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.

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.

The unabbreviated version of this paper [3] is about 25% longer; most of the extra space goes into more proofs, while some extra space is devoted to additional semantics (sequences of steps) and informative remarks.

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	$Name$	$(X, Y : Name)$	

An environment says what variables stand for what values:

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

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 $\operatorname{dom} \rho = \{x, y, \ldots\}$ mapping x to v, y to w, and so on. Operator $\oplus$ denotes *overriding*: $(\rho \oplus \sigma)(x) = \sigma\, x$ if $x \in \operatorname{dom} \sigma$ 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*. 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)	$e ::= \varnothing \mid \{a(p)\} \mid e \cup e'$
process	(P, Q, R)	$P ::= X \mid \frac{e}{e'}{\rightarrow}P \mid P{+}P \mid P\|P \mid \mathit{STOP}_e$
process definitions	(Defs)	$\mathit{Defs} ::= X \widehat{=} P, \ldots$

In $a(p)$ the length of p must be equal to $\mathit{arity}(a)$. 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 $\frac{e}{e'}{\rightarrow}P$ ("prefix") is the basic means to express the steps that processes can make. Process $\frac{e}{e'}{\rightarrow}P$ 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 $P = \frac{e}{e'}{\rightarrow}P'$ 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.

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 $\mathit{body}\, X = P$ 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: $\sum$.

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* $\mathit{Ch}(\_)$ 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:

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

Similarly, the parameters $?x$ in the e of $STOP_e$ jointly bind each other. The *free* variable occurrences are those that are not bound. Formalised this yields the following definitions:

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

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

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

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. For the semantics of recursion, the reader may consult standard fixed point theory, as explained, for instance, by Winskel [10].

9 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 (§16). 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 $[\rightarrow]$ (§12) 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

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

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.

11 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: §14.)

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

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

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

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':

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

For example, consider:

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

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:

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

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 §13.

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

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):

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

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

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

13 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 $[\|_{\mathit{both}}]$ we have assumed that $\mathit{Fr}\; P \cup \mathit{Bnd}\; P$ is disjoint from $\mathit{Bnd}\; Q$, and conversely, $\mathit{Bnd}\; P$ is disjoint from $\mathit{Fr}\; Q \cup \mathit{Bnd}\; Q$. Here $\mathit{Bnd}\; P$ 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 $\mathit{Fr}\; P$ is disjoint from $\operatorname{dom}\tau$, and $\operatorname{dom}\sigma$ from $\mathit{Fr}\; Q$, 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 (§24, case $P_1\|P_2$).

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

14 Sequences of steps. The full paper also deals with (definitions of and theorems about) sequences of steps in succession; for brevity we don't present them here, except for the rule to compose sequences of steps.

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

15 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:

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

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.

16 Saturation of processes. The saturation for events has been given earlier, in §9. Here is the saturation of processes (for ρ satisfying $Fr\ P \subseteq \mathrm{dom}\,\rho$):

P	$[\![P]\!]_\rho$
X	X_ρ
$\frac{e}{e'}\!\to\! P$	$\sum_{\sigma \mid \mathrm{dom}\,\sigma = Ch\ e} \frac{[\![e]\!]_\rho^\sigma}{[\![e']\!]_{\rho\oplus\sigma}}\!\to\![\![P]\!]_{\rho\oplus\sigma}$
$P \otimes Q$	$[\![P]\!]_\rho \otimes [\![Q]\!]_\rho$

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.

17 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).

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

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

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

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

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

The left-hand β_ρ is the original one from §10; the right-hand β is the new, simpler one from §18. In other words, the following diagram commutes:

$$\begin{tikzcd} P \arrow[r, "\beta_\rho"] \arrow[d, "{[\![\_]\!]_\rho}"'] & \cdot \arrow[d, "id"] \\ \cdot \arrow[r, "\beta"] & \cdot \end{tikzcd}$$

□

The proof is spelled out in the unabbreviated version of this paper; it is by straightforward induction, although some care is needed to deal properly with the recursive structure in the definition of β: “Scott’s fixed point induction”.

20 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:

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

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':

$$[\rightarrow]\,\frac{}{\frac{e}{e'}\!\rightarrow P \ \xrightarrow{e''}\ P}\ \Big\{\ e'' \cap e' = \varnothing \wedge e \subseteq e''$$

A choice can do a step if either operand can:

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

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

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

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

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.

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:

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

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

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

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

Proof

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

22 $\quad [\![P]\!]_{\rho_1} \quad = \quad [\![P]\!]_{\rho_2} \quad \text{if } \forall\, x : \mathit{Fr}\,P \bullet \rho_1 x = \rho_2 x$

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

Now we prove claim 21 by induction on the structure of the derivations, using Theorem 19 along the way.

23 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 21)
>
> 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 21)

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:

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

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

These implications are referred to by 'rule $[+_{\mathit{left}}]$' and 'rule $[+_{\mathit{right}}]$', 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.

24 Proof of claim 21. We distinguish the form that process expression P can take. In case P is process name X, then:

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

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

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

In case P is $P_1 \| P_2$ and the last step of the derivation in the premise is rule $[\|_{\mathit{left}}]$, then:

$$\begin{array}{ll}
[\![P_1 \| P_2]\!]_\rho \xrightarrow{e} Q & \\
\Leftrightarrow [\![P_1]\!]_\rho \| [\![P_2]\!]_\rho \xrightarrow{e} Q & [\text{defn } [\![_]\!]_\rho] \\
\Leftrightarrow \exists\, Q_1 \bullet [\![P_1]\!]_\rho \xrightarrow{e} Q_1 \wedge e \cap \beta [\![P_2]\!]_\rho = \varnothing \wedge Q_1 \| [\![P_2]\!]_\rho = Q & [\text{(last step), rule } [\|_{\mathit{left}}\text{-iff}]] \\
\cdots\cdots\cdots & \cdots\cdots\cdots \\
\Leftrightarrow \exists\, Q_1 \bullet (\exists\, \sigma, P_1' \bullet P_1 \xrightarrow{e}_{\rho,\sigma} P_1' \xmapsto{[\![_]\!]_{\rho\oplus\sigma}} Q_1) \wedge e \cap \beta_\rho P_2 = \varnothing \wedge Q_1 \| [\![P_2]\!]_\rho = Q & \\
& [\text{induction hypothesis, Theorem 19}] \\
\Leftrightarrow \exists\, Q_1, \sigma, P_1' \bullet P_1 \xrightarrow{e}_{\rho,\sigma} P_1' \xmapsto{[\![_]\!]_{\rho\oplus\sigma}} Q_1 \wedge e \cap \beta_\rho P_2 = \varnothing \wedge Q_1 \| [\![P_2]\!]_\rho = Q & [\text{pred. logic}] \\
\Leftrightarrow \exists\, Q_1, \sigma, P_1' \bullet P_1 \xrightarrow{e}_{\rho,\sigma} P_1' \xmapsto{[\![_]\!]_{\rho\oplus\sigma}} Q_1 \wedge e \cap \beta_\rho P_2 = \varnothing \wedge Q_1 \| [\![P_2]\!]_{\rho\oplus\sigma} = Q &
\end{array}$$

[naming convention, Lemma 22]

$\Leftrightarrow \exists\, \sigma, P_1' \bullet P_1 \xrightarrow{e}_{\rho,\sigma} P_1' \wedge e \cap \beta_\rho P_2 = \varnothing \wedge P_1' \| P_2 \xmapsto{[\![\_]\!]_{\rho\oplus\sigma}} Q$ [one-point rule, defn $[\![\_]\!]\_$]

$\Leftrightarrow \exists\, \sigma, P' \bullet P_1 \| P_2 \xrightarrow{e}_{\rho,\sigma} P' \xmapsto{[\![\_]\!]_{\rho\oplus\sigma}} Q$ [(last step), rule $[\|_{left}$-iff]]

Similarly when the last step of the derivation in the premise is rule $[\|_{right}]$ or $[\|_{both}]$.
The cases for $P_1 + P_2$ and $STOP_e$ are simple, and can be found in the unabbreviated paper.

End of Proof

E Eliminating negative prefixes

Now we're going to translate our processes into conventional ones: conventional processes have only a positive prefix (so no explicit means for blocking), and just a single action in the prefix.

25 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 '$e{\to}P$' parts, the use of conventional tools will only succeed for *toy* processes.

26 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)	$P ::= X \mid e{\to}P \mid P{+}P \mid P\|P \mid STOP_{E'}$
process definitions	$(Defs)$	$Defs ::= X \widehat{=} P, \ldots$

The alphabet αP of a conventional process is defined thus:

P	αP
$e{\to}P$	$\{e\} \cup \alpha P$
$P \otimes Q$	$\alpha P \cup \alpha Q$
$STOP_{E'}$	E'
X	$\alpha(body\ X)$

27 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:

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

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

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

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

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

A communication can do a step 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 $STOP_{E'}$ cannot do a step at all; so, there is no inference rule for it.

28 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 *actions*. 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 $\frac{e_1}{e_2}\rightarrow\ldots$ 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:

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

and

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

In the $\mathcal{T}$-translation of a system, the process definitions $X \mathrel{\widehat{=}} \mathit{body}\,X$ have to be transformed into $X \mathrel{\widehat{=}} \mathcal{S}(\mathit{body}\,X)$. 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 30 below doesn't longer hold, thus complicating the proof of correctness.

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

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

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

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

In the right-hand side the step relation $\xrightarrow{e}$ 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):

30 $\quad P \xrightarrow{e} Q \quad \Leftrightarrow \quad \mathcal{S}P \xrightarrow{e} \mathcal{T}Q$

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

31 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* $e \cap \beta P = \varnothing \wedge P = Q$ 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 29)
> and: *if* derivation Δ ends with the conclusion $\mathcal{T}P \xrightarrow{e} \mathcal{T}Q$,
> *then* $e \cap \beta P = \varnothing \wedge P = Q$ or (there exists a derivation of) $P \xrightarrow{e} Q$,
> (thus establishing part $\Leftarrow$ of claim 29)
> and: ...... similarly for claim 30 instead of 29 ......

Since claims 29 and 30 are proved simultaneously, we may also use claim 30 as an induction hypothesis on sub-derivations in the proof of claim 29, and conversely.

We use induction on the structure of derivation Δ — with the same conventions as in the proof of Theorem 21. 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 21.

Here is the proof of claim 30: $P \xrightarrow{e} Q \Leftrightarrow \mathcal{S}P \xrightarrow{e} \mathcal{T}Q$ (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:

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

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

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

$$(\mathcal{T}P_1 \xrightarrow{e} \mathcal{T}Q_1 \wedge \mathcal{S}P_2 \xrightarrow{e} \mathcal{T}Q_2) \ \vee \ (\ldots \ \textit{symmetrically} \ldots)$$

$$\Leftrightarrow \exists\, Q_1, Q_2 \mid \mathcal{T}Q_1 \| \mathcal{T}Q_2 = \mathcal{T}Q \bullet \qquad \text{[induction hypotheses 29 and 30, respectively]}$$

$$((e \cap \beta P_1 = \varnothing \wedge P_1 = Q_1 \ \vee \ P_1 \xrightarrow{e} Q_1) \wedge P_2 \xrightarrow{e} Q_2) \ \vee \ (\ldots \ \textit{symmetrically} \ldots)$$

$$\Leftrightarrow \exists\, Q_1, Q_2 \mid Q_1 \| Q_2 = Q \bullet \qquad \text{[property } \mathcal{T}\text{]}$$

$$((e \cap \beta P_1 = \varnothing \wedge P_1 = Q_1 \ \vee \ P_1 \xrightarrow{e} Q_1) \wedge P_2 \xrightarrow{e} Q_2) \ \vee \ (\ldots \ \textit{symmetrically} \ldots)$$

$$\Leftrightarrow P_1 \| P_2 \xrightarrow{e} Q \qquad \text{[rule [}\|\text{-iff]]}$$

The cases for $P_1 + P_2$, STOP_{e0} and X are simple, and occur in the unabbreviated version of this paper.

For the proof of claim 29 we distinguish two cases according to the form that P can take. In case P is not a communication, then:

$$\mathcal{T}P \xrightarrow{e} \mathcal{T}Q$$

$$\Leftrightarrow X \xrightarrow{e} \mathcal{T}Q \quad \text{with } \mathcal{T}P = X \mathrel{\widehat{=}} (\textstyle\sum_{e:E \mid e \cap \beta P = \varnothing} e \to X) + \mathcal{S}P \qquad \text{[defn } \mathcal{T}\text{]}$$

$$\Leftrightarrow (\textstyle\sum_{e:E \mid e \cap \beta P = \varnothing} e \to X) \xrightarrow{e} \mathcal{T}Q \vee \mathcal{S}P \xrightarrow{e} \mathcal{T}Q \quad \text{with } \mathcal{T}P = X \mathrel{\widehat{=}} \ldots \qquad \text{[rules [}\widehat{=}, +\text{-iff]]}$$

$$\Leftrightarrow (e \cap \beta P = \varnothing \wedge X = \mathcal{T}Q) \vee (\mathcal{S}P \xrightarrow{e} \mathcal{T}Q) \quad \text{with } \mathcal{T}P = X \mathrel{\widehat{=}} \ldots \qquad \text{[rule [+-iff]]}$$

$$\Leftrightarrow (e \cap \beta P = \varnothing \wedge P = Q) \vee \mathcal{S}P \xrightarrow{e} \mathcal{T}Q \qquad \text{[}\mathcal{T} \text{ is an injective function]}$$

$$\Leftrightarrow (e \cap \beta P = \varnothing \wedge P = Q) \vee P \xrightarrow{e} Q \qquad \text{[just shown above: claim 30 for this case]}$$

We could eliminate the reference to 'claim 30 for this case' by copying here the referenced proof, so that only proper induction hypotheses of claim 29 and 30 remain.
Finally, in case $P = P_1 \| P_2$, then:

$$\mathcal{T}P \xrightarrow{e} \mathcal{T}Q$$

$$\Leftrightarrow \mathcal{T}(P_1 \| P_2) \xrightarrow{e} \mathcal{T}Q \qquad \text{[case assumption on } P\text{]}$$

$$\Leftrightarrow \mathcal{T}P_1 \| \mathcal{T}P_2 \xrightarrow{e} \mathcal{T}Q \qquad \text{[defn } \mathcal{T}\text{]}$$

$$\Leftrightarrow \mathcal{T}P_1 \| \mathcal{T}P_2 \xrightarrow{e} Q' \| Q'' \quad \text{with } Q' \| Q'' = \mathcal{T}Q \qquad \text{[inference system]}$$

$$\Leftrightarrow \mathcal{T}P_1 \| \mathcal{T}P_2 \xrightarrow{e} \mathcal{T}Q_1 \| \mathcal{T}Q_2 \quad \text{with } Q_1 \| Q_2 = Q \qquad \text{[property } \mathcal{T}\text{]}$$

$$\Leftrightarrow \mathcal{T}P_1 \xrightarrow{e} \mathcal{T}Q_1 \wedge \mathcal{T}P_2 \xrightarrow{e} \mathcal{T}Q_2 \qquad \text{[rule [}\|\text{-iff]]}$$

$$\Leftrightarrow ((e \cap \beta P_1 = \varnothing \wedge P_1 = Q_1) \vee P_1 \xrightarrow{e} Q_1) \wedge (\ldots \ 2 \ \ldots) \qquad \text{[induction hypothesis 29]}$$

$$\Leftrightarrow (e \cap \beta P_1 = \varnothing \wedge P_1 = Q_1 \wedge e \cap \beta P_2 = \varnothing \wedge P_2 = Q_2) \vee \qquad \text{[proposition logic]}$$

$$(e \cap \beta P_1 = \varnothing \wedge P_1 = Q_1 \wedge P_2 \xrightarrow{e} Q_2) \vee$$

$$(P_1 \xrightarrow{e} Q_1 \wedge e \cap \beta P_2 = \varnothing \wedge P_2 = Q_2) \vee$$

$$(P_1 \xrightarrow{e} Q_1 \wedge P_2 \xrightarrow{e} Q_2)$$

$$\Leftrightarrow (e \cap \beta(P_1 \| P_2) = \varnothing \wedge P_1 \| P_2 = Q) \vee P_1 \| P_2 \xrightarrow{e} Q \qquad \text{[rules [}\|_{both,left,right}\text{-iff]]}$$

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

End of Proof

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.