A process-algebraic approach to life-cycle inheritance : inheritance = encapsulation + abstraction

Eindhoven University of Technology
Department of Mathematics and Computing Science

A Process-Algebraic Approach to Life-Cycle Inheritance
Inheritance = Encapsulation + Abstraction

by

T. Basten and W.M.P. van der Aalst

96/05

ISSN 0926-4515

editors: prof.dr. R.C. Backhouse
prof.dr. J.C.M. Baeten

Reports are available at:
http://www.win.tue.nl/win/cs

Computing Science Report 96/05
Eindhoven, March 1996

A Process-Algebraic Approach to Life-Cycle Inheritance

Inheritance = Encapsulation + Abstraction

T. Basten and W.M.P. van der Aalst

Department of Mathematics and Computing Science
Eindhoven University of Technology, The Netherlands
email: {tbasten,wsinwa}@win.tue.nl

Abstract. One of the key issues of object-oriented modeling is *inheritance*. It allows for the definition of subclasses that inherit features of some superclass. Inheritance is well defined for static properties of classes such as attributes and methods. However, there is no general agreement on the meaning of inheritance when considering the dynamic behavior of objects, determined by their life cycles. This paper studies the latter in the context of a simple process algebra. Process algebra is chosen, because it concentrates on dynamic behavior, while abstracting from the internal states of processes. Inheritance can be expressed in terms of *encapsulation* and *abstraction*. The combination captures all basic operators for constructing life cycles of subclasses from life cycles of superclasses, namely choice, sequential composition, and parallel composition.

Keywords: object orientation – inheritance – object life cycle – process algebra – dynamic behavior

1 Introduction

In software-engineering practice, the popularity of object-oriented modeling and design is increasing rapidly. Two methodologies are in common use: OMT [9] and OOD [6]. One of the key issues in any object-oriented methodology is *inheritance*. The inheritance mechanism allows the user to specify a subclass that inherits features of some other class, its superclass. Thus, it is possible to specify that the subclass has the same features as the superclass, but that in addition it may have some other features. The subclass is a specialization of the superclass. It is not necessary to argue that the inheritance mechanism has many advantages for software development. The interested reader is referred to the literature on object orientation.

The concept of inheritance is usually well defined for *static* features of an object class, i.e., the set of methods of an object and its attributes. However, an object also has dynamic features, namely its behavior determined by its *life cycle*. The life cycle of an object describes the order in which its methods can be called. Methodologies as OMT and OOD do not specify the meaning of inheritance of life cycles. In general, there seems to be no agreement on what exactly inheritance of dynamic behavior is (See for example [2]). That is, it is not clear when some object class inherits the life cycle of some other class.

Techniques as OMT and OOD use state-transition diagrams for specifying the life cycles of objects belonging to some class. Such a diagram shows the state space of an object and the method calls that cause a transition from one state to another. Although the graphical nature and the explicit representation of states are essential to the success and usefulness of OMT and OOD, particularly the latter impedes a clear understanding of life-cycle inheritance. For studying inheritance of dynamic behavior, the most important aspects of a life cycle are the *state changes* and not the states themselves.

Therefore, this paper studies the specification of object life cycles and the meaning of life-cycle inheritance in process algebra. Process algebra is particularly well suited to describe process behavior without explicitly referring to process states. In addition, we believe that life-cycle inheritance corresponds to abstraction and encapsulation of methods. These two concepts are well investigated in the field of process algebra, in particular the process algebra ACP [5]. We hope to get a clear understanding of the meaning of life-cycle inheritance in terms of a simple algebraic theory. In another paper [1], we turn to Petri nets which is a graphical formalism with a solid theoretical basis, but much closer to existing object-oriented techniques such as OMT and OOD. In that paper, the concepts developed in process algebra are translated to Petri nets.

This translation is illustrative for translations of the fundamentals developed in this paper to other graphical, state-based formalisms.

There seem to be many possible answers to the question when some object inherits the life cycle of some other object. It is important to note that we have to ask this question from the viewpoint of the *environment* of an object consisting of other objects and possibly the object itself. Objects from the environment call methods from the object under consideration, thus observing its behavior. So the appropriate question is when does the environment agree that an object inherits the life cycle of another object. We give the following two answers to this question and we hope to convince the reader that they contain the essentials of life-cycle inheritance.

The first answer is that an object inherits the life-cycle of another object if and only if the environment cannot distinguish the two objects while only calling methods of the latter. The second answer is that an object inherits the life-cycle of another object if and only if the environment cannot distinguish the two objects as long as it is willing to call methods of the first object not present in the second one appropriately.

In this paper, we show that the first answer corresponds to *blocking* or *encapsulating* methods, whereas the second one corresponds to *hiding* methods by means of *abstraction*. We also show that the combination of encapsulation and abstraction is sufficient to capture three basic operators that are useful to describe object life cycles, namely choice, sequential composition, and parallel composition. That is, if the behavior of an object is the behavior of another object, but with the option to choose at some point an alternative behavior, then the first object is a subclass of the second one. Similarly, if the behavior of the first object is the behavior of the second one with some other behavior added somewhere in between or in parallel, then the first object is also a subclass of the other one.

The remainder of this paper is organized as follows. The two section introduces a simple process algebra, sufficient for describing object life cycles. Section 3 formalizes the above informal answers to the question which we try to answer in this paper. Four inheritance relations are given and several basic properties of these relations are proven. In Section 4, some general rules are given showing what subclass relations are valid under any of the four forms of inheritance. These rules show that encapsulation and abstraction indeed capture the three basic operators, choice, sequential composition, and parallel composition. Finally, Section 5 ends with some concluding remarks.

2 Object Life Cycles in Process Algebra

This section presents a process algebra which is sufficient to define simple object life cycles and which contains encapsulation and abstraction operators needed to define inheritance relations. In process algebra, processes are constructed from a set of atomic processes or atomic actions by means of operators. In this paper, processes represent object life cycles and atomic actions conform to methods; the operators used to construct life cycles are the standard algebraic operators choice, denoted $+$, sequential composition, denoted $\cdot$, and parallel composition or merge, denoted $\|$. Since our only goal is to get a good understanding of life-cycle inheritance, we restrict ourselves to these three operators. In particular, we do not formalize recursion. Recursion would unnecessarily complicate the theory and distract the readers attention from the important concepts. For a detailed treatment of recursion in ACP, the interested reader is referred to [5, 4].

For modeling object life cycles, assume that we have a set L of labels denoting methods. Label τ is used to denote *internal methods*. Since we are not interested in the effect of methods, but only in the order in which methods can be called, we do not distinguish between different internal methods. A life cycle is simply an algebraic term constructed from the labels in L and τ with only one restriction. A life cycle always starts with an *object-creation* action, denoted by the special label $\triangledown$. The introduction of this restriction is realistic for any object-oriented methodology and it proves to be convenient in the remainder of this paper. Let L_s denote the set of labels including the special labels τ and $\triangledown$.

Before formally defining an object life cycle, Table 1 gives the theory $\mathrm{PA}^{\tau}_{\delta\rho}$, Process Algebra with deadlock, internal actions, and renaming. The first entry of Table 1 gives the sorts of the theory; P is the sort of all

processes containing all methods. The second entry lists the functions of the theory; the third entry contains the axioms. Besides the three operators already mentioned, the theory contains a constant δ, denoting *deadlock* or *inaction*, an auxiliary operator $\lfloor\!\lfloor$ called the left merge which is used to axiomatize the merge operator, an encapsulation operator $\partial_{\_}$ which is parameterized by the set of labels to be encapsulated, and, finally, an abstraction operator $\tau_{\_}$ which is parameterized by the set of methods that must be hidden.

$\_\mathrm{PA}^{\tau}_{\delta\rho}(L_s)\_$

$P;\ L_s \subseteq P$

$\delta : P \qquad \_+\_, \_\cdot\_, \_\parallel\_, \_\lfloor\!\lfloor\_ : P \times P \to P \qquad \partial_{\_}, \tau_{\_} : \mathcal{P}(L) \to (P \to P)$

$a : L_s \cup \{\delta\};\ H, I : \mathcal{P}(L);\ x, y, z : P;$

$x + y = y + x$	$A1$	$x \parallel y = x \lfloor\!\lfloor y + y \lfloor\!\lfloor x$	$M1$
$(x + y) + z = x + (y + z)$	$A2$	$a \lfloor\!\lfloor x = a \cdot x$	$M2$
$x + x = x$	$A3$	$a \cdot x \lfloor\!\lfloor y = a \cdot (x \parallel y)$	$M3$
$(x + y) \cdot z = x \cdot z + y \cdot z$	$A4$	$(x + y) \lfloor\!\lfloor z = x \lfloor\!\lfloor z + y \lfloor\!\lfloor z$	$M4$
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	$A5$		
$x + \delta = x$	$A6$	$x \cdot \tau = x$	$B1$
$\delta \cdot x = \delta$	$A7$	$x \cdot (\tau \cdot (y + z) + y) = x \cdot (y + z)$	$B2$
$a \notin H \Rightarrow \partial_H(a) = a$	$D1$	$a \notin I \Rightarrow \tau_I(a) = a$	$T1$
$a \in H \Rightarrow \partial_H(a) = \delta$	$D2$	$a \in I \Rightarrow \tau_I(a) = \tau$	$T2$
$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$	$D3$	$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$	$T3$
$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$	$D4$	$\tau_I(x \cdot y) = \tau_I(x) \cdot \tau_I(y)$	$T4$

Table 1: The process algebra $\mathrm{PA}^{\tau}_{\delta\rho}$.

Intuitively, the left merge means parallel execution, but with the restriction that the left process executes the first action. The encapsulation and abstraction operators are simply renaming operators that rename method labels to δ and τ respectively. Note that it follows from the requirement that H and I are subsets of L that the special labels τ and $\triangledown$ cannot be encapsulated or hidden. Axioms $A6$ and $A7$ show the behavior of δ in a choice or sequential-composition context. It is easy to see that δ does indeed model inaction. Axiom $A4$ states the right distributivity of sequential composition over choice. The absence of the left-distributivity axiom implies that processes with different moments of choice are distinguished. The $B1$ and $B2$ axioms are the so-called branching axioms, which state that internal actions not implying a choice can be removed.

As explained below, $\mathrm{PA}^{\tau}_{\delta\rho}$ is an axiomatization of an equivalence in which processes with the same observable behavior, but with possibly different internal behavior are equal. Furthermore, the equivalence relation distinguishes processes with different moments of choice. The timing of choices is important, since it can influence the behavior observed by an environment.

An important role in this paper plays the theory PA^{τ} which is a subtheory of $\mathrm{PA}^{\tau}_{\delta\rho}$. The theory PA^{τ} contains the action symbols in L_s plus the sequential-composition, choice, merge, and left-merge operators. It has the axioms $A1$ through $A5$, $M1$ through $M4$, and $B1$, and $B2$. It is the theory which exactly contains all operators that we want to use to specify object life cycles. Hence, at this point, object life cycles can be formally defined as follows. For any theory T, let $C(T)$ denote the set of *closed* T terms. The operator α, which determines the alphabet of visible method labels of an arbitrary closed term, is defined below.

Definition 2.1. (Object life cycle) A closed PA^{τ} term $p \in C(\mathrm{PA}^{\tau})$ denotes an object life cycle if and only if it is of the form $\triangledown \cdot q$, for some closed PA^{τ} term q such that $\alpha(q) \subseteq L$. That is, p must be equal to a term $\triangledown \cdot q$, where q does not contain any other creation action.

Example 2.2. The life cycle of a person can be described by the algebraic term: *birth* · (*marriage* · *death*), where "*birth*" is the special object-creation action ∇.

Definition 2.3. (Alphabet) The alphabet operator on closed $\mathrm{PA}^{\tau}_{\delta\rho}$ terms is defined inductively as follows. For any $a \in L_s \setminus \{\tau\}$ and closed terms $p, q \in \mathcal{C}(\mathrm{PA}^{\tau}_{\delta\rho})$, $\alpha(\delta) = \emptyset$, $\alpha(\tau) = \emptyset$, $\alpha(a) = \{a\}$, $\alpha(\tau \cdot p) = \alpha(p)$, $\alpha(a \cdot p) = \{a\} \cup \alpha(p)$, and $\alpha(p + q) = \alpha(p) \cup \alpha(q)$.

The above definition of the alphabet operator is taken from [3]. In that paper, it is shown that the alphabet operator is a congruence for the operators of $\mathrm{PA}^{\tau}_{\delta\rho}$. That is, if two $\mathrm{PA}^{\tau}_{\delta\rho}$ terms are derivably equal, then they have the same alphabet. Derivability of an equation e from a theory T is denoted $T \vdash e$.

Property 2.4. *For any* $p, q \in \mathcal{C}(\mathrm{PA}^{\tau}_{\delta\rho})$, $\mathrm{PA}^{\tau}_{\delta\rho} \vdash p = q \Rightarrow \alpha(p) = \alpha(q)$.

Note that the alphabet operator is only defined for the choice and the sequential composition of closed terms. However, it is a well known result that the merge, left-merge, encapsulation, and abstraction operators can be eliminated from any closed term yielding a so-called *basic* term. Many of the proofs to follow also use this elimination property. Hence, it is formalized in Property 2.6 given below.

In order to define basic terms, two subtheories of $\mathrm{PA}^{\tau}_{\delta\rho}$ are introduced. The theory BPA, for Basic Process Algebra, is the equational theory whose signature consists of the action symbols in L_s plus sequential composition and choice. It has the axioms $A1$ through $A5$. The theory BPA_{δ} contains in addition the constant δ and the axioms $A6$ and $A7$. Note that PA^{τ} is an extension of BPA, but not of BPA_{δ}.

Definition 2.5. (Basic terms) The set of basic BPA terms, denoted $\mathcal{B}(\mathrm{BPA})$, is inductively defined as follows. The atomic actions L_s are contained in $\mathcal{B}(\mathrm{BPA})$. Furthermore, for any $a \in L_s$ and $s, t \in \mathcal{B}(\mathrm{BPA})$, also $a \cdot t$ and $s + t$ are elements of $\mathcal{B}(\mathrm{BPA})$. The set of basic BPA_{δ} terms, $\mathcal{B}(\mathrm{BPA}_{\delta})$, is defined in a similar way: $L_s \cup \{\delta\} \subseteq \mathcal{B}(\mathrm{BPA}_{\delta})$ and for any $a \in L_s$ and $s, t \in \mathcal{B}(\mathrm{BPA}_{\delta})$, $a \cdot t \in \mathcal{B}(\mathrm{BPA}_{\delta})$ and $s + t \in \mathcal{B}(\mathrm{BPA}_{\delta})$.

Property 2.6. (Elimination) *For any closed term* $p \in \mathcal{C}(\mathrm{PA}^{\tau})$, *there exists a basic term* $t \in \mathcal{B}(\mathrm{BPA})$, *such that* $\mathrm{PA}^{\tau} \vdash p = t$. *For any closed term* $p \in \mathcal{C}(\mathrm{PA}^{\tau}_{\delta\rho})$, *there exists a basic term* $t \in \mathcal{B}(\mathrm{BPA}_{\delta})$, *such that* $\mathrm{PA}^{\tau}_{\delta\rho} \vdash p = t$.

Proof. It is straightforward to prove the property using the axioms of Table 1 excluding $A1$ and $A2$ as rewrite rules from left to right. For a description of some standard term-rewriting techniques used in process algebra, see [4]. □

It is straightforward to give an operational semantics for $\mathrm{PA}^{\tau}_{\delta\rho}$ (See for example [5]). It is a standard result in process algebra that the equational theory $\mathrm{PA}^{\tau}_{\delta\rho}$ is a sound and complete axiomatization of an equivalence called *rooted branching bisimulation*. Branching bisimulation and rooted branching bisimulation were originally introduced by Van Glabbeek and Weijland in [8]. The latter is a restriction of the former by means of an additional root condition. The root condition is necessary to guarantee that the equivalence is a congruence for the algebraic choice operator. For object life cycles, which always start with the object-creation action, rooted branching bisimulation and branching bisimulation coincide. In [7], Van Glabbeek shows that branching bisimulation is exactly the equivalence that distinguishes processes with different moments of choice and in which processes with possibly different internal behavior but with the same observable behavior are equal.

It goes beyond the scope of this paper to present a complete definition of the operational semantics of $\mathrm{PA}^{\tau}_{\delta\rho}$ and a detailed proof of the soundness and completeness result. Rooted branching bisimulation is not used explicitly in this paper. Techniques for proving soundness and completeness of equational theories and many basic results can be found in [4] and [5]. Although the soundness and completeness of $\mathrm{PA}^{\tau}_{\delta\rho}$ is not proven in either of these references, it is a fairly straightforward consequence of some results in [5] and the proof techniques of [4].

Theorem 2.7. (Soundness and completeness) *For any closed terms* $p, q \in \mathcal{C}(\mathrm{PA}^{\tau}_{\delta\rho})$, $\mathrm{PA}^{\tau}_{\delta\rho} \vdash p = q$ *if and only if* p *and* q *are rooted branching bisimilar.*

3 Inheritance Relations in Process Algebra

In this section, we try to answer the question when an object life cycle is a subclass of some other object life cycle. Four inheritance relations are introduced and some basic properties of these relations are proven. All four inheritance relations are reflexive and transitive. Furthermore, it is shown that two life cycles are each others subclasses if and only if they are derivably equal. These properties show that the definitions of the inheritance relations are sound.

As explained in the introduction, informally, two basic definitions of life-cycle inheritance seem to be appropriate. Let p and q be object life cycles. The first definition is as follows.

> If the environment only calls methods of p which are also present in q and it cannot distinguish the observable behavior of p and q, then p is a subclass of q.

Intuitively, this definition conforms to *blocking* or *encapsulating* methods new in p. Life cycle p inherits the *protocol* of q, where the protocol of an object life cycle is its prescribed behavior.

Example 3.1. Let us return to Example 2.2. A person that can decide whether to marry or to stay single should be a subclass of the person described in Example 2.2. That is, $birth \cdot ((marriage + stay\_single) \cdot death)$ should be a subclass of $birth \cdot (marriage \cdot death)$. It is easy to see that an environment which does not call the method *stay_single* cannot distinguish the behavior of the two persons, which is indeed the desired result. To show that the informal definition conforms to encapsulation of methods, let H be the singleton $\{stay\_single\}$. It follows immediately from the encapsulation axioms in Table 1 that $\partial_H(birth \cdot ((marriage + stay\_single) \cdot death)) = (birth \cdot ((marriage + \delta) \cdot death)) = birth \cdot (marriage \cdot death)$.

The second definition that seems to be appropriate is as follows.

> If the environment is willing to call the methods of p which are not present in q and it cannot distinguish the observable behavior of p and q with respect to the methods of q, then p is a subclass of q.

This definition conforms to *hiding* methods new in p. Life cycle p inherits the *projection* of the protocol of p onto the methods of q.

Example 3.2. Consider again Example 2.2. A person that decides to divorce after he or she got married should also be a subclass of the person described in Example 2.2. That is, $birth \cdot ((marriage \cdot divorce) \cdot death)$ should be a subclass of $birth \cdot (marriage \cdot death)$. Again, it is not difficult to see that an environment which is willing to call the method *divorce* appropriately cannot distinguish the behavior of the two persons. In order to show that this form of inheritance conforms to hiding, let I be equal to the singleton $\{divorce\}$. It follows from the abstraction axioms in Table 1 that $\tau_I(birth \cdot ((marriage \cdot divorce) \cdot death)) = birth \cdot ((marriage \cdot \tau) \cdot death) = birth \cdot (marriage \cdot death)$.

Of course, it is also possible to combine these two definitions. One might argue that both requirements must hold at the same time, or one might argue that for some methods the first requirement must hold, whereas for some other methods the other requirement holds. This leads to the following four possible inheritance relations. Note that the formal definitions are slightly more general than the informal definitions given above: A life cycle is a subclass of another life cycle if and only if there exists *some* set of methods such that encapsulating or hiding these methods in the first life cycle yields the other life cycle. Not requiring that the methods being encapsulated or hidden must be *exactly* the methods appearing in the first life cycle and not in the second one can sometimes be convenient, as some of the examples in the remainder show. Also for convenience, the inheritance relations are defined for arbitrary closed PA^τ terms and not only for life cycles.

Definition 3.3. (Inheritance relations)

i) *Protocol inheritance*:
For any $p, q \in \mathcal{C}(\mathrm{PA}^\tau)$, p is said to be a subclass of q under *protocol inheritance*, denoted $p \leq_{pt} q$, if and only if there exists an $H \subseteq L$ such that $\mathrm{PA}^\tau_{\delta\rho} \vdash \partial_H(p) = q$.

ii) *Projection inheritance*:
For any $p, q \in \mathcal{C}(\mathrm{PA}^\tau)$, p is a subclass of q under *projection inheritance*, denoted $p \leq_{pj} q$, if and only if there exists an $I \subseteq L$ such that $\mathrm{PA}^\tau_{\delta\rho} \vdash \tau_I(p) = q$.

iii) *Protocol/projection inheritance*:
For any $p, q \in \mathcal{C}(\mathrm{PA}^\tau)$, p is a subclass of q under *protocol/projection inheritance*, denoted $p \leq_{pp} q$, if and only if there exists an $H \subseteq L$ such that $\mathrm{PA}^\tau_{\delta\rho} \vdash \partial_H(p) = q$ *and* there exists an $I \subseteq L$ such that $\mathrm{PA}^\tau_{\delta\rho} \vdash \tau_I(p) = q$.

iv) *Life-cycle inheritance*:
For any $p, q \in \mathcal{C}(\mathrm{PA}^\tau)$, p is a subclass of q under *life-cycle inheritance*, denoted $p \leq_{lc} q$, if and only if there exist *disjoint* subsets H and I of L such that $\mathrm{PA}^\tau_{\delta\rho} \vdash \tau_I \circ \partial_H(p) = q$.

Note that the above definitions are formulated in terms of equality of closed $\mathrm{PA}^\tau_{\delta\rho}$ terms. The completeness of $\mathrm{PA}^\tau_{\delta\rho}$ for rooted branching bisimulation (Theorem 2.7) implies that this formulation is equivalent to a formulation in terms of rooted branching bisimulation. Without completeness, the above definitions would be too restrictive. It would be possible that an object life cycle p after encapsulation and/or abstraction of the appropriate methods would be rooted branching bisimilar with q, but that this equality would not be derivable from the axioms of $\mathrm{PA}^\tau_{\delta\rho}$. This would be undesirable, because, according to the above definitions, p would not be a subclass of q.

The requirement that H and I must be disjoint in the definition of life-cycle inheritance means that methods are either consistently encapsulated or consistently hidden. It implies that the order of encapsulation and abstraction can be changed without actually changing the definition (See also Lemma 3.9 *iii*)). To our opinion it is not very meaningful to treat different calls of one method in a different way. It is not possible to hide some calls of a method in some part of an object life cycle, whereas some other calls of the same method in another part of the life cycle are encapsulated or left untouched. It is not clear how an inheritance relation can be defined which allows a different treatment of different calls of the same method. It is definitely more complicated than the definition given above. Before studying such a definition, the need for it should be shown by practical experience with the relations given in this paper.

Also note that life-cycle inheritance is defined in terms of a function composition and not simply as the disjunction of the two definitions of protocol and projection inheritance. The latter would not be a true combination of protocol and projection inheritance. Such a definition would also lack desirable properties such as transitivity.

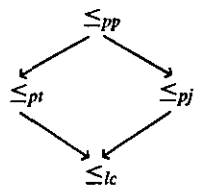

Figure 1: An overview of life-cycle-inheritance relations.

Figure 1 gives an overview of the four inheritance relations. The arrows depict strict inclusion relations. It follows immediately from the definitions that the inclusion relations between protocol/projection inheritance, on the one hand, and protocol and projection inheritance, on the other hand, are correct; the other two

inclusion relations follow from the definitions and the following lemma, which implies that encapsulating or hiding the empty set of methods yields the original life cycle.

Lemma 3.4. *For any closed term* $p \in \mathcal{C}(\mathrm{PA}^{\tau}_{\delta\rho})$ *and sets* $H, I \subseteq L$,

i) $\alpha(p) \cap H = \emptyset \Rightarrow \mathrm{PA}^{\tau}_{\delta\rho} \vdash \partial_H(p) = p$,

ii) $\alpha(p) \cap I = \emptyset \Rightarrow \mathrm{PA}^{\tau}_{\delta\rho} \vdash \tau_I(p) = p$.

Proof. We only give a proof of *i*). It is a straightforward proof by structural induction which is illustrative for the other part of Lemma 3.4, as well as for several other lemmas to follow.

It follows from Property 2.6 that there exists a basic BPA_{δ} term p' such that $\mathrm{PA}^{\tau}_{\delta\rho} \vdash p = p'$. It suffices to show that $\partial_H(p') = p'$. Given this result, it easily follows that $\partial_H(p) = \partial_H(p') = p' = p$. The proof is by induction on the structure of basic BPA_{δ} terms. The $\equiv$ sign denotes structural equivalence. Note that p' cannot contain actions which were not already elements of $\alpha(p)$ (Property 2.4). Hence, $\alpha(p') \cap H = \emptyset$. First, assume that $p' \equiv a$, for some $a \in L_s \cup \{\delta\}$. Since a is not an element of H, it follows immediately that $\partial_H(a) = a$. Second, assume $p' \equiv a \cdot t$, for some $a \in L_s$ and $t \in \mathcal{B}(\mathrm{BPA}_{\delta})$. From $\partial_H(a \cdot t) = a \cdot \partial_H(t)$, it follows by induction that $\partial_H(a \cdot t) = a \cdot t$. Finally, assume $p' \equiv s + t$, for some $s, t \in \mathcal{B}(\mathrm{BPA}_{\delta})$. Again, it follows by induction that $\partial_H(s + t) = \partial_H(s) + \partial_H(t) = s + t$. □

By means of a few examples, it is straightforward to show that the inclusion relations in Figure 1 are indeed strict and that there are no inclusion relations between protocol inheritance and projection inheritance.

Example 3.5. Consider again the running example. Example 3.1 shows that $\mathit{birth} \cdot ((\mathit{marriage} + \mathit{stay\_single}) \cdot \mathit{death} \leq_{pi} \mathit{birth} \cdot (\mathit{marriage} \cdot \mathit{death})$. It is not hard to see that $\tau_{\{\mathit{stay\_single}\}}(\mathit{birth} \cdot ((\mathit{marriage} + \mathit{stay\_single}) \cdot \mathit{death})) = (\mathit{birth} \cdot ((\mathit{marriage} + \tau) \cdot \mathit{death})) \neq \mathit{birth} \cdot (\mathit{marriage} \cdot \mathit{death})$. Therefore, in Figure 1, there is no arrow from protocol inheritance to projection inheritance. This example also shows that the inclusion between $\leq_{pp}$ and $\leq_{pi}$ is strict.

Example 3.2 shows that $\mathit{birth} \cdot ((\mathit{marriage} \cdot \mathit{divorce}) \cdot \mathit{death}) \leq_{pj} \mathit{birth} \cdot (\mathit{marriage} \cdot \mathit{death})$. It is also straightforward to verify that there does not exist a protocol-inheritance relation between the two life cycles, which explains the absence of an arrow from $\leq_{pj}$ to $\leq_{pi}$ as well as the fact that the inclusion between $\leq_{pp}$ and $\leq_{pj}$ is strict.

In order to show that protocol/projection inheritance is not an empty relation, consider the following example. A person who can choose to engage before getting married or who simply wants to get married without engagement is a subclass of the person of Example 2.2 under protocol/projection inheritance: $(\mathit{birth} \cdot ((\mathit{engagement} \cdot \mathit{marriage} + \mathit{marriage}) \cdot \mathit{death})) \leq_{pp} \mathit{birth} \cdot (\mathit{marriage} \cdot \mathit{death})$.

Finally, to show that $\leq_{pi}$ is a strict inclusion of $\leq_{lc}$, as well as that $\leq_{pj}$ is a strict inclusion of $\leq_{lc}$, consider yet another example. A person that can decide to first marry and then divorce or to stay single without worrying about marriage, is a subclass of our person of Example 2.2: $(\mathit{birth} \cdot ((\mathit{marriage} \cdot \mathit{divorce} + \mathit{stay\_single}) \cdot \mathit{death})) \leq_{lc} \mathit{birth} \cdot (\mathit{marriage} \cdot \mathit{death})$. It is not difficult to see that there is no relation between the first person and the second person under any of the other inheritance relations.

Projection inheritance is also suggested by Wieringa in [10]. However, based on an example similar to the last one above, Wieringa concludes that projection inheritance is not a proper definition for inheritance of dynamic behavior. We agree that, in general, projection inheritance is too restricted. However, it is interesting to study what rules are valid for projection inheritance, or, for that matter, for any of the other definitions of inheritance. At the end of the next section, we return to this point in an example.

The following property shows that for protocol, projection, and protocol/projection inheritance, there exists a canonical set of methods that can be encapsulated and/or hidden, namely the set of methods new in the subclass. This result conforms to the intuitive definitions of inheritance given at the beginning of this section.

Property 3.6. *For any closed terms* $p, q \in \mathcal{C}(\mathrm{PA}^\tau)$,

i) $p \leq_{pt} q \Leftrightarrow \partial_{\alpha(p)\setminus\alpha(q)}(p) = q$,
ii) $p \leq_{pj} q \Leftrightarrow \tau_{\alpha(p)\setminus\alpha(q)}(p) = q$,
iii) $p \leq_{pp} q \Leftrightarrow \partial_{\alpha(p)\setminus\alpha(q)}(p) = q \wedge \tau_{\alpha(p)\setminus\alpha(q)}(p) = q$.

Proof. We only prove Property *i)*. The proof of *ii)* is similar; Property *iii)* follows from *i)* and *ii)*.

It follows from the definition of protocol inheritance that $\partial_{\alpha(p)\setminus\alpha(q)}(p) = q \Rightarrow p \leq_{pt} q$. To prove the other implication, assume $p \leq_{pt} q$. It follows from Lemma 3.7 *iii)* and Lemma 3.4 *i)* that there exists an $H \subseteq \alpha(p)$ such that $\partial_H(p) = q$. It follows from Lemma 3.7 *i)* that H cannot contain any methods in $\alpha(q)$. Again Lemma 3.7 *iii)* and Lemma 3.4 *i)* yield that H can be extended to all elements of $\alpha(p)$ which are not elements of $\alpha(q)$. Hence, $\partial_{\alpha(p)\setminus\alpha(q)}(p) = q$. □

Property 3.6 does not have a counterpart for life-cycle inheritance. That is, given closed PA^τ terms p and q such that $p \leq_{lc} q$, there does not exist a canonical partitioning of $\alpha(p) \setminus \alpha(q)$ into H and I such that $\tau_I \circ \partial_H(p) = q$. Assume, for example, that $p \leq_{pp} q$. It follows from Property 3.6 *iii)* that for $H = \alpha(p)\setminus\alpha(q)$ and $I = \emptyset$, as well as for $H = \emptyset$ and $I = \alpha(p) \setminus \alpha(q)$, $\tau_I \circ \partial_H(p) = q$.

Lemma 3.7. *For any closed term* $p \in \mathcal{C}(\mathrm{PA}^\tau_{\delta\rho})$, $H, H', I, I' \subseteq L$,

i) $\alpha(\partial_H(p)) \cap H = \emptyset$,
ii) $\alpha(\tau_I(p)) \cap I = \emptyset$,
iii) $\mathrm{PA}^\tau_{\delta\rho} \vdash \partial_{H'\cup H}(p) = \partial_{H'} \circ \partial_H(p)$,
iv) $\mathrm{PA}^\tau_{\delta\rho} \vdash \tau_{I'\cup I}(p) = \tau_{I'} \circ \tau_I(p)$,

Proof. Structural induction on BPA_δ terms. □

The following property shows that the four inheritance relations are reflexive and transitive, strengthening our belief that the relations are defined properly.

Property 3.8. *Protocol, projection, protocol/projection, and life-cycle inheritance are preorders.*

Proof. It follows from Lemma 3.4 *i)* and *ii)* that, for any $p \in \mathcal{C}(\mathrm{PA}^\tau)$, $\partial_\emptyset(p) = p$ and $\tau_\emptyset(p) = p$. Hence, $\leq_{pt}$, $\leq_{pj}$, $\leq_{pp}$, and $\leq_{lc}$ are reflexive. For the first three relations, it is also straightforward to show that they are transitive. Let $p, q, r \in \mathcal{C}(\mathrm{PA}^\tau)$ such that $p \leq_{pt} q$ and $q \leq_{pt} r$. Assume $H, H' \subseteq L$ are such that $\partial_H(p) = q$ and $\partial_{H'}(q) = r$. Using Lemma 3.7 *iii)*,

$\partial_{H'\cup H}(p) = \partial_{H'} \circ \partial_H(p) = \partial_{H'}(q) = r$.

Hence, $p \leq_{pt} r$, which proves transitivity of protocol inheritance. Using Lemma 3.7 *iii)* and *iv)*, the proofs for $\leq_{pj}$ and $\leq_{pp}$ are very similar.

Showing that life-cycle inheritance is transitive is more involved. The crucial point is the observation that, given $p \leq_{lc} q$ and $q \leq_{lc} r$, it is not possible that methods in p are encapsulated (hidden) whereas the *same methods* in q are hidden (encapsulated). The reason is simple: Methods of p that are encapsulated or hidden, simply do not occur in q anymore. Formally, the proof is as follows. From the assumption that $p \leq_{lc} q$ and $q \leq_{lc} r$, it follows that there are subsets H, H', I, and I' of L such that $\tau_I \circ \partial_H(p) = q$ and $\tau_{I'} \circ \partial_{H'}(q) = r$. Lemmas 3.4 and 3.7 imply that these sets can be chosen such that H and I are subsets of $\alpha(p) \setminus \alpha(q)$ and such that H' and I' are subsets of $\alpha(q) \setminus \alpha(r)$. It follows from Lemma 3.9 *i)* and *ii)* that $\alpha(r) \subseteq \alpha(q) \subseteq \alpha(p)$, and hence that $(H \cup I) \cap (H' \cup I') = \emptyset$. Using Lemmas 3.7 *iii)*, *iv)* and 3.9 *iii)*, it follows that

$\tau_{I'\cup I} \circ \partial_{H'\cup H}(p) = \tau_{I'} \circ \tau_I \circ \partial_{H'} \circ \partial_H(p) = \tau_{I'} \circ \partial_{H'} \circ \tau_I \circ \partial_H(p) = \tau_{I'} \circ \partial_{H'}(q) = r$.

Hence, $p \leq_{lc} r$. □

Lemma 3.9. *For any closed term* $p \in \mathcal{C}(\mathrm{PA}^\tau_{\delta\rho})$, $H, I \subseteq L$,

i) $\alpha(\partial_H(p)) \subseteq \alpha(p)$,
ii) $\alpha(\tau_I(p)) \subseteq \alpha(p)$,
iii) $H \cap I = \emptyset \Rightarrow \mathrm{PA}^{\tau}_{\delta\rho} \vdash \tau_I \circ \partial_H(p) = \partial_H \circ \tau_I(p)$.

Proof. Structural induction on BPA_δ terms. □

Any preorder induces an equivalence relation. The meaning of the equivalence relations induced by the inheritance preorders is "subclass equivalence" under the corresponding form of inheritance. Intuitively, two life cycles should be subclass equivalent under any form of inheritance if and only if their equality is derivable from the axioms of $\mathrm{PA}^{\tau}_{\delta\rho}$.

Definition 3.10. (Subclass equivalence) Let $\approx_*$, where $* \in \{pp,pt,pj,lc\}$, be the equivalence relation induced by $\leq_*$. That is, for any closed PA^{τ} terms p and q, $p \approx_* q \Leftrightarrow p \leq_* q \wedge q \leq_* p$. Processes p and q are said to be *subclass equivalent* under $*$ inheritance.

The following result states that the four subclass-equivalence relations indeed all coincide with derivability from the axioms of $\mathrm{PA}^{\tau}_{\delta\rho}$. The completeness result of the previous section implies that two life cycles are subclass equivalent under any form of inheritance if and only if they are rooted branching bisimilar.

Property 3.11. *For any* $p, q \in C(\mathrm{PA}^{\tau})$ *and* $* \in \{pp,pt,pj,lc\}$, $p \approx_* q \Leftrightarrow \mathrm{PA}^{\tau}_{\delta\rho} \vdash p = q$.

Proof. We only prove the result for protocol inheritance. The other proofs are similar. First, assume that $\mathrm{PA}^{\tau}_{\delta\rho} \vdash p = q$. It follows from Lemma 3.4 *i)* that $\partial_\emptyset(p) = p = q$ and $\partial_\emptyset(q) = q = p$. Hence, $p \leq_{pt} q$ and $q \leq_{pt} p$, which in turn implies that $p \approx_{pt} q$.

Second, assume $p \approx_{pt} q$, which implies that $p \leq_{pt} q$ and $q \leq_{pt} p$. It follows from Property 3.6 *i)* that $\partial_{\alpha(p)\setminus\alpha(q)}(p) = q$ and $\partial_{\alpha(q)\setminus\alpha(p)}(q) = p$. It follows from Lemma 3.9 *i)* that $\alpha(q) \subseteq \alpha(p)$ and $\alpha(p) \subseteq \alpha(q)$. Hence, $\alpha(p) = \alpha(q)$, which means that $\alpha(p) \setminus \alpha(q) = \alpha(q) \setminus \alpha(p) = \emptyset$. It follows from Lemma 3.4 *i)* that $\partial_{\alpha(p)\setminus\alpha(q)}(p) = p$ and hence that $\mathrm{PA}^{\tau}_{\delta\rho} \vdash p = q$.

Note that since Property 3.6 does not have a counterpart for life-cycle inheritance, one has to use the more basic results of Lemmas 3.4, 3.7, and 3.9 to prove the property for life-cycle inheritance. □

It is important to note that the inheritance preorders are not precongruences for the operators of PA^{τ}. That is, it is not possible to apply them in arbitrary contexts.

Example 3.12. It is easy to see that, for any distinct $a, b \in L$, $a + b \leq_{pt} a$, because $\partial_{\{b\}}(a + b) = a$. However, in a context where an occurrence of b is followed by an a, it is not allowed to replace a by its subclass $a + b$. Doing so, yields $b \cdot (a + b)$. Obviously, since $\partial_{\{b\}}(b \cdot (a + b)) = \delta$, $b \cdot (a + b) \not\leq_{pt} b \cdot a$. A similar example can be constructed for projection inheritance. It is straightforward to show that $a \cdot b \leq_{pj} a$, whereas $a \cdot b + b \not\leq_{pj} a + b$. However, for any method $c \in L$ that is different from b, it easily follows that $c \cdot (a + b) \leq_{pt} c \cdot a$ and $a \cdot b + c \leq_{pj} a + c$.

Another instructive example is the following, where first a subclass of an action a under protocol inheritance is constructed and, subsequently, this subclass is further refined to a more specialized subclass. For any distinct $a, b, c, d \in L$, $a + b \leq_{pt} a$ and $a \cdot c + b \leq_{pt} b$. Replacing the occurrence of b in the former with its subclass $a \cdot c + b$, yields $a + (a \cdot c + b)$. It is not difficult to see that $a + (a \cdot c + b) \not\leq_{pt} a$. Another subclass of b is $d \cdot c + b$. Replacing b with this subclass, yields $a + (d \cdot c + b)$. It easily follows that $a + (d \cdot c + b) \leq_{pt} a$.

These examples show that a problem may arise when a method that is encapsulated or hidden also appears in the context. The fact that the inheritance relations cannot be applied in arbitrary contexts is not really unexpected. The reason is that it is not allowed to treat different calls of the same method in a different way. In de remainder of this section, we formalize under which conditions it is allowed to refine a subclass to a more specialized subclass. First, we formalize the notion of a context. A context $C[\_]$ is a closed PA^{τ} term that contains a "hole." Contexts can be defined inductively in the following way.

Definition 3.13. (Context) The most simple context is simply a hole, denoted by an underscore "_". Furthermore, for any closed term $p \in C(\mathrm{PA}^\tau)$ and context $C[\_]$, $p \oplus C[\_]$ and $C[\_] \oplus p$ are contexts, where $\oplus \in \{\cdot, +, \parallel, \between\}$.

The following property gives the conditions we are looking for. Informally, the condition in Property *i)* that $\alpha(r) \cap H = \emptyset$ means that it is not allowed to encapsulate methods in p that are *not* encapsulated in $C[q]$; the former being methods in H and the latter being methods in $\alpha(r)$. The conditions in the other properties have similar meanings. The additional requirement in Property *iv)* that $(H' \cup H) \cap (I' \cup I) = \emptyset$ means that the methods being encapsulated must be disjoint from the methods being hidden.

Property 3.14. *Let $p, q, r \in C(\mathrm{PA}^\tau)$ and let $C[\_]$ be a context. Let H, H', I, and I' be subsets of L.*

i) If $p \leq_{pt} q$ such that $\partial_H(p) = q$, and $C[q] \leq_{pt} r$, then $\alpha(r) \cap H = \emptyset \Rightarrow C[p] \leq_{pt} r$.

ii) If $p \leq_{pj} q$ such that $\tau_I(p) = q$, and $C[q] \leq_{pj} r$, then $\alpha(r) \cap I = \emptyset \Rightarrow C[p] \leq_{pj} r$.

iii) If $p \leq_{pp} q$ such that $\partial_H(p) = q$ and $\tau_I(p) = q$, and $C[q] \leq_{pp} r$, then $\alpha(r) \cap (H \cup I) = \emptyset \Rightarrow C[p] \leq_{pp} r$.

iv) If $p \leq_{lc} q$ such that $\tau_I \circ \partial_H(p) = q$, and $C[q] \leq_{lc} r$ such that $\tau_{I'} \circ \partial_{H'}(C[q]) = r$, then $(H' \cup H) \cap (I' \cup I) = \emptyset \wedge \alpha(r) \cap (H \cup I) = \emptyset \Rightarrow C[p] \leq_{lc} r$.

Proof. First, we prove Property *i)*. It follows from the encapsulation axioms and Lemma 3.15 given below that encapsulation distributes over all operators that may appear in a context. It then follows from Lemmas 3.4, 3.7, and 3.9 that the following derivation is correct. In the final step, the requirement that $\alpha(r) \cap H = \emptyset$ is used.

$\partial_{H' \cup H}(C[p]) = \partial_{H'} \circ \partial_H(C[p]) = \partial_{H'} \circ \partial_H(C[\partial_H(p)]) = \partial_{H'} \circ \partial_H(C[q]) = \partial_{H' \cup H}(C[q]) = \partial_{H \cup H'}(C[q]) = \partial_H \circ \partial_{H'}(C[q]) = \partial_H(r) = r.$

This derivation shows that $C[p] \leq_{pt} r$. Properties *ii)* and *iii)* are proven similarly. Property *iv)* is shown as follows. The condition that $(H' \cup H) \cap (I' \cup I) = \emptyset$ is used in the second and fifth step. As before, the other condition is used in the final step.

$\tau_{I' \cup I} \circ \partial_{H' \cup H}(C[p]) = \tau_{I'} \circ \tau_I \circ \partial_{H'} \circ \partial_H(C[p]) = \tau_{I'} \circ \partial_{H'} \circ \tau_I \circ \partial_H(C[p]) = \tau_{I'} \circ \partial_{H'} \circ \tau_I \circ \partial_H(C[\tau_I \circ \partial_H(p)] = \tau_{I'} \circ \partial_{H'} \circ \tau_I \circ \partial_H(C[q]) = \tau_I \circ \partial_H \circ \tau_{I'} \circ \partial_{H'}(C[q]) = \tau_I \circ \partial_H(r) = r.$

□

Lemma 3.15. *For any closed terms $p, q \in C(\mathrm{PA}^\tau_{\delta\rho})$, $H, I \subseteq L$,*

i) $\mathrm{PA}^\tau_{\delta\rho} \vdash \partial_H(p \between q) = \partial_H(p) \between \partial_H(q)$,

ii) $\mathrm{PA}^\tau_{\delta\rho} \vdash \partial_H(p \parallel q) = \partial_H(p) \parallel \partial_H(q)$,

iii) $\mathrm{PA}^\tau_{\delta\rho} \vdash \tau_I(p \between q) = \tau_I(p) \between \tau_I(q)$,

iv) $\mathrm{PA}^\tau_{\delta\rho} \vdash \tau_I(p \parallel q) = \tau_I(p) \parallel \tau_I(q)$,

Proof. Property *i)* can be proven by simultaneous induction on the structure of p and the sum of the number of symbols in p and in q. Property *ii)* follows immediately from *i)*. Properties *iii)* and *iv)* are proven in a similar way. □

Example 3.16. Let us return to Example 3.12. Let context $C_0[\_]$ be defined as $b \cdot \_$ and $C_1[\_]$ as $c \cdot \_$, for distinct methods b and c in L. Since $\leq_{pt}$ is reflexive, $C_0[a] \leq_{pt} C_0[a]$ and $C_1[a] \leq_{pt} C_1[a]$, for any $a \in L$ distinct from b and c. We want to know whether it is possible to refine $C_i[a]$, for $i \in \{0, 1\}$, by replacing the occurrence of a by its subclass $a + b$. That is, we want to find out whether $C_i[a + b]$ is a subclass of $C_i[a]$. Note that to show that $a + b$ is a subclass of a, method b is being encapsulated. Since $b \in \alpha(C_0[a])$, we cannot derive from Property 3.14 *i)* that $C_0[a + b]$ is a subclass of $C_0[a]$, which confirms the conclusion of Example 3.12. However, since $b \notin \alpha(C_1[a])$, Property 3.14 *i)* yields that $C_1[a + b] \leq_{pt} C_1[a]$.

Another example shows an application of Property 3.14 *iv*). Let $a, b, c \in L$ be distinct methods. Consider the context $C[\_]$ defined as $\_ + b$. By encapsulating method b, it is easy to show that $C[a] \leq_{lc} a$. We also have that $a \cdot c \leq_{lc} a$, which follows from hiding method c. Replacing the occurrence of a in $C[a]$ with its subclass $a \cdot c$, yields $C[a \cdot c]$. In order to apply Property 3.14 *iv*), let $H' = \{b\}$, $I = \{c\}$, and $H = I' = \emptyset$. Obviously, this satisfies the requirement that $(H' \cup H) \cap (I' \cup I) = \emptyset$. Since, in addition, $\alpha(a)$ and $H \cup I$ are disjoint, we derive that $C[a \cdot c] \leq_{lc} a$. That is, $a \cdot c + b \leq_{lc} a$. However, $a \cdot b$ is also a subclass of a, which can be easily shown by hiding the singleton $I = \{b\}$. Substituting $a \cdot b$ for a in $C[a]$ yields $C[a \cdot b]$. Since in this case $(H' \cup H) \cap (I' \cup I)$, where H, H', and I' are as before, is not empty, we cannot apply Property 3.14 *iv*). It is not difficult to verify that $C[a \cdot b]$ is not a subclass of a: $C[a \cdot b] \not\leq_{lc} a$. The reason is that in $C[a]$ method b is encapsulated whereas in $a \cdot b$ method b is hidden.

The results presented in this section show that the definitions of the four inheritance relations are sound. The conditions under which it is allowed to apply the inheritance preorders in arbitrary contexts or to refine a subclass to a more specialized subclass seem reasonable, although more experience has to show whether they are not too restrictive. The examples give an impression what subclass relations are valid under the different forms of inheritance. In the next section, we study the latter in more detail.

4 Subclass Relations under Different Forms of Inheritance

This section presents some general subclass relations valid under the four forms of inheritance. In particular, it shows that extending a life cycle with an alternative behavior yields a subclass under protocol inheritance. It also shows that extending a life cycle by inserting some alternative behavior in between parts of the original life cycle, as well as putting some alternative behavior in parallel with (part of) the original life cycle yields a subclass under projection inheritance. Life-cycle inheritance allows arbitrary combinations of these three operations.

Property 4.1. *For any* $p, q \in C(\mathrm{PA}^\tau)$ *and* $b \in L \setminus \alpha(q)$,

$$\frac{}{q + b \cdot p \leq_{pt} q} \quad PT$$

Proof. Let $H = \{b\}$. It follows from the encapsulation axioms, Lemma 3.4 *i*), and axioms $A7$ and $A6$ that
$\partial_H(q + b \cdot p) = \partial_H(q) + \partial_H(b) \cdot \partial_H(p) = q + \delta \cdot \partial_H(p) = q + \delta = q$.
Hence, $q + b \cdot p \leq_{pt} q$. □

Method b functions as some sort of a "*guard.*" Blocking the guard means that the enviroment cannot choose the behavior $b \cdot p$. For this reason, b may not appear in q, since blocking b would otherwise change the behavior of q. Rule PT shows that encapsulation is sufficient to capture inheritance by means of the choice operator. This rule also shows that it is sufficient to encapsulate a single method, whereas the canonical set $\alpha(q + b \cdot p) \setminus \alpha(q)$ might be much larger.

Property 4.2. *For any* $a \in L_s$ *and* $q, q_0, q_1, r \in C(\mathrm{PA}^\tau)$ *such that* $\alpha(r) \subseteq L \setminus (\alpha(q) \cup \alpha(q_0) \cup \alpha(q_1) \cup \{a\})$,

$$\frac{}{\begin{array}{ll} q \cdot r \leq_{pj} q & PJ1 \\ a \cdot (r \cdot (q_0 + q_1) + q_0) \leq_{pj} a \cdot (q_0 + q_1) & PJ2 \\ a \cdot (q \parallel r) \leq_{pj} a \cdot q & PJ3 \end{array}}$$

Proof. Let I be equal to $\alpha(r)$. It follows from the abstraction axioms, Lemma 3.4 *ii*), Lemma 4.3 *i*) given below, and axiom $B1$ that
$\tau_I(q \cdot r) = \tau_I(q) \cdot \tau_I(r) = q \cdot \tau = q$,
which proves $PJ1$. The proof for $PJ2$ is similar, only axiom $B2$ is used instead of $B1$:
$\tau_I(a \cdot (r \cdot (q_0 + q_1) + q_0)) = \tau_I(a) \cdot (\tau_I(r) \cdot (\tau_I(q_0) + \tau_I(q_1)) + \tau_I(q_0)) = a \cdot (\tau \cdot (q_0 + q_1) + q_0) = a \cdot (q_0 + q_1)$.

Rule $PJ3$ follows from the abstraction axioms, Lemmas 3.4 *ii*), 3.15 *iv*), and 4.3, and axiom $B2$:
$\tau_I(a \cdot (q \parallel r)) = \tau_I(a) \cdot (\tau_I(q) \parallel \tau_I(r)) = a \cdot (q \parallel \tau) = a \cdot (\tau \cdot q + q) = a \cdot q$. □

Lemma 4.3. *For any closed term $p \in C(\mathrm{PA}^\tau)$, and $I \subseteq L$,*

i) $\alpha(p) \subseteq I \Rightarrow \mathrm{PA}^\tau_{\delta\rho} \vdash \tau_I(p) = \tau$,
ii) $\mathrm{PA}^\tau_{\delta\rho} \vdash p \parallel \tau = \tau \cdot p + p$,

Proof. Property *i*) is shown by induction on the structure of basic BPA terms. Property *ii*) can also be proven by structural induction. However, the auxiliary result that $\mathrm{PA}^\tau_{\delta\rho} \vdash p \between \tau = p$ is needed, which is, not unexpectedly, also proven by structural induction. □

Note that Lemma 4.3 *i*) cannot be proven for arbitrary closed $\mathrm{PA}^\tau_{\delta\rho}$ terms, because it is not possible to hide constant δ.

Rules $PJ1$ and $PJ2$ are inspired by the two branching axioms $B1$ and $B2$. Together they state that inserting new behavior in an object life cycle that does not disable any behavior of the original life cycle, yields a subclass under projection inheritance. Rule $PJ3$ shows that putting alternative behavior in parallel with the original life cycle also yields a subclass under projection inheritance. The last two rules clearly show why the object-creation action $\triangledown$ is useful. They cannot be proven if the initial a is omitted, because an internal action at the beginning of a life cycle may determine a choice and, hence, cannot be removed. The requirement that an object life cycle starts with an object-creation action guarantees that an initial internal action never occurs.

Example 4.4. Consider again the running example. A person that can get children any time during his or her life is a subclass of the person of Example 2.2: $birth \cdot ((marriage \| children) \cdot death) \leq_{pj} birth \cdot (marriage \cdot death)$. This result follows immediately from the context rule of Property 3.14 *ii*) and rule $PJ3$ above.

Property 4.5. *For any $a \in L_s$, $q, r \in C(\mathrm{PA}^\tau)$ such that $\alpha(r) \subseteq L \setminus (\alpha(q) \cup \{a\})$, and $b \in L \setminus (\alpha(q) \cup \{a\})$,*

$$\frac{}{a \cdot ((b \cdot r) \cdot q + q) \leq_{pp} a \cdot q} \qquad PP$$

Proof. Let H be equal to $\{b\}$ and I be equal to $\alpha(r) \cup \{b\}$. Note that b is not equal to τ. It follows immediately from the encapsulation axioms, Lemma 3.4 *i*), and axioms $A7$ and $A6$ that
$\partial_H(a \cdot ((b \cdot r) \cdot q + q)) = \partial_H(a) \cdot (\partial_H(b \cdot r) \cdot \partial_H(q) + \partial_H(q)) = a \cdot ((\delta \cdot \partial_H(r)) \cdot q + q) = a \cdot q$.
Furthermore, it follows from the abstraction axioms, Lemmas 3.4 *ii*), and 4.3 *i*), and axioms $B2$ and $A3$ that
$\tau_I(a \cdot ((b \cdot r) \cdot q + q)) = \tau_I(a) \cdot (\tau_I(b \cdot r) \cdot \tau_I(q) + \tau_I(q)) = a \cdot (\tau \cdot q + q) = a \cdot q$.
It follows from the above two derivations that $a \cdot ((b \cdot r) \cdot q + q) \leq_{pp} a \cdot q$. □

Rule PP shows that under protocol/projection inheritance it is allowed to *postpone* behavior. In case it is possible to specify recursive behavior, a nice variant of PP is a rule in which the behavior $b \cdot r$ is iterated arbitrary many times before continuing with q. An example of such a rule with iteration can be found in [1].

Property 4.6. *For any $a \in L_s$, $p, q, r \in C(\mathrm{PA}^\tau)$ such that $\alpha(r) \subseteq L \setminus (\alpha(q) \cup \{a\})$, and $b \in L \setminus (\alpha(r) \cup \alpha(q) \cup \{a\})$,*

$$\frac{}{\begin{array}{ll} a \cdot (r \cdot (q + b \cdot p)) \leq_{lc} a \cdot q & LC1 \\ a \cdot ((q + b \cdot p) \parallel r) \leq_{lc} a \cdot q & LC2 \\ a \cdot (q \parallel (r + b \cdot p)) \leq_{lc} a \cdot q & LC3 \end{array}}$$

Rules $LC1$-3 are combinations of the rules for protocol and projection inheritance, which are illustrative for life-cycle inheritance. It is not difficult to find several more of such combinations. It is possible to prove the above rules by means of Property 3.14. However, the proof given below uses basic lemmas and axioms, which yields a more readable proof. The other proof is an interesting exercise.

Proof. Let H be equal to $\{b\}$ and I be equal to $\alpha(r)$. It follows from the encapsulation and abstraction axioms, Lemma 3.4, Lemma 4.3 *i*), and axioms $A6$, $A7$, and $B1$ that

$\tau_I \circ \partial_H(a \cdot (r \cdot (q + b \cdot p))) = \tau_I \circ \partial_H(a) \cdot (\tau_I \circ \partial_H(r) \cdot (\tau_I \circ \partial_H(q) + \tau_I \circ \partial_H(b \cdot p))) = a \cdot (\tau \cdot (q + \delta)) = a \cdot q$,

which proves $LC1$. Rule $LC2$ follows from again the encapsulation and abstraction axioms, Lemma 3.4, Lemma 3.15 *ii*) and *iv*), Lemma 4.3, and axioms $A6$, $A7$, and $B2$:

$\tau_I \circ \partial_H(a \cdot ((q + b \cdot p) \parallel r)) = \tau_I(\partial_H(a) \cdot (\partial_H(q + b \cdot p) \parallel \partial_H(r))) = \tau_I(a \cdot ((q + \delta) \parallel r)) = \tau_I(a \cdot (q \parallel r))$,

after which the proof proceeds as for rule $PJ3$. Rule $LC3$ is proven similarly. □

Example 4.7. Let us return to our running example. A person that can get children any time during his or her life, and can choose to marry or to stay single is a subclass of the person of Example 2.2: $birth \cdot (((marriage + stay\_single) \parallel children) \cdot death) \leq_{lc} birth \cdot (marriage \cdot death)$. This result follows from the context rule of Property 3.14 *iv*) and rule $LC2$.

Example 4.8. The following example is taken from [10], where it is used to illustrate the shortcomings of projection inheritance in isolation. Although we have not formalized recursion in this paper, it should be clear that the following calculations are correct. Assume we have the following equations:

$X = a \cdot X$

$Y_1 = b_1 \cdot Y_2$

$Y_2 = (b_2 + a) \cdot Y_2 + b_3$

In [10], X is the life cycle of a person and Y the life cycle of an employee. Action a denotes a change of address; b_1 is the hiring of an employee; b_2 is a promotion and b_3 denotes the employee leaving the job. Obviously, an employee should be a subclass of a person. However, it is not hard to show that Y_1 is not a subclass of X under projection inheritance. Let $I = \{b_1, b_2, b_3\}$.

$\tau_I(Y_1) = \tau \cdot \tau_I(Y_2)$

$\tau_I(Y_2) = (\tau + a) \cdot \tau_I(Y_2) + \tau$

It is clear that $\tau_I(Y_1)$ is not equal to X which means that $Y_1 \not\leq_{pj} X$. The problem is two-fold. The initial τ of $\tau_I(Y_1)$ cannot be removed, nor can the two τ-actions in the context of a choice in process $\tau_I(Y_2)$ (although the first τ in $\tau_I(Y_2)$ can be eliminated if some kind of fairness principle is used, see [5]). The first problem can be solved easily. Observe that in the framework of this paper X and Y_1 do not denote object life cycles. Therefore, we add the following two equations:

$X_0 = \triangledown \cdot X$

$Y_0 = \triangledown \cdot Y_1$

Process X_0 now denotes the life cycle of a person; Y_0 is the life cycle of an employee. Note that the defining equations for Y_0 can be simplified as follows.

$Y_0 = (\triangledown \cdot b_1) \cdot Y_2$

$Y_2 = (b_2 + a) \cdot Y_2 + b_3$

Although this solves the first problem, it is still not possible to show that $Y_0 \leq_{pj} X_0$. However, it is possible to show that an employee is a subclass of a person under the more general life-cycle inheritance. That is, it is possible to show that $Y_0 \leq_{lc} X_0$. Let $H = \{b_2, b_3\}$ and $I = \{b_1\}$. The axioms for encapsulation and abstraction yield that

$\tau_I \circ \partial_H(Y_0) = (\triangledown \cdot \tau) \cdot \tau_I \circ \partial_H(Y_2)$

$\tau_I \circ \partial_H(Y_2) = (\delta + a) \cdot \tau_I \circ \partial_H(Y_2) + \delta$

It follows from axioms $B1$ and $A6$ that

$\tau_I \circ \partial_H(Y_0) = \triangledown \cdot \tau_I \circ \partial_H(Y_2)$

$\tau_I \circ \partial_H(Y_2) = a \cdot \tau_I \circ \partial_H(Y_2)$

Hence, $\tau_I \circ \partial_H(Y_0) = X_0$, which implies that $Y_0 \leq_{lc} X_0$.

It is also possible to arrive at this result by applying the rules given in this section plus several of the properties of the previous section. First, rule $PJ1$ yields that $\triangledown \cdot b_1 \leq_{lc} \triangledown$. Since method b_1 does not appear in X_0, we derive that

$(\nabla \cdot b_1) \cdot X \cdot \leq_{lc} X_0$.

Second, by applying rule *PT* twice, it follows that $Y_2 \leq_{lc} X$. The context property of the previous section implies that

$(\nabla \cdot b_1) \cdot Y_2 \leq_{lc} X_0$.

As before, it follows that $Y_0 \leq_{lc} X_0$.

As Wieringa already observes in [10], abstraction is a useful notion for describing life-cycle-inheritance relations, but it does not always appear to be sufficient. In that case, the combination of abstraction and encapsulation seems to be a solution.

5 Concluding Remarks

This paper presents a characterization of life-cycle inheritance in a simple process-algebraic setting. The reason to choose process algebra is two-fold. First, we believe that life-cycle inheritance can be expressed in terms of *encapsulation* and *abstraction*. These concepts are well investigated in process algebra. Second, process algebra has no explicit representation of process states, but focusses on state changes instead. This is an advantage when studying the concept of inheritance of dynamic behavior.

We have formulated four inheritance relations, all in terms of encapsulation and abstraction. We have shown that all four relations are sound and have several useful properties. Numerous examples and some general rules show that the definitions are meaningful as well, although more practical experience has to point out whether they are expressive enough to cope with real-world examples. The rules given in Section 4 show that they capture three basic operators to construct a subclass from a superclass, namely choice, sequential composition, and parallel composition.

A disadvantage of the theory developed in this paper as a whole is that it is not immediately useful for practical purposes. The algebraic theory does not include recursion and it has no explicit representation of internal states. Moreover, it is not straightforward to add these features and still maintain a framework as simple and clear as the theory in this paper. Therefore, in another paper [1], we turn to Petri nets, which is a graphical formalism, much closer to the state diagrams used in existing techniques as OMT and OOD. Petri nets inherently allow recursion and they have an explicit representation of states, two features that are essential to a successful object-oriented methodology. The translation of the concepts developed in this paper to Petri nets in [1] is illustrative for translations to other state-based formalisms. The reason for the approach in two steps is that process algebra is a very useful framework for developing a good understanding of the *concept* of life-cycle inheritance, whereas Petri nets are a formalism which is very intuitive to understand and is close to practice. It is our experience that it is difficult to achieve the two goals of a clear conceptual understanding and a practical object-oriented methodology in a single framework.

Acknowledgements. The authors want to thank Jos Baeten, Marc Voorhoeve, and Michel Reniers for the valuable discussions and their useful comments on earlier versions of this paper.

References

1. W.M.P. van der Aalst and T. Basten. Life-Cycle Inheritance: A Petri-Net-Based Approach. Computing Science Report 96/06, Eindhoven University of Technology, Department of Mathematics and Computing Science, Eindhoven, The Netherlands, March 1996.

2. G. Agha et al. Panel discussion at the workshop on Object-Oriented Programming and Models of Concurrency. 16th. International Conference on the Application and Theory of Petri Nets, Torino, Italy, June 1995.

3. J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Conditional Axioms and α/β-calculus in Process Algebra. In M. Wirsing, editor, *Proceedings of the IFIP Conference on Formal Description of Programming Concepts - III*, pages 53–75, Ebberup, Denmark, 1986. North–Holland, Amsterdam, The Netherlands, 1987.

4. J.C.M. Baeten and C. Verhoef. Concrete Process Algebra. In S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4, *Semantic Modelling*, pages 149–268. Oxford University Press, Oxford, UK, 1995.

5. 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.

6. G. Booch. *Object-Oriented Analysis and Design: With Applications*. Benjamin/Cummings, Redwood City, CA, USA, 1994.

7. R.J. van Glabbeek. What is Branching Time Semantics and Why to Use It? In *Bulletin of the EATCS*, number 53, pages 191–198. European Association for Theoretical Computer Science, June 1994.

8. R.J. van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics (extended abstract). In G.X. Ritter, editor, *Information Processing 89: Proceedings of the IFIP 11th. World Computer Congress*, pages 613–618, San Fransisco, CA, USA, August/September 1989. Elsevier Science Publishers B.V., North-Holland, 1989.

9. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1991.

10. R.J. Wieringa. *Algebraic Foundations for Dynamic Conceptual Models*. PhD thesis, Free University, Amsterdam, The Netherlands, 1990.

Computing Science Reports

Department of Mathematics and Computing Science
Eindhoven University of Technology

In this series appeared:

93/01	R. van Geldrop	Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.
93/02	T. Verhoeff	A continuous version of the Prisoner's Dilemma, p. 17
93/03	T. Verhoeff	Quicksort for linked lists, p. 8.
93/04	E.H.L. Aarts J.H.M. Korst P.J. Zwietering	Deterministic and randomized local search, p. 78.
93/05	J.C.M. Baeten C. Verhoef	A congruence theorem for structured operational semantics with predicates, p. 18.
93/06	J.P. Veltkamp	On the unavoidability of metastable behaviour, p. 29
93/07	P.D. Moerland	Exercises in Multiprogramming, p. 97
93/08	J. Verhoosel	A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.
93/09	K.M. van Hee	Systems Engineering: a Formal Approach Part I: System Concepts, p. 72.
93/10	K.M. van Hee	Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
93/11	K.M. van Hee	Systems Engineering: a Formal Approach Part III: Modeling Methods, p. 101.
93/12	K.M. van Hee	Systems Engineering: a Formal Approach Part IV: Analysis Methods, p. 63.
93/13	K.M. van Hee	Systems Engineering: a Formal Approach Part V: Specification Language, p. 89.
93/14	J.C.M. Baeten J.A. Bergstra	On Sequential Composition, Action Prefixes and Process Prefix, p. 21.
93/15	J.C.M. Baeten J.A. Bergstra R.N. Bol	A Real-Time Process Logic, p. 31.
93/16	H. Schepers J. Hooman	A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27
93/17	D. Alstein P. van der Stok	Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.
93/18	C. Verhoef	A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.
93/19	G-J. Houben	The Design of an Online Help Facility for ExSpect, p.21.
93/20	F.S. de Boer	A Process Algebra of Concurrent Constraint Programming, p. 15.
93/21	M. Codish D. Dams G. Filé M. Bruynooghe	Freeness Analysis for Logic Programs - And Correctness, p. 24
93/22	E. Poll	A Typechecker for Bijective Pure Type Systems, p. 28.
93/23	E. de Kogel	Relational Algebra and Equational Proofs, p. 23.
93/24	E. Poll and Paula Severi	Pure Type Systems with Definitions, p. 38.
93/25	H. Schepers and R. Gerth	A Compositional Proof Theory for Fault Tolerant Real-Time Distributed Systems, p. 31.
93/26	W.M.P. van der Aalst	Multi-dimensional Petri nets, p. 25.
93/27	T. Kloks and D. Kratsch	Finding all minimal separators of a graph, p. 11.
93/28	F. Kamareddine and R. Nederpelt	A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.
93/29	R. Post and P. De Bra	GOLD, a Graph Oriented Language for Databases, p. 42.
93/30	J. Deogun T. Kloks D. Kratsch H. Müller	On Vertex Ranking for Permutation and Other Graphs, p. 11.

93/31	W. Körver	Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.
93/32	H. ten Eikelder and H. van Geldrop	On the Correctness of some Algorithms to generate Finite Automata for Regular Expressions, p. 17.
93/33	L. Loyens and J. Moonen	ILIAS, a sequential language for parallel matrix computations, p. 20.
93/34	J.C.M. Baeten and J.A. Bergstra	Real Time Process Algebra with Infinitesimals, p.39.
93/35	W. Ferrer and P. Severi	Abstract Reduction and Topology, p. 28.
93/36	J.C.M. Baeten and J.A. Bergstra	Non Interleaving Process Algebra, p. 17.
93/37	J. Brunekreef J-P. Katoen R. Koymans S. Mauw	Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks, p. 73.
93/38	C. Verhoef	A general conservative extension theorem in process algebra, p. 17.
93/39	W.P.M. Nuijten E.H.L. Aarts D.A.A. van Erp Taalman Kip K.M. van Hee	Job Shop Scheduling by Constraint Satisfaction, p. 22.
93/40	P.D.V. van der Stok M.M.M.P.J. Claessen D. Alstein	A Hierarchical Membership Protocol for Synchronous Distributed Systems, p. 43.
93/41	A. Bijlsma	Temporal operators viewed as predicate transformers, p. 11.
93/42	P.M.P. Rambags	Automatic Verification of Regular Protocols in P/T Nets, p. 23.
93/43	B.W. Watson	A taxonomy of finite automata construction algorithms, p. 87.
93/44	B.W. Watson	A taxonomy of finite automata minimization algorithms, p. 23.
93/45	E.J. Luit J.M.M. Martin	A precise clock synchronization protocol,p.
93/46	T. Kloks D. Kratsch J. Spinrad	Treewidth and Patwidth of Cocomparability graphs of Bounded Dimension, p. 14.
93/47	W. v.d. Aalst P. De Bra G.J. Houben Y. Komatzky	Browsing Semantics in the "Tower" Model, p. 19.
93/48	R. Gerth	Verifying Sequentially Consistent Memory using Interface Refinement, p. 20.
94/01	P. America M. van der Kammen R.P. Nederpelt O.S. van Roosmalen H.C.M. de Swart	The object-oriented paradigm, p. 28.
94/02	F. Kamareddine R.P. Nederpelt	Canonical typing and Π-conversion, p. 51.
94/03	L.B. Hartman K.M. van Hee	Application of Marcov Decision Processe to Search Problems, p. 21.
94/04	J.C.M. Baeten J.A. Bergstra	Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18.
94/05	P. Zhou J. Hooman	Formal Specification and Compositional Verification of an Atomic Broadcast Protocol, p. 22.
94/06	T. Basten T. Kunz J. Black M. Coffin D. Taylor	Time and the Order of Abstract Events in Distributed Computations, p. 29.
94/07	K.R. Apt R. Bol	Logic Programming and Negation: A Survey, p. 62.
94/08	O.S. van Roosmalen	A Hierarchical Diagrammatic Representation of Class Structure, p. 22.
94/09	J.C.M. Baeten J.A. Bergstra	Process Algebra with Partial Choice, p. 16.

94/10	T. verhoeff	The testing Paradigm Applied to Network Structure. p. 31.
94/11	J. Peleska C. Huizing C. Petersohn	A Comparison of Ward & Mellor's Transformation Schema with State- & Activitycharts, p. 30.
94/12	T. Kloks D. Kratsch H. Müller	Dominoes, p. 14.
94/13	R. Seljée	A New Method for Integrity Constraint checking in Deductive Databases, p. 34.
94/14	W. Peremans	Ups and Downs of Type Theory, p. 9.
94/15	R.J.M. Vaessens E.H.L. Aarts J.K. Lenstra	Job Shop Scheduling by Local Search, p. 21.
94/16	R.C. Backhouse H. Doombos	Mathematical Induction Made Calculational, p. 36.
94/17	S. Mauw M.A. Reniers	An Algebraic Semantics of Basic Message Sequence Charts, p. 9.
94/18	F. Kamareddine R. Nederpelt	Refining Reduction in the Lambda Calculus, p. 15.
94/19	B.W. Watson	The performance of single-keyword and multiple-keyword pattern matching algorithms, p. 46.
94/20	R. Bloo F. Kamareddine R. Nederpelt	Beyond β-Reduction in Church's $\lambda\rightarrow$, p. 22.
94/21	B.W. Watson	An introduction to the Fire engine: A C++ toolkit for Finite automata and Regular Expressions.
94/22	B.W. Watson	The design and implementation of the FIRE engine: A C++ toolkit for Finite automata and regular Expressions.
94/23	S. Mauw and M.A. Reniers	An algebraic semantics of Message Sequence Charts, p. 43.
94/24	D. Dams O. Grumberg R. Gerth	Abstract Interpretation of Reactive Systems: Abstractions Preserving $\forall$CTL*, $\exists$CTL* and CTL*, p. 28.
94/25	T. Kloks	$K_{1,3}$-free and W_4-free graphs, p. 10.
94/26	R.R. Hoogerwoord	On the foundations of functional programming: a programmer's point of view, p. 54.
94/27	S. Mauw and H. Mulder	Regularity of BPA-Systems is Decidable, p. 14.
94/28	C.W.A.M. van Overveld M. Verhoeven	Stars or Stripes: a comparative study of finite and transfinite techniques for surface modelling, p. 20.
94/29	J. Hooman	Correctness of Real Time Systems by Construction, p. 22.
94/30	J.C.M. Baeten J.A. Bergstra Gh. Ştefanescu	Process Algebra with Feedback, p. 22.
94/31	B.W. Watson R.E. Watson	A Boyer-Moore type algorithm for regular expression pattern matching, p. 22.
94/32	J.J. Vereijken	Fischer's Protocol in Timed Process Algebra, p. 38.
94/33	T. Laan	A formalization of the Ramified Type Theory, p.40.
94/34	R. Bloo F. Kamareddine R. Nederpelt	The Barendregt Cube with Definitions and Generalised Reduction, p. 37.
94/35	J.C.M. Baeten S. Mauw	Delayed choice: an operator for joining Message Sequence Charts, p. 15.
94/36	F. Kamareddine R. Nederpelt	Canonical typing and Π-conversion in the Barendregt Cube, p. 19.
94/37	T. Basten R. Bol M. Voorhoeve	Simulating and Analyzing Railway Interlockings in ExSpect, p. 30.
94/38	A. Bijlsma C.S. Scholten	Point-free substitution, p. 10.

94/39	A. Blokhuis T. Kloks		On the equivalence covering number of splitgraphs, p. 4.
94/40	D. Alstein		Distributed Consensus and Hard Real-Time Systems, p. 34.
94/41	T. Kloks D. Kratsch		Computing a perfect edge without vertex elimination ordering of a chordal bipartite graph, p. 6.
94/42	J. Engelfriet J.J. Vereijken		Concatenation of Graphs, p. 7.
94/43	R.C. Backhouse M. Bijsterveld		Category Theory as Coherently Constructive Lattice Theory: An Illustration, p. 35.
94/44	E. Brinksma R. Gerth W. Janssen S. Katz M. Poel C. Rump	J. Davies S. Graf B. Jonsson G. Lowe A. Pnueli J. Zwiers	Verifying Sequentially Consistent Memory, p. 160
94/45	G.J. Houben		Tutorial voor de ExSpect-bibliotheek voor "Administratieve Logistiek", p. 43.
94/46	R. Bloo F. Kamareddine R. Nederpelt		The λ-cube with classes of terms modulo conversion, p. 16.
94/47	R. Bloo F. Kamareddine R. Nederpelt		On Π-conversion in Type Theory, p. 12.
94/48	Mathematics of Program Construction Group		Fixed-Point Calculus, p. 11.
94/49	J.C.M. Baeten J.A. Bergstra		Process Algebra with Propositional Signals, p. 25.
94/50	H. Geuvers		A short and flexible proof of Strong Normalazation for the Calculus of Constructions, p. 27.
94/51	T. Kloks D. Kratsch H. Müller		Listing simplicial vertices and recognizing diamond-free graphs, p. 4.
94/52	W. Penczek R. Kuiper		Traces and Logic, p. 81
94/53	R. Gerth R. Kuiper D. Peled W. Penczek		A Partial Order Approach to Branching Time Logic Model Checking, p. 20.
95/01	J.J. Lukkien		The Construction of a small CommunicationLibrary, p.16.
95/02	M. Bezem R. Bol J.F. Groote		Formalizing Process Algebraic Verifications in the Calculus of Constructions, p.49.
95/03	J.C.M. Baeten C. Verhoef		Concrete process algebra, p. 134.
95/04	J. Hidders		An Isotopic Invariant for Planar Drawings of Connected Planar Graphs, p. 9.
95/05	P. Severi		A Type Inference Algorithm for Pure Type Systems, p.20.
95/06	T.W.M. Vossen M.G.A. Verhoeven H.M.M. ten Eikelder E.H.L. Aarts		A Quantitative Analysis of Iterated Local Search, p.23.
95/07	G.A.M. de Bruyn O.S. van Roosmalen		Drawing Execution Graphs by Parsing, p. 10.
95/08	R. Bloo		Preservation of Strong Normalisation for Explicit Substitution, p. 12.
95/09	J.C.M. Baeten J.A. Bergstra		Discrete Time Process Algebra, p. 20
95/10	R.C. Backhouse R. Verhoeven O. Weber		MathJpad: A System for On-Line Prepararation of Mathematical Documents, p. 15

95/11	R. Seljée	Deductive Database Systems and integrity constraint checking, p. 36.
95/12	S. Mauw and M. Reniers	Empty Interworkings and Refinement Semantics of Interworkings Revised, p. 19.
95/13	B.W. Watson and G. Zwaan	A taxonomy of sublinear multiple keyword pattern matching algorithms, p. 26.
95/14	A. Ponse, C. Verhoef, S.F.M. Vlijmen (eds.)	De proceedings: ACP'95, p.
95/15	P. Niebert and W. Penczek	On the Connection of Partial Order Logics and Partial Order Reduction Methods, p. 12.
95/16	D. Dams, O. Grumberg, R. Gerth	Abstract Interpretation of Reactive Systems: Preservation of CTL*, p. 27.
95/17	S. Mauw and E.A. van der Meulen	Specification of tools for Message Sequence Charts, p. 36.
95/18	F. Kamareddine and T. Laan	A Reflection on Russell's Ramified Types and Kripke's Hierarchy of Truths, p. 14.
95/19	J.C.M. Baeten and J.A. Bergstra	Discrete Time Process Algebra with Abstraction, p. 15.
95/20	F. van Raamsdonk and P. Severi	On Normalisation, p. 33.
95/21	A. van Deursen	Axiomatizing Early and Late Input by Variable Elimination, p. 44.
95/22	B. Arnold, A. v. Deursen, M. Res	An Algebraic Specification of a Language for Describing Financial Products, p. 11.
95/23	W.M.P. van der Aalst	Petri net based scheduling, p. 20.
95/24	F.P.M. Dignum, W.P.M. Nuijten, L.M.A. Janssen	Solving a Time Tabling Problem by Constraint Satisfaction, p. 14.
95/25	L. Feijs	Synchronous Sequence Charts In Action, p. 36.
95/26	W.M.P. van der Aalst	A Class of Petri nets for modeling and analyzing business processes, p. 24.
95/27	P.D.V. van der Stok, J. van der Wal	Proceedings of the Real-Time Database Workshop, p. 106.
95/28	W. Fokkink, C. Verhoef	A Conservative Look at term Deduction Systems with Variable Binding, p. 29.
95/29	H. Jurjus	On Nesting of a Nonmonotonic Conditional, p. 14
95/30	J. Hidders, C. Hoskens, J. Paredaens	The Formal Model of a Pattern Browsing Technique, p.24.
95/31	P. Kelb, D. Dams and R. Gerth	Practical Symbolic Model Checking of the full μ-calculus using Compositional Abstractions, p. 17.
95/32	W.M.P. van der Aalst	Handboek simulatie, p. 51.
95/33	J. Engelfriet and JJ. Vereijken	Context-Free Graph Grammars and Concatenation of Graphs, p. 35.
95/34	J. Zwanenburg	Record concatenation with intersection types, p. 46.
95/35	T. Basten and M. Voorhoeve	An algebraic semantics for hierarchical P/T Nets, p. 32.
96/01	M. Voorhoeve and T. Basten	Process Algebra with Autonomous Actions, p. 12.
96/02	P. de Bra and A. Aerts	Multi-User Publishing in the Web: DreSS, A Document Repository Service Station, p. 12 .
96/03	W.M.P. van der Aalst	Parallel Computation of Reachable Dead States in a Free-choice Petri Net, p. 26.
96/04	S. Mauw	Example specifications in phi-SDL.