# Inheritance of Behavior

Twan Basten[1] and Wil M.P. van der Aalst[2]

[1] Dept. of Electrical Engineering, Eindhoven University of Technology, The Netherlands
a.a.basten@tue.nl
[2] Dept. of Technology Management and Dept. of Computing Science
Eindhoven University of Technology, The Netherlands
w.m.p.v.d.aalst@tue.nl

**Abstract**

One of the key issues of object-oriented modeling and design 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, captured by their life cycles. This paper studies inheritance of behavior both in a simple process-algebraic setting and in a Petri-net framework. Process algebra is chosen, because it concentrates on behavior, while abstracting from the internal states of processes. The result of the algebraic study is a clear conceptual understanding of inheritance of behavior. It can be expressed in terms of blocking and hiding method calls. The results in the algebraic framework inspire the development of the concept of inheritance of behavior in the Petri-net framework. The Petri-net formalism allows for a graphical representation of life cycles of objects with an explicit representation of object states. In the Petri-net framework, four inheritance rules are defined that can be used to construct life cycles of subclasses from the object life cycles of given (super-)classes. These inheritance rules can be used to structure a design process and they stimulate the reuse of life-cycle specifications. It turns out that the combination of blocking and hiding method calls captures a number of important operators for constructing life cycles of subclasses from life cycles of superclasses, namely choice, sequential composition, parallel composition, and iteration. A small case study validates our approach to inheritance of behavior.

**Key words:** object orientation – inheritance – process – object life cycle – process algebra – Petri nets – free-choice Petri nets – reuse

## Contents

# 1 Introduction

To date, a popular approach to the modular design of complex systems is the *object-oriented* approach. The modularity construct in any object-oriented design method is the *class* construct. A *class* describes a set of *objects* with a common structure and behavior. An *object* is an *instance* of a class. Classes may, for example, describe persons, cars, or production units. Objects of such classes are typically person $X$, car $Y$, or production unit $Z$. Each class and, hence, each object has a set of *attributes*. Attributes describe properties of objects. The *value* of these attributes determines the *state* of an object. In addition, each class has a set of *methods*. A *method* is an operation on an object. Methods may, for example, be used to read the value of an attribute, or to change the state of an object. Finally, a class contains a definition of the dynamic behavior of objects. That is, it specifies the order in which the methods of an object may be executed. Such a specification is called the *life cycle* of an object.

The Unified Modeling Language (UML) [69, 21, 58] has been accepted throughout the software industry as the standard object-oriented framework for specifying, constructing, visualizing, and documenting software-intensive systems. The development of UML began in late 1994 when Booch and Rumbaugh of Rational Software Corporation began their work on unifying the OOD [20] and OMT [68] methods. In the fall of 1995, Jacobson and his Objectory company joined Rational, incorporating the OOSE method [40] in the unification effort. The given references to UML and the other methods are a good starting point for the reader interested in an introduction to object-oriented design including a detailed explanation of all the abovementioned concepts.

One of the main goals of object-oriented design is the *reuse* of system components. A key concept to achieve this goal is the concept of *inheritance*. The inheritance mechanism allows the designer to specify a class, the *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 concept of inheritance is usually well defined for the *static structure* of a class consisting of the set of methods and the attributes. However, as mentioned, a class contains also a definition of the dynamic behavior of an object, the object life cycle. The current version of UML, Version 1.3 [58], supports nine types of diagrams: class diagrams, object diagrams, use case diagrams, sequence diagrams, collaboration diagrams, statechart diagrams, activity diagrams, component diagrams, and deployment diagrams. Four of these types of diagrams, namely sequence diagrams, collaboration diagrams, statechart diagrams, and activity diagrams capture (a part of) the behavior of the modeled system. Sequence diagrams and collaboration diagrams only model examples of interactions between objects. Activity diagrams emphasize the flow of control from activity to activity, whereas statechart diagrams emphasize the potential states and the transitions among those states. Both statechart diagrams and activity diagrams can be used to specify the dynamics of various aspects of a system ranging from the life cycle of a single object to complex interactions between societies of objects. Activity diagrams typically address the dynamics of the whole system including interactions between objects. Statechart diagrams are typically used to model an object's life cycle. Therefore, we focus on statechart diagrams. Statechart diagrams are based on a technique invented by Harel [38].

Looking at the informal definition of inheritance in UML, it states the following: "The mechanism by which more specific elements incorporate structure and behavior defined by more general elements." [69, Page 299]. However, only the class diagrams, describing purely structural aspects of a class, are equipped with a concrete notion of inheritance. It is implicitly assumed that the behavior of the objects of a subclass is an extension of the behavior of the objects of its superclass.

Consider two classes $Unit_1$ and $Unit_2$ modeling production units. Both classes have the same methods, namely $pmat_1$ and $pmat_2$, modeling two processing operations on input material. Objects of class $Unit_1$ first execute $pmat_1$ and then $pmat_2$. Objects of class $Unit_2$ perform either $pmat_1$ or $pmat_2$ but not both. Should one of the classes $Unit_1$ or $Unit_2$ be a subclass of the other one? Although the two classes have the same set of methods, their behavior is clearly different. Hence, the answer to the above question should be negative.

Therefore, in this paper, we study several formalizations of what it means for an object life cycle to extend the behavior prescribed by another object life cycle. Combining the usual definition of inheritance of methods and attributes with a definition of inheritance of behavior yields a complete formal definition of inheritance, thus, stimulating the reuse of life-cycle specifications during the design process. However, it is beyond the scope of this paper to develop a complete object-oriented method including a notion of inheritance of life cycles. Instead, this paper focuses on the fundamentals of inheritance of behavior. The integration of the results in a full fledged object-oriented design method is left for future work.

Let us consider the question of when one object life cycle extends another object life cycle in some more detail. In other words, the question is as follows: *When is one object life cycle a subclass of another life cycle?* There seem to be many possible answers to this question. 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. Usually, a method operating on some object interacts with the environment of the object. Such a method is called an *external* method. The order in which external methods may be executed determines the *external behavior* of an object. The external behavior of an object determines how the environment of the object observes the object. Sometimes, a method can only be executed by the object itself and has only internal effects. Such a method does not contribute to the external behavior of an object. The basis of this paper is formed by two possible answers to the above question. Each of these answers yields a fundamental form of inheritance of behavior.

Assume that $p$ and $q$ are two object life cycles. The first answer is as follows.

> *If it is not possible to distinguish the external behavior of $p$ and $q$ when only methods of $p$ that are also present in $q$ are executed, then $p$ is a subclass of $q$.*

Intuitively, this basic form of inheritance conforms to *blocking* calls to methods new in $p$. In the remainder, life cycle $p$ is said to inherit the *protocol* of $q$; the resulting fundamental form of inheritance is referred to as *protocol inheritance*.

The second answer to the above question is as follows.

> *If it is not possible to distinguish the external behavior of $p$ and $q$ when arbitrary methods of $p$ are executed, but when only the effects of methods that are also present in $q$ are considered, then $p$ is a subclass of $q$.*

This second basic form of inheritance of behavior conforms to *hiding* the effect of methods new in $p$. Life cycle $p$ inherits the *projection* of the life cycle of $p$ onto the methods of $q$; the resulting form of inheritance is called *projection inheritance*.

As mentioned, UML uses statechart diagrams to specify object life cycles. Although the graphical nature and the explicit representation of states are essential to the success and usefulness of UML, particularly the latter impedes a clear understanding of inheritance of behavior. For studying inheritance of life cycles, the most important aspects of a life cycle are the *state changes* and not the states themselves. Therefore, the first part of this paper studies the problem of inheritance of behavior in a *process-algebraic setting*. In general, a process-algebraic theory does not have an explicit representation of process states. In addition, it has been mentioned that *blocking* and *hiding* method calls play a fundamental role in inheritance of life cycles. In process-algebraic terms, the former corresponds to *encapsulation* and the latter to *abstraction*. Encapsulation and abstraction are well understood in the context of process algebra. Note that the terms "abstraction" and "encapsulation" in process algebra have a different meaning than the same terms in object-oriented design. In this paper, they always refer to the process-algebraic concepts. The second part of this paper translates the results developed in the algebraic framework to *Petri nets*. Petri nets have a solid theoretical basis and, due to their explicit representation of process states and their graphical nature, they are close to the statechart diagrams (as well as the activity diagrams) used in UML. The translation of the fundamentals developed in Section 4 to Petri nets is illustrative for translations to other graphical, state-based formalisms such as statecharts. To validate the approach to inheritance of behavior chosen in this paper, the final part of this paper discusses a case study. It describes the use of inheritance in the design of a groupware editor.

The remainder of this paper is organized as follows. Section 2 introduces the basic semantic framework used throughout this paper. The framework of labeled transition systems is used to formalize the notions of a process and equivalence of processes. The process framework is used in the remainder of the paper to precisely define the behavior of process-algebraic terms and Petri-net models. Sections 3 and 5 provide introductions to process algebra and Petri nets, respectively. These sections contain (almost) no new material. They are included to provide a sound basis for the other sections of this paper and to make the paper self-contained. Readers already familiar with process algebra and/or Petri nets are advised to browse through these sections in order to get familiar with the exact frameworks and the notation that is being used. In Section 4, the concept of inheritance of behavior is developed in a process-algebraic setting. In Sections 6 and 7, the results of the algebraic framework are translated to Petri nets. Section 6 contains the basic definitions and results, whereas Section 7 focuses on a set of transformation rules that can be used to construct subclasses from given object life cycles. In Section 8, a small case study is described. It shows how the transformation rules of Section 7 can be used to structure an object-oriented design process. Finally, Section 9 discusses some conclusions, related work, and open problems.

# 2 Process Theory

## 2.1 Processes

A very natural and elementary way to formalize a behavioral description is by means of a *labeled transition system*. A labeled transition system is a set of *states* plus a *transition relation* on states. Each transition is labeled with an *action*. In the context of this paper, an action typically corresponds to a method invocation. For our purposes, the details of methods are not important. Therefore, actions are assumed to be atomic entities without internal structure. The set of states in a labeled transition system is an abstraction of all possible states of an object. The transition relation describes the change in the state of an object when some method, the label of the transition, is executed.

The basic notion in the framework of labeled transition systems used in this paper is the so-called *process space*. A process space describes a *set of processes*. A process space is a labeled transition system as described above extended with a termination predicate on states. Each state in a process space can be interpreted as the initial state of a process. A *process* is a labeled transition system extended with a termination predicate and a distinguished initial state. The termination predicate of a process defines in what states the process *can terminate successfully*. If a process is in a state where it cannot perform any actions or terminate successfully, then it is said to be in a *deadlock*. The possibility to distinguish between successful termination and deadlock is useful in the remainder. Process spaces form the basic semantic framework in this paper. A predicate on the elements of some set is represented as a subset of this set: It holds for elements in the subset and it does not hold for elements outside the subset.

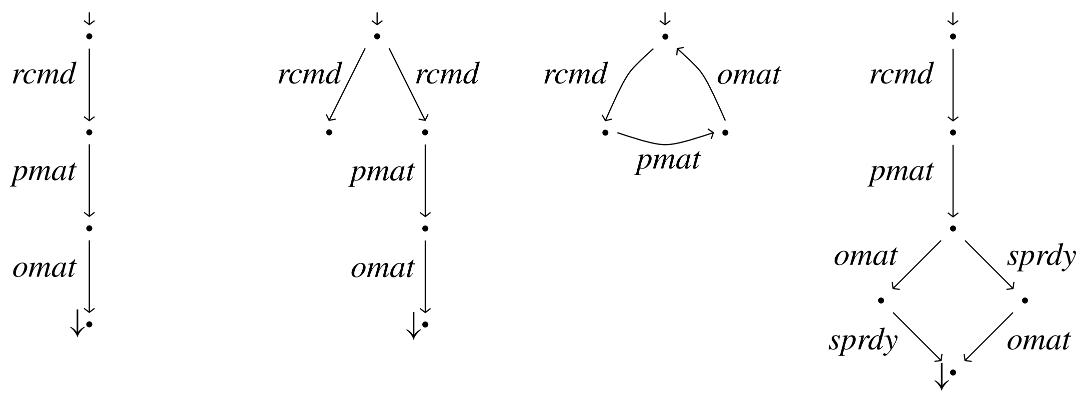
**Definition 2.1. (Process space)** A *process space* is a quadruple $(\mathcal{P}, \mathcal{A}, \longrightarrow, \downarrow)$, where $\mathcal{P}$ is a set of states, $\mathcal{A}$ is a set of actions, $\_ \overset{-}{\longrightarrow} \_ \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$ is a ternary transition relation, and $\downarrow \_ \subseteq \mathcal{P}$ is a termination predicate.

Let $(\mathcal{P}, \mathcal{A}, \longrightarrow, \downarrow)$ be some process space. Each state $p$ in $\mathcal{P}$ uniquely determines a process that consists of all states reachable from $p$.

**Definition 2.2. (Reachability)** The *reachability relation* $\_ \overset{*}{\Longrightarrow} \_ \subseteq \mathcal{P} \times \mathcal{P}$ is defined as the smallest relation satisfying, for any $p, p', p'' \in \mathcal{P}$ and $\alpha \in \mathcal{A}$,
$p \overset{*}{\Longrightarrow} p$ and
$(p \overset{*}{\Longrightarrow} p' \wedge p' \overset{\alpha}{\longrightarrow} p'') \;\Rightarrow\; p \overset{*}{\Longrightarrow} p''$.
State $p'$ is said to be *reachable* from state $p$ if and only if $p \overset{*}{\Longrightarrow} p'$. The set of all states reachable from $p$ is denoted $p*$.

**Definition 2.3. (Process)** Let $p$ be a state in $\mathcal{P}$. The *process* defined by $p$ is the 5-tuple $(p, p*, \mathcal{A}, \longrightarrow \cap (p* \times \mathcal{A} \times p*), \downarrow \cap\, p*)$. State $p$ is the *initial state* of the process.

In the remainder, processes are identified with their initial states. Sometimes, it is intuitive to think about elements of $\mathcal{P}$ as states, whereas sometimes it is more natural to see them as processes.




$(a)$ $(b)$ $(c)$ $(d)$

Figure 2.4: Some simple examples of processes.

**Example 2.5.** Figure 2.4 shows some examples of processes modeling variants of a very simple production unit. Process states are depicted as dots. The initial state of a process is marked with a small incoming arrow. The production unit in Figure 2.4($a$) is a simple sequential process that starts with receiving a command by performing action *rcmd*. After it has received a command, it processes some input material (*pmat*), outputs the processed material (*omat*), and then terminates. The unit in Figure 2.4($b$) is the same process, except that, upon receiving a command, it may deadlock. A reason could be that the command is not understood. The process in Figure 2.4($c$) is a variant that iterates the behavior of the production unit in ($a$). It does not have an option to terminate. The production unit of Figure 2.4($d$) is again a non-iterative process. The interesting aspect of this process is that it exhibits concurrency (assuming a total-order view on concurrency). In parallel to its output action, it sends a ready-processing signal (*sprdy*) to, for example, an operator. This can be useful for an operator who must pick up the processed material if processing takes a large, but variable amount of time. In the context of process spaces, concurrency typically means that actions may be executed in any order.

As mentioned, process spaces are used to provide the process-algebraic and Petri-net formalisms in the remainder with a formal operational semantics. However, it might occur that the behavior of two different processes is very similar, even to such an extent that the processes may be considered equivalent. The next subsection formalizes two so-called *semantic equivalence relations*. A semantic equivalence precisely defines when two processes are equivalent.

## 2.2 Equivalence of processes

As explained in the introduction to this paper, it is important to be able to distinguish between external and internal behavior of processes. Internal behavior is often also referred to as *silent* behavior. It is straightforward to extend the framework of the previous subsection in such a way that processes may exhibit internal behavior. It suffices to introduce so-called *silent actions*. Silent actions are actions that cannot be observed. Usually, silent actions are denoted with the action label $\tau$. A single symbol is sufficient, since all internal actions are equal in the sense that they do not have any visible effects.

In case one is interested in external behavior only, processes with the same external behavior but with different internal behavior should be equal. *Branching bisimilarity* is a well-known semantic equivalence that satisfies this requirement. Branching bisimilarity was first introduced in [35]. The definition given in this subsection is slightly different from the original definition. In fact, it is the definition of *semi*-branching bisimilarity, which was first defined in [74, Chapter 1]. It can be shown that the two notions are equivalent in the sense that they define the same equivalence relation on processes [36, 12]. The reason for using the alternative definition is that it is more concise and more intuitive than the original definition. It also yields shorter proofs. A comparison of the two definitions can be found in [12].

Branching bisimilarity is a slightly finer equivalence than the well-known observation equivalence [53, 54]. That is, it distinguishes more processes than observation equivalence (see [34]). A comparison of branching bisimilarity, observation equivalence, and a few other equivalences on processes with silent behavior can be found in [36].

Let $(\mathcal{P}, \mathcal{A}, \longrightarrow, \downarrow)$ be a process space as defined in Definition 2.1. The set of actions $\mathcal{A}$ is defined as $A \cup \{\tau\}$, where $A$ is some set of externally observable actions. To define branching bisimilarity, two auxiliary definitions are needed: *i*) a relation expressing that a process can evolve into another process by executing a sequence of zero or more $\tau$ actions; *ii*) a predicate expressing that a process can terminate by performing zero or more $\tau$ actions.

**Definition 2.6.** Relation $\_ \Longrightarrow \_ \subseteq \mathcal{P} \times \mathcal{P}$ is defined as the smallest relation satisfying, for any $p, p', p'' \in \mathcal{P}$,
$p \Longrightarrow p$ and
$(p \Longrightarrow p' \wedge p' \xrightarrow{\tau} p'') \Rightarrow p \Longrightarrow p''$.

**Definition 2.7.** Predicate $\Downarrow \_ \subseteq \mathcal{P}$ is defined as the smallest set of processes satisfying, for any $p, p' \in \mathcal{P}$,
$\downarrow p \Rightarrow \Downarrow p$ and
$(\Downarrow p \wedge p' \xrightarrow{\tau} p) \;\Rightarrow\; \Downarrow p'$.

Note that it is also possible to define the predicate $\Downarrow$ in terms of $\Longrightarrow$.

Let, for any processes $p, p' \in \mathcal{P}$ and action $\alpha \in \mathcal{A}$, $p \xrightarrow{(\alpha)} p'$ be an abbreviation of the predicate $(\alpha = \tau \wedge p = p') \vee p \xrightarrow{\alpha} p'$. Thus, $p \xrightarrow{(\tau)} p'$ means that zero $\tau$ actions are performed, when the first disjunct of the predicate is satisfied, or that one $\tau$ action is performed, when the second disjunct is satisfied. For any external action $a \in A$, the first disjunct of the predicate can never be satisfied. Hence, $p \xrightarrow{(a)} p'$ is simply equal to $p \xrightarrow{a} p'$, meaning that a single $a$ action is performed.

**Definition 2.8. ((Rooted) branching bisimilarity)** A binary relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ is called a *branching bisimulation* if and only if, for any $p, p', q, q' \in \mathcal{P}$ and $\alpha \in \mathcal{A}$,

*i)* $p\mathcal{R}q \wedge p \xrightarrow{\alpha} p' \Rightarrow$
$(\exists\, q', q'' : q', q'' \in \mathcal{P} : q \Longrightarrow q'' \xrightarrow{(\alpha)} q' \wedge p\mathcal{R}q'' \wedge p'\mathcal{R}q')$,

*ii)* $p\mathcal{R}q \wedge q \xrightarrow{\alpha} q' \Rightarrow$
$(\exists\, p', p'' : p', p'' \in \mathcal{P} : p \Longrightarrow p'' \xrightarrow{(\alpha)} p' \wedge p''\mathcal{R}q \wedge p'\mathcal{R}q')$, and

*iii)* $p\mathcal{R}q \Rightarrow (\downarrow p \Rightarrow \Downarrow q \wedge \downarrow q \Rightarrow \Downarrow p)$.

Two processes are called *branching bisimilar*, denoted $p \sim_b q$, if and only if there exists a branching bisimulation $\mathcal{R}$ such that $p\mathcal{R}q$.

A branching bisimulation $\mathcal{R}$ is called a *rooted* branching bisimulation between $p$ and $q$ in $\mathcal{P}$ if and only if $p\mathcal{R}q$ and, for any $p', q' \in \mathcal{P}$ and $\alpha \in \mathcal{A}$,

*iv)* $p \xrightarrow{\alpha} p' \Rightarrow (\exists\, q' : q' \in \mathcal{P} : q \xrightarrow{\alpha} q' \wedge p'\mathcal{R}q')$,

*v)* $q \xrightarrow{\alpha} q' \Rightarrow (\exists\, p' : p' \in \mathcal{P} : p \xrightarrow{\alpha} p' \wedge p'\mathcal{R}q')$, and

*vi)* $\downarrow p \Leftrightarrow \downarrow q$.

Two processes $p$ and $q$ are called *rooted branching bisimilar*, denoted $p \sim_{rb} q$, if and only if there exists a rooted branching bisimulation between $p$ and $q$.

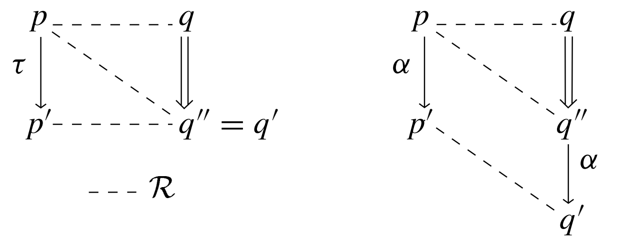



Figure 2.9: The essence of a branching bisimulation.

Figure 2.9 shows the essence of a branching bisimulation. A process must be able to simulate any action of an equivalent process after performing any number of silent actions, except for a silent action which it may or may not simulate. The third property in Definition 2.8 guarantees that related processes always have the same termination options. The *root condition*, introduced in the last three requirements of the definition, states that the *initial* actions of two rooted branching bisimilar processes must be the same. The root condition is needed, because branching bisimilarity is not a congruence for the process-algebraic choice operator which is introduced in Section 3. Rooted branching bisimilarity, on the other hand, is a congruence for all the process-algebraic operators used in this paper. As explained in Section 3, any semantic equivalence used in a process-algebraic context must have the congruence property for all the algebraic operators. Below, it is shown that branching bisimilarity and rooted branching bisimilarity are equivalence relations. However, before giving these results, a few examples are given to illustrate both equivalences.

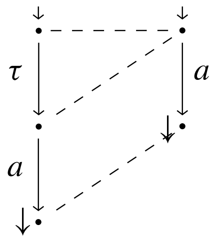



Figure 2.10: Two branching bisimilar processes that are not rooted branching bisimilar.

**Example 2.11.** Figure 2.10 shows two processes that are branching bisimilar but not *rooted* branching bisimilar. The problem is caused by the initial silent action of the left process, which cannot be simulated by the right one. This example shows the essential difference between branching bisimilarity and rooted branching bisimilarity. Branching bisimilarity allows to remove initial silent actions, whereas rooted branching bisimilarity does not.

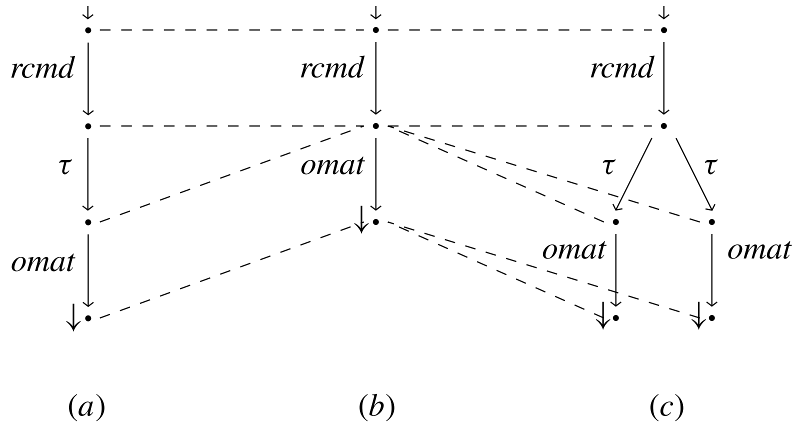



Figure 2.12: Some examples of branching bisimilar processes.

**Example 2.13.** Figure 2.12 shows three examples of branching bisimilar processes. The processes in $(a)$ and $(c)$ could, for example, be the result of abstraction. Consider a simple production unit with a single processing step; process $(a)$ represents this production unit after abstracting away the processing step. Process $(c)$ models a slightly more complex production unit which has a choice of two processing actions; both processing actions are hidden. The reason for the abstractions could be that one is interested in the input/output behavior of the production units. In both cases, the input/output behavior should be the receipt of a command followed by the output of processed material. Figure 2.12 shows that this is indeed the case. It is not difficult to verify that the two relations depicted by the dashed lines are indeed branching bisimulations. Note that they also satisfy the root condition. Hence, the three processes are not only branching bisimilar, but also rooted branching bisimilar.

**Theorem 2.14.** *Branching bisimilarity, $\sim_b$, and rooted branching bisimilarity, $\sim_{rb}$, are equivalence relations.*

**Proof.** It must be shown that branching bisimilarity and rooted branching bisimilarity are reflexive, symmetric, and transitive. Reflexivity follows from the fact that the identity relation on $\mathcal{P} \times \mathcal{P}$ is both a branching bisimulation and a rooted branching bisimulation relating an arbitrary process to itself. Symmetry follows easily from the symmetry in Definition 2.8 ((Rooted) branching bisimilarity). Finally, transitivity follows from the fact that the relation composition of two (rooted) branching bisimulations is again a (rooted) branching bisimulation. The details of the proof of this fact are tedious but straightforward; they can be found in [13, Section 2.2.3]. In [12], it is shown in detail that branching bisimilarity is an equivalence relation in a context without distinction between successful termination and deadlock. $\square$

# 3 Process Algebra

The goal of this section is to introduce a simple process-algebraic theory in the style of the Algebra of Communicating Processes (ACP). The theory ACP originates from [17]. Good introductions to ACP-style process algebra can be found in [10, 11, 33]. Other well-known process-algebraic theories are CCS [53, 54] and CSP [39]. For a detailed comparison of ACP, CCS, and CSP, the reader is referred to [11, Chapter 8].

## 3.1 Equational theory

Any ACP-style process-algebraic theory is essentially an *equational theory*. An equational theory consists of a *signature* and a set of *axioms*. The signature defines the *sorts* of the theory, a set of *variables* for each sort, and the *functions* of the theory. Functions and variables can be used to construct *terms*. Terms not containing any variables are called *closed* terms. The axioms of the theory determine which terms are equal. A process-algebraic theory usually has only a single sort; terms of this sort represent processes. A 0-ary function is often called a *constant*; other functions are often called *operators*.

The signature and axioms of the equational theory $(\mathrm{PA}_\delta^\tau + \mathrm{RN})(\boldsymbol{A})$, which is an abbreviation for Process Algebra with inaction, silent actions, and renaming, are given in Table 3.1. The theory is parameterized by a set of constants $\boldsymbol{A}$, which is a set of actions. It is assumed that $\boldsymbol{A}$ is defined as $A \cup \{\tau\}$, where $A$ is some set of external actions. The first part of Table 3.1 lists the sorts in the signature of $(\mathrm{PA}_\delta^\tau + \mathrm{RN})(\boldsymbol{A})$; the second part defines the constants and the operators in the signature. The third entry of Table 3.1 gives the variables and lists the axioms of the equational theory. Note that new variables may be introduced any time when necessary. An informal explanation of the operators and the axioms is given below.

As mentioned, $\boldsymbol{A}$ is a set of actions. Terms of sort $\boldsymbol{P}$ represent processes. Each action is a process, namely the process that can only execute the action and then terminates successfully.

The two basic operators of $(\mathrm{PA}_\delta^\tau + \mathrm{RN})(\boldsymbol{A})$ are $+$ and $\cdot$, denoting alternative composition or choice and sequential composition, respectively. These two operators are elementary in describing the behavior of sequential processes. Sequential composition binds stronger than choice. Choice and sequential composition are axiomatized by Axioms $A1$ through $A5$. Most of these axioms are self-explanatory. Only Axiom $A4$ might need some explanation. It states the *right distributivity* of sequential composition over choice. The converse, left distributivity, is not an axiom of the theory. As a result, processes with different moments of choice are distinguished.

The constant $\delta$ stands for *inaction*, often also called *deadlock*. However, the former name is best suited, as follows from Axiom $A6$. It says that a process which can choose between some behavior $x$ and doing nothing is equivalent to the process that has no choice and can only do $x$. Hence, in the context of a choice, $\delta$ is not a true deadlock. Axiom $A7$ shows that $\delta$ is a deadlock in the context of a sequential composition.

Axioms $M1$ through $M4$ axiomatize the behavior of concurrent processes. Constant $a$ ranges over $\boldsymbol{A} \cup \{\delta\}$. Thus, an axiom such as $M2$ containing the constant $a$ is actually an axiom *scheme*. That is, the

$$
\begin{array}{l}
\text{— } (\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A}) \text{ ——} \\
\boldsymbol{P}; \\
\hline
\boldsymbol{A} \subseteq \boldsymbol{P}, \quad \delta : \boldsymbol{P}, \quad \_+\_, \_\cdot\_, \_\parallel\_, \_\lfloor\!\lfloor\_ : \boldsymbol{P} \times \boldsymbol{P} \to \boldsymbol{P}, \quad \partial_H, \tau_I : \boldsymbol{P} \to \boldsymbol{P}; \\
\hline
x, y, z : \boldsymbol{P};
\end{array}
$$

| | | | |
|---|---|---|---|
| $x + y = y + x$ | $A1$ | $x \parallel y = x \lfloor\!\lfloor y + y \lfloor\!\lfloor x$ | $M1$ |
| $(x + y) + z = x + (y + z)$ | $A2$ | $a \lfloor\!\lfloor x = a \cdot x$ | $M2$ |
| $x + x = x$ | $A3$ | $a \cdot x \lfloor\!\lfloor y = a \cdot (x \parallel y)$ | $M3$ |
| $(x + y) \cdot z = x \cdot z + y \cdot z$ | $A4$ | $(x + y) \lfloor\!\lfloor z = x \lfloor\!\lfloor z + y \lfloor\!\lfloor z$ | $M4$ |
| $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ | $A5$ | | |
| $x + \delta = x$ | $A6$ | $x \cdot \tau = x$ | $B1$ |
| $\delta \cdot x = \delta$ | $A7$ | $x \cdot (\tau \cdot (y + z) + y) = x \cdot (y + z)$ | $B2$ |
| $a \notin H \Rightarrow \partial_H(a) = a$ | $D1$ | $a \notin I \Rightarrow \tau_I(a) = a$ | $TI1$ |
| $a \in H \Rightarrow \partial_H(a) = \delta$ | $D2$ | $a \in I \Rightarrow \tau_I(a) = \tau$ | $TI2$ |
| $\partial_H(x + y) = \partial_H(x) + \partial_H(y)$ | $D3$ | $\tau_I(x + y) = \tau_I(x) + \tau_I(y)$ | $TI3$ |
| $\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$ | $D4$ | $\tau_I(x \cdot y) = \tau_I(x) \cdot \tau_I(y)$ | $TI4$ |

Table 3.1: The equational theory $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A})$.

equational theory $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A})$ contains one axiom for each possible constant $a \in \boldsymbol{A} \cup \{\delta\}$. The parallel-composition operator $\parallel$, often called the *merge* operator, denotes the parallel execution of its operands. It is axiomatized using an auxiliary operator, namely $\lfloor\!\lfloor$, called the *left* merge. The left merge has the same meaning as the merge except that the left process must perform the first action.

Axioms $B1$ and $B2$ are the basis for an axiomatization of rooted branching bisimilarity (see [36]). Together, $B1$ and $B2$ state that it is allowed to remove a silent action provided that it does not enforce a choice.

Finally, the equational theory contains a so-called encapsulation operator $\partial_H$ for each $H \subseteq A$ and an abstraction operator $\tau_I$ for each $I \subseteq A$. (Note that $H$ and $I$ can only contain observable actions.) The axiom schemes for the encapsulation and abstraction operators are very similar. Again, constant $a$ ranges over $\boldsymbol{A} \cup \{\delta\}$. The encapsulation and abstraction operators belong to the general class of algebraic renaming operators. The encapsulation operator $\partial_H$ simply renames occurrences of actions in $H$ in a process term to the inaction constant $\delta$; the abstraction operator $\tau_I$ renames actions in $I$ to the silent action $\tau$.

An equational theory such as $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A})$ of Table 3.1 provides the basis for reasoning about processes. The set of axioms of an equational theory defines an equivalence relation on process terms, called *derivability*. For any process terms $x$ and $y$ in some given equational theory $X$, $X \vdash x = y$ denotes that $x = y$ can be derived from the axioms of $X$. Derivability in an equational theory is defined as follows. First, the axioms themselves can be derived from the axioms of the theory. Second, since derivability is an equivalence relation, it is reflexive, symmetric, and transitive. Third, if an equation is derivable from the axioms, then also any equation obtained by substituting terms for variables in this equation is derivable. Finally, any equation obtained by replacing a term in an arbitrary context by another derivably equivalent term is also derivable from the theory. The axioms of an equational theory must be chosen in such a way that derivability defines a meaningful equivalence relation on processes. In the next subsection, it is explained that in the case of the theory $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A})$ derivability corresponds to rooted branching bisimilarity.

**Example 3.2.** Assume that the set of actions $\boldsymbol{A}$ contains the actions *rcmd*, $pmat_1$, $pmat_2$, and *omat*. It can be shown that $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A}) \vdash rcmd \cdot ((pmat_1 + pmat_2) \cdot omat) = rcmd \cdot (pmat_1 \cdot omat + pmat_2 \cdot omat)$.

The first step is to substitute $pmat_1$, $pmat_2$, and $omat$ for the variables $x$, $y$, and $z$ in Axiom $A4$, which shows that $(\mathrm{PA}_\delta^\tau + \mathrm{RN})(\boldsymbol{A}) \vdash t_1 = t_2$, where $t_1$ is the term $(pmat_1 + pmat_2) \cdot omat$ and $t_2$ denotes the term $pmat_1 \cdot omat + pmat_2 \cdot omat$. The second and final step consists of an application of the context rule explained above: Replacing term $t_1$ in $rcmd \cdot t_1$ with the equivalent term $t_2$ yields the desired result.

The intuitive meaning of the operators, the axioms, and the induced equivalence relation given above can be formalized by giving a semantics in terms of the framework introduced in the previous section. However, before going into more details about the semantics for $(\mathrm{PA}_\delta^\tau + \mathrm{RN})(\boldsymbol{A})$, a few more aspects of the theory $(\mathrm{PA}_\delta^\tau + \mathrm{RN})(\boldsymbol{A})$ itself are explained.

Although the theory $(\mathrm{PA}_\delta^\tau + \mathrm{RN})(\boldsymbol{A})$ is not very complex, it already contains operators and axioms for reasoning about sequential as well as parallel processes. In addition, it contains constants and axioms for reasoning about deadlocks and silent actions. It is possible to define equational theories for studying any of these aspects in isolation or in arbitrary combinations. For example, the equational theory $\mathrm{BPA}(\boldsymbol{A})$, where BPA is an abbreviation for Basic Process Algebra, consists of the action constants in $\boldsymbol{A}$, the choice and sequential-composition operators, and Axioms $A1$ through $A5$. It is a very simple theory for reasoning about sequential processes. Adding the inaction constant $\delta$ to the signature of $\mathrm{BPA}(\boldsymbol{A})$ and extending the set of axioms with Axioms $A6$ and $A7$ yields the equational theory $\mathrm{BPA}_\delta(\boldsymbol{A})$. A theory that is suitable for reasoning about sequential and parallel processes with internal behavior is the theory $\mathrm{PA}^\tau(\boldsymbol{A})$, for Process Algebra with silent actions. The signature of this theory consists of the action constants, the silent-action constant $\tau$, and the choice, sequential-composition, merge, and left-merge operators; the set of axioms consists of Axioms $A1$ through $A5$, $M1$ through $M4$, and $B1$ and $B2$. The theories $\mathrm{BPA}(\boldsymbol{A})$, $\mathrm{BPA}_\delta(\boldsymbol{A})$, and $\mathrm{PA}^\tau(\boldsymbol{A})$ play a role in the remainder of this paper. Other combinations of the abovementioned aspects are, of course, also possible.

The main purpose of a theory as $(\mathrm{PA}_\delta^\tau + \mathrm{RN})(\boldsymbol{A})$ is to reason about processes in an equational way. The goal is to prove a desired equality on processes by applying simple term-rewriting techniques. As a consequence, a very useful property of an equational theory is when its (closed) terms can be reduced to (unique) normal forms. If such normal forms exist, an equational proof becomes very simple. The equality of two process terms can be shown by reducing them to their normal forms. If the normal forms are equal, then, obviously, also the two processes are equal. By considering all axioms in Table 3.1 except for Axiom $A1$ as rewrite rules from left to right, it can be shown that closed $(\mathrm{PA}_\delta^\tau + \mathrm{RN})(\boldsymbol{A})$ terms have normal forms. For the purpose of this paper, it is sufficient to know that the normal forms of closed $(\mathrm{PA}_\delta^\tau + \mathrm{RN})(\boldsymbol{A})$ terms are contained in a very specific class of terms, called *basic terms*. In general ACP-style process algebra, the class of basic terms depends on whether or not the inaction constant $\delta$ is contained in the signature of the equational theory. The set of basic terms corresponding to a theory without the inaction constant can be defined as a subset of the set of closed BPA terms, whereas the basic terms for a theory with the inaction constant form a subset of closed $\mathrm{BPA}_\delta$ terms.

**Definition 3.3. (Basic terms)** The set of basic $\mathrm{BPA}(\boldsymbol{A})$ terms, denoted $\mathcal{B}(\mathrm{BPA}(\boldsymbol{A}))$, is inductively defined as follows. The set of actions $\boldsymbol{A}$ is contained in $\mathcal{B}(\mathrm{BPA}(\boldsymbol{A}))$. Furthermore, for any $a \in \boldsymbol{A}$ and basic terms $s, t \in \mathcal{B}(\mathrm{BPA}(\boldsymbol{A}))$, also $a \cdot t$ and $s + t$ are elements of $\mathcal{B}(\mathrm{BPA}(\boldsymbol{A}))$.

The set of basic $\mathrm{BPA}_\delta(\boldsymbol{A})$ terms, denoted $\mathcal{B}(\mathrm{BPA}_\delta(\boldsymbol{A}))$, is defined in a similar way: $\boldsymbol{A} \cup \{\delta\} \subseteq \mathcal{B}(\mathrm{BPA}_\delta(\boldsymbol{A}))$ and, for any $a \in \boldsymbol{A}$ and $s, t \in \mathcal{B}(\mathrm{BPA}_\delta(\boldsymbol{A}))$, $a \cdot t \in \mathcal{B}(\mathrm{BPA}_\delta(\boldsymbol{A}))$ and $s + t \in \mathcal{B}(\mathrm{BPA}_\delta(\boldsymbol{A}))$.

Note that not all basic terms are normal forms when using the axioms in Table 3.1, with the exception of Axiom $A1$, as rewrite rules from left to right. For example, basic ($\mathrm{BPA}(\boldsymbol{A})$ or $\mathrm{BPA}_\delta(\boldsymbol{A})$) term $a + a$, where $a$ is some action in $\boldsymbol{A}$, can be reduced by means of Axiom $A3$ and is, therefore, not a normal form.

The following property formalizes the claims made above concerning normal forms of equational theories for the theories $(\mathrm{PA}_\delta^\tau + \mathrm{RN})(\boldsymbol{A})$ and $\mathrm{PA}^\tau(\boldsymbol{A})$. For any equational theory $X$, the set of all closed terms over the signature of $X$ is denoted $\mathcal{C}(X)$

**Property 3.4. (Elimination)**

*i) For any closed term* $p \in \mathcal{C}(\mathrm{PA}^\tau(\boldsymbol{A}))$, *there is a basic term* $t \in \mathcal{B}(\mathrm{BPA}(\boldsymbol{A}))$, *such that* $\mathrm{PA}^\tau(\boldsymbol{A}) \vdash p = t$.

*ii) For any closed term* $p \in \mathcal{C}((\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A}))$, *there is a basic term* $t \in \mathcal{B}(\mathrm{BPA}_\delta(\boldsymbol{A}))$, *such that* $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A}) \vdash p = t$.

**Proof.** Using the standard techniques of [10], the proof is straightforward. □

The above results are called *elimination* properties, because they show that the general sequential composition $\cdot$ as defined in Table 3.1 as well as all other operators different from the choice operator can be eliminated from any closed term, yielding a term containing only constants, choices, and so-called *prefix compositions*. A prefix composition is a sequential composition whose left operand is a single action (see Definition 3.3 (Basic terms)).

**Example 3.5.** Consider again Example 3.2. The simple derivation in this example shows that closed $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A})$ term $\mathit{rcmd}\cdot((\mathit{pmat}_1 + \mathit{pmat}_2)\cdot \mathit{omat})$, which is not a basic term, can be rewritten to the basic $\mathrm{BPA}_\delta(\boldsymbol{A})$ term $\mathit{rcmd} \cdot (\mathit{pmat}_1 \cdot \mathit{omat} + \mathit{pmat}_2 \cdot \mathit{omat})$.

Elimination properties are useful for the following reason. Assume we would like to prove a property for all closed terms of some given equational theory. If closed terms can be reduced to basic terms, it suffices to prove this property only for basic terms. Properties for basic terms can often be proven by means of *structural induction*. An example of a property with a proof that goes along these lines is Lemma 4.8 given in the next section.

Two equational theories play a central role in the next section, namely $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A})$ and $\mathrm{PA}^\tau(\boldsymbol{A})$. Recall that both the signature and the axioms of $\mathrm{PA}^\tau(\boldsymbol{A})$ consist of a subset of the signature and the axioms of theory $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A})$. This leads to the interesting question whether it is possible to derive any equalities between $\mathrm{PA}^\tau(\boldsymbol{A})$ terms from the axioms of $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A})$ that are not also derivable from the theory $\mathrm{PA}^\tau(\boldsymbol{A})$. The next result shows that this is not the case. It states that $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A})$ is a so-called *conservative* extension of $\mathrm{PA}^\tau(\boldsymbol{A})$.

**Property 3.6. (Conservative extension)** *For any closed terms* $p, q \in \mathcal{C}(\mathrm{PA}^\tau(\boldsymbol{A}))$,
$(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A}) \vdash p = q \Leftrightarrow \mathrm{PA}^\tau(\boldsymbol{A}) \vdash p = q$.

**Proof.** Using the techniques of [10], the proof is straightforward. Note that these techniques depend on the operational semantics of the equational theories. For the purpose of this proof, the semantics given in the next subsection can be used. The reason for presenting the conservativity result before the semantics is defined, is that the result itself is independent of the semantics. Furthermore, there are also proof techniques that do not use any specific semantics (see [11]). □

As a final remark, note that the equational framework introduced in this subsection does not include features to specify and analyze, for example, recursive processes or communicating processes. Recursion and communication are often present in algebraic theories, but they do not play a role in this paper. The interested reader is referred to, for example, [10, 11, 33].

## 3.2 Operational semantics

As before, assume that $\boldsymbol{A}$ is a set of action constants. The semantics of terms in an equational theory is formalized by defining a so-called *model* of the theory also called an *algebra* for the theory. Since the terms in a single-sorted equational theory such as $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A})$ are supposed to be interpreted as processes, a model of an ACP-style equational theory is also called a *process algebra*.

In general, a model of a single-sorted equational theory $X$ consists of a *domain* of elements plus a number of functions on that domain, called the *signature* of the model. A model $\mathcal{M}$ with domain $\mathcal{D}$ must satisfy the following properties. First, there must exist an interpretation of the functions in the signature of $X$ in terms of the functions in the signature of the model $\mathcal{M}$ that preserves the arity of functions. Second, any equation that is derivable from the axioms of the equational theory must be *valid* in the model, where validity is defined as follows. Let $t$ be a term in the equational theory $X$; let $\sigma$ be a mapping from variables in $t$ to elements from domain $\mathcal{D}$, called a variable substitution. The interpretation of $t$ in the model $\mathcal{M}$ under substitution $\sigma$, denoted $[\![t]\!]_\sigma$, is obtained by replacing all functions in $t$ by the corresponding functions in $\mathcal{M}$ and by replacing all variables in $t$ by elements of domain $\mathcal{D}$ according to $\sigma$. An equation $t_1 = t_2$ in $X$ is *valid* in model $\mathcal{M}$, denoted $\mathcal{M} \models t_1 = t_2$, if and only if, for *all* substitutions $\sigma$ for the variables in $t_1$ and $t_2$, $[\![t_1]\!]_\sigma =_\mathcal{D} [\![t_2]\!]_\sigma$, where $=_\mathcal{D}$ is the identity on domain $\mathcal{D}$. If $\mathcal{M}$ is a model of an equational theory $X$, it is also said that $X$ is a *sound axiomatization* of $\mathcal{M}$.

**Example 3.7.** Assume that $\mathcal{M}$ with domain $\mathcal{D}$ is a model of the equational theory $(\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})$ of Table 3.1. Recall that the functions in the signature of $(\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})$ consist of the inaction constant $\delta$, the action constants in $\boldsymbol{A}$, and the operators listed in Table 3.1. Assume that, for any function $f$ in the signature of $(\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})$, $\bar{f}$ denotes the corresponding function in the signature of $\mathcal{M}$. Consider the equation $a+\delta = a$, where $a$ is an action in $\boldsymbol{A}$. It follows from Axiom $A6$ that this equation is derivable from theory $(\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})$. Since $\mathcal{M}$ is a model of this theory, $a+\delta = a$ must be valid in $\mathcal{M}$, which means that the equality $\bar{a}\bar{+}\bar{\delta} =_\mathcal{D} \bar{a}$, where $=_\mathcal{D}$ is the identity on domain $\mathcal{D}$, must hold. In fact, Axiom $A6$ itself is derivable from the theory and must, therefore, be valid in $\mathcal{M}$. That is, for any $d \in \mathcal{D}$, the equality $d\bar{+}\bar{\delta} =_\mathcal{D} d$ must hold.

The basis for a model of the equational theory $(\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})$ is a process space, as defined in Definition 2.1. This means that a set of processes, a set of actions, a transition relation, and a termination predicate need to be defined.

Recall that $\mathcal{C}((\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A}))$ is the set of *closed* $(\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})$ terms. The set $\mathcal{C}((\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A}))$ forms the basis for the set of processes in the process space. Since the equational theory $(\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})$ has no means to express the process that can perform no actions, but can only terminate successfully, a special process $\surd$, pronounced "tick," is introduced. Thus, the set of processes in the abovementioned process space is the set $\mathcal{C}((\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})) \cup \{\surd\}$. The set $\boldsymbol{A}$ is the set of actions. The termination predicate is the singleton $\{\surd\}$. That is, process $\surd$ is the only process that can terminate successfully. The transition relation $\_ \xrightarrow{\_} \_ \subseteq (\mathcal{C}((\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})) \cup \{\surd\}) \times \boldsymbol{A} \times (\mathcal{C}((\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})) \cup \{\surd\})$ can now be defined as the smallest relation satisfying the derivation rules in Table 3.8. It is not difficult to verify that the transition relation conforms to the informal explanation of the operators given in the previous subsection.

The process space $(\mathcal{C}((\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})) \cup \{\surd\}, \boldsymbol{A}, \longrightarrow, \{\surd\})$ can be turned into a model $\mathcal{M}(\boldsymbol{A})$ of the equational theory $(\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})$ as follows.

Recall that rooted branching bisimilarity, as defined in Definition 2.8, is an equivalence relation on the set of processes $\mathcal{C}((\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})) \cup \{\surd\}$. Thus, it is possible to define equivalence classes of processes modulo rooted branching bisimilarity in the usual way: For any $p \in \mathcal{C}((\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})) \cup \{\surd\}$, the equivalence class of $p$ modulo rooted branching bisimilarity, denoted $[p]_{\sim_{rb}}$, is the set $\{q \in \mathcal{C}((\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})) \cup \{\surd\} \mid q \sim_{rb} p\}$. It follows from Definition 2.8 (Rooted branching bisimilarity) that the special element $[\surd]_{\sim_{rb}}$ only contains the process $\surd$. The *domain* of the model under construction is formed by the set of all the equivalence classes of closed terms modulo rooted branching bisimilarity.[1] The special element $[\surd]_{\sim_{rb}}$ is excluded from the domain of model $\mathcal{M}(\boldsymbol{A})$ for technical reasons: As mentioned, the equational theory $(\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})$ has no means to express the process that can only terminate successfully.

[1]In standard process-algebraic terminology, the elements in the domain of a model of some ACP-style equational theory are referred to as processes. However, Definition 2.3 defines a process as some kind of labeled transition system. The domain of model $\mathcal{M}(\boldsymbol{A})$ consists of equivalence classes of such labeled transition systems. In the literature on concurrency theory, the use of the term "process" for both equivalence classes of labeled transition systems and individual representatives of such equivalence classes is common practice and does not lead to confusion.

$a \in \boldsymbol{A}$; $H, I \subseteq A$; $p, p', q, q' \in \mathcal{C}((\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A}))$;

$$a \xrightarrow{a} \surd \qquad \frac{p \xrightarrow{a} p'}{p \cdot q \xrightarrow{a} p' \cdot q} \qquad \frac{p \xrightarrow{a} \surd}{p \cdot q \xrightarrow{a} q}$$

$$\frac{p \xrightarrow{a} p'}{p + q \xrightarrow{a} p'} \qquad \frac{q \xrightarrow{a} q'}{p + q \xrightarrow{a} q'} \qquad \frac{p \xrightarrow{a} \surd}{p + q \xrightarrow{a} \surd} \qquad \frac{q \xrightarrow{a} \surd}{p + q \xrightarrow{a} \surd}$$

$$\frac{p \xrightarrow{a} p'}{p \parallel q \xrightarrow{a} p' \parallel q} \qquad \frac{q \xrightarrow{a} q'}{p \parallel q \xrightarrow{a} p \parallel q'} \qquad \frac{p \xrightarrow{a} \surd}{p \parallel q \xrightarrow{a} q} \qquad \frac{q \xrightarrow{a} \surd}{p \parallel q \xrightarrow{a} p}$$

$$\frac{p \xrightarrow{a} p'}{p \mathbin{⌊\!⌊} q \xrightarrow{a} p' \parallel q} \qquad \frac{p \xrightarrow{a} \surd}{p \mathbin{⌊\!⌊} q \xrightarrow{a} q}$$

$$\frac{p \xrightarrow{a} p', \quad a \notin H}{\partial_H(p) \xrightarrow{a} \partial_H(p')} \qquad \frac{p \xrightarrow{a} \surd, \quad a \notin H}{\partial_H(p) \xrightarrow{a} \surd}$$

$$\frac{p \xrightarrow{a} p', \quad a \notin I}{\tau_I(p) \xrightarrow{a} \tau_I(p')} \qquad \frac{p \xrightarrow{a} \surd, \quad a \notin I}{\tau_I(p) \xrightarrow{a} \surd} \qquad \frac{p \xrightarrow{a} p', \quad a \in I}{\tau_I(p) \xrightarrow{\tau} \tau_I(p')} \qquad \frac{p \xrightarrow{a} \surd, \quad a \in I}{\tau_I(p) \xrightarrow{\tau} \surd}$$

Table 3.8: The transition relation for $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A})$.

It remains to define the constants and operators of $(\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})$ on the domain of the model. The interpretation $\bar{c}$ in model $\mathcal{M}(\boldsymbol{A})$ of some constant $c$ in the signature of $(\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})$ is defined as the equivalence class $[c]_{\sim_{rb}}$. Note that this definition fulfills the requirement that $\bar{c}$ is a 0-ary function on the domain of $\mathcal{M}(\boldsymbol{A})$. Also for the operators there is a straightforward way to interpret them in the domain of $\mathcal{M}(\boldsymbol{A})$, provided that rooted branching bisimilarity is a *congruence* for all the operators of $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A})$. That is, the following property must be satisfied. Let $\oplus$ be an arbitrary $n$-ary operator in the signature of $(\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})$, where $n$ is some positive natural number; let $p_1, \ldots, p_n, q_1, \ldots, q_n$ be closed terms in $\mathcal{C}((\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A}))$ such that $p_1 \sim_{rb} q_1, \ldots, p_n \sim_{rb} q_n$. Then, the congruence property requires that $\oplus(p_1, \ldots, p_n) \sim_{rb} \oplus(q_1, \ldots, q_n)$.

**Property 3.9. (Congruence)** *Rooted branching bisimilarity, $\sim_{rb}$, is a congruence for the operators of* $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A})$.

**Proof.** It is not difficult to construct rooted branching bisimulations for each of the operators of $(\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})$. The interested reader is referred to [13, Section 2.4.4] for more details. □

Informally, the congruence property says that equivalence classes of processes can be constructed independently of their representatives. Let $p_1, \ldots, p_n$ be closed terms in $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A})$, where $n$ is some positive natural number; for any $n$-ary operator $\oplus$ in the signature of $(\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})$, function $\bar{\oplus}$ is defined on equivalence classes of closed terms as follows: $\bar{\oplus}([p_1]_{\sim_{rb}}, \ldots, [p_n]_{\sim_{rb}}) = [\oplus(p_1, \ldots, p_n)]_{\sim_{rb}}$.

At this point, the construction of model $\mathcal{M}(\boldsymbol{A})$ of the equational theory $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A})$ is complete. The domain consists of the equivalence classes of closed terms in $\mathcal{C}((\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A}))$ modulo rooted branching bisimilarity; the interpretation of any of the constants in the signature of $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A})$ is the corresponding equivalence class; the interpretation of any operator in the signature of the theory is that same operator lifted to equivalence classes of closed terms. Informally, two closed terms in $\mathcal{C}((\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(\boldsymbol{A}))$ that are derivably equal in the equational theory $(\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})$ yield the same equivalence class when they are interpreted in the model $\mathcal{M}(\boldsymbol{A})$, which in turn implies that the corresponding processes are rooted branching bisimilar. Thus, the equational theory $(\mathrm{PA}_\delta^\tau+\mathrm{RN})(\boldsymbol{A})$ is a sound axiomatization of rooted branching bisimilarity. It is even possible to prove a stronger result. Given two rooted branching bisimilar processes, it is *always* possible to prove the equality of these processes in the equational theory $(\mathrm{PA}_\delta^\tau + \mathrm{RN})(\boldsymbol{A})$. Theory $(\mathrm{PA}_\delta^\tau + \mathrm{RN})(\boldsymbol{A})$ is said to be a *complete* axiomatization of model $\mathcal{M}(\boldsymbol{A})$.

**Theorem 3.10. (Soundness and completeness)** *For closed terms* $p, q \in \mathcal{C}((\mathrm{PA}_\delta^\tau + \mathrm{RN})(\boldsymbol{A}))$,
$(\mathrm{PA}_\delta^\tau + \mathrm{RN})(\boldsymbol{A}) \vdash p = q \Leftrightarrow \mathcal{M}(\boldsymbol{A}) \models p = q$.

**Proof.** The theorem is a fairly straightforward result of Property 3.9, the completeness of $\mathrm{BPA}^\tau(\boldsymbol{A})$ for rooted branching bisimilarity (see [36]) and the proof techniques in [10]. □

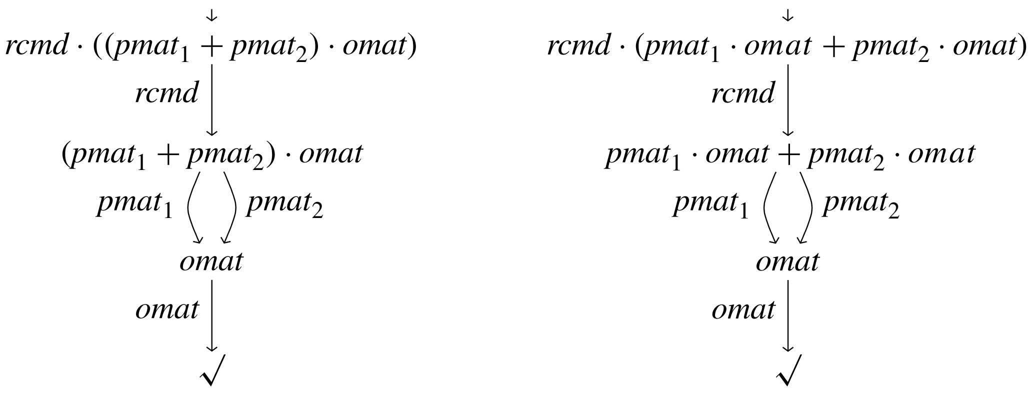



Figure 3.11: Visualizing the semantics of closed $(\mathrm{PA}_\delta^\tau + \mathrm{RN})(\boldsymbol{A})$ terms.

**Example 3.12.** Since the basis of the semantics of $(\mathrm{PA}_\delta^\tau + \mathrm{RN})(\boldsymbol{A})$ is a process space, it is possible to visualize the semantics of closed terms in $\mathcal{C}((\mathrm{PA}_\delta^\tau + \mathrm{RN})(\boldsymbol{A}))$. Consider again Example 3.2. Figure 3.11 depicts the semantics of closed terms $\mathit{rcmd} \cdot ((\mathit{pmat}_1 + \mathit{pmat}_2) \cdot \mathit{omat})$ and $\mathit{rcmd} \cdot (\mathit{pmat}_1 \cdot \mathit{omat} + \mathit{pmat}_2 \cdot \mathit{omat})$. Clearly, these two processes are rooted branching bisimilar, which conforms to Theorem 3.10 (Soundness and completeness) and the conclusion of Example 3.2 that the two closed terms are derivably equal.

# 4 Inheritance in the Algebraic Framework

The goal of this section is to characterize the essential properties of inheritance of behavior. For this purpose, only the object life cycle of a class is considered. All other aspects of a class are omitted. As explained in the introduction, it must be formalized when an object life cycle of some class extends the object life cycle of another class. Based on the two types of inheritance identified in the introduction, in the next subsection, four different inheritance relations on object life cycles are formalized. Each formalization characterizes a different type of extension. In Section 4.2, equational laws preserving each of the inheritance relations are presented. These so-called *axioms of inheritance* are useful for several purposes. First, they illustrate the characteristic properties of each of the inheritance relations. Second, given two object life cycles, they can be used to verify in an algebraic framework whether one life cycle is a subclass of the other life cycle. Finally, they can be used as *transformation rules* to *construct* subclasses of life cycles. Thus, they stimulate the reuse of life-cycle specifications during the design process. In Section 7, the axioms of inheritance are used as a source of inspiration for transformation rules on object life cycles in a framework of Petri nets.

### 4.1 Object life cycles and inheritance relations

The first step in formalizing the concept of inheritance of behavior is to define the notion of an object life cycle. As explained in the introduction, the life cycle of an object describes the order in which the methods of the object may be executed. Since the implementation details of methods are not relevant in this paper, a method is simply represented by an identifier. Recall that an *external* method of some object interacts with the environment of the object; an *internal* method is a method that can only be executed by the object itself and has only internal effects. Internal methods are denoted by the special identifier $\tau$. The set of external methods is denoted $E$. The set of all methods is denoted $M$. That is, $M$ equals $E \cup \{\tau\}$.

It should not come as a surprise that, in this section, an object life cycle is specified by a process-algebraic term. The question is what process-algebraic theory of those introduced in the previous section is best suited to specify object life cycles. Clearly, the set of actions of the theory corresponds to the set of methods $M$. Since $M$ includes the internal method $\tau$ that behaves as a silent action, the theory must include silent actions. Operators that are useful in the specification of object life cycles are choice, sequential composition, and parallel composition. Thus, a process-algebraic theory well suited for specifying object life cycles is the theory $\mathrm{PA}^\tau(M)$.

**Definition 4.1. (Object life cycle)** An object life cycle is a closed $\mathrm{PA}^\tau(M)$ term.

**Example 4.2.** The life cycle of a simple production unit $U$ can be described by the algebraic term $rcmd \cdot pmat \cdot omat$, where *rcmd*, *pmat*, and *omat* are method identifiers in $M$. The production unit first receives a command. After processing some piece of raw material, it delivers processed output material.

Having defined the notion of an object life cycle, the next step is to formalize when one class inherits the life cycle of another class. In the introduction to this paper, two basic forms of inheritance of life cycles have been identified, namely protocol inheritance and projection inheritance, corresponding to the algebraic principles of encapsulation and abstraction, respectively. Therefore, the process-algebraic theory used to formalize inheritance of behavior is the theory $\mathrm{PA}^\tau_\delta$+RN as defined in Section 3, instantiated with the set of method identifiers $M$. Theory $(\mathrm{PA}^\tau_\delta+\mathrm{RN})(M)$ extends the theory $\mathrm{PA}^\tau(M)$, used to define object life cycles, with encapsulation and abstraction. These two operators are exactly the operators needed to reason about inheritance.

**Example 4.3.** Consider again Example 4.2. The operator who is responsible for issuing commands to the production unit sometimes makes a mistake and sends the wrong command. Therefore, production unit $U$ of Example 4.2 is extended with an error-handling facility. A new method $error \in M$ is added that is executed when the unit does not understand the command it receives. The behavior of the new unit $U_1$ is described as follows: $rcmd \cdot (pmat \cdot omat + error)$. Since the addition of an error-handling facility should not influence the correct behavior of the unit, the new unit should be a subclass of $U$. It is not difficult to see that the behavior of the two production units is identical when the new method *error* is not executed. Hence, according to the informal definition of *protocol* inheritance given in the introduction, production unit $U_1$ is indeed a subclass of unit $U$ of Example 4.2.

To show that the definition of protocol inheritance conforms to encapsulation, let $H \subseteq E$ be the singleton $\{error\}$. It follows easily from the axioms for encapsulation of the theory $(\mathrm{PA}^\tau_\delta+\mathrm{RN})(M)$ and Axiom $A6$ that $\partial_H(rcmd \cdot (pmat \cdot omat + error)) = rcmd \cdot pmat \cdot omat$. That is, $\partial_H(U_1) = U$.

**Example 4.4.** Consider a production unit $U_2$ that between the receipt of a command and the main processing step performs a preprocessing step on the raw input material. Unit $U_2$ is specified as follows: $rcmd \cdot ppmat \cdot pmat \cdot omat$. In some sense, the behavior of $U_2$ extends the behavior of production unit $U$ of Example 4.2. Therefore, $U_2$ should be a subclass of $U$. It is not difficult to see that the behaviors of units $U$ and $U_2$ are identical when the new method *ppmat* of $U_2$ is executed, but its effect is ignored. Hence, unit $U_2$ is a subclass of $U$ according to the informal definition of projection inheritance given in the introduction. It is also not difficult to see that projection inheritance conforms to hiding method calls. Let $I$ be the singleton $\{ppmat\}$. It follows from the abstraction axioms of theory $(\mathrm{PA}^{\tau}_{\delta}+\mathrm{RN})(M)$ and Axiom $B1$ that $\tau_I(U_2) = U$.

The subtle difference between the two forms of inheritance introduced so far is that under protocol inheritance methods new in the life cycle of the subclass are executed without taking into account their effect, whereas under projection inheritance they are not executed at all. The examples in the remainder of this section further illustrate this difference.

There are several other possibilities for meaningful definitions of inheritance relations for behavior than the two forms introduced so far. It is, for example, possible to combine the two forms. One might argue that for methods new in the subclass the requirements of both basic forms 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. The two basic forms of inheritance given in the introduction and the two combinations yield four possible inheritance relations. Although probably other meaningful relations can be found, this paper focuses on these four relations. An attempt is made to show that they capture the essentials of inheritance of behavior.

The formal definitions of the four inheritance relations given below are slightly more general than the informal definitions given in the introduction and 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.

Before going to the formal definition of the four inheritance relations, a note on notation is in order. Many formulas in this section contain multiple occurrences of encapsulation and abstraction operators. To avoid large numbers of brackets in formulas, the notation for function composition is overloaded to algebraic operators. Let $x$ be a $(\mathrm{PA}^{\tau}_{\delta}+\mathrm{RN})(M)$ term and let $f$ and $g$ be functions in the signature of $(\mathrm{PA}^{\tau}_{\delta}+\mathrm{RN})(M)$. The notation $f \circ g(x)$ is an abbreviation for the algebraic term $f(g(x))$.

**Definition 4.5. (Inheritance relations)**

*i*) Protocol inheritance:
For any object life cycles $p, q \in \mathcal{C}(\mathrm{PA}^{\tau}(M))$, life cycle $p$ is a subclass of $q$ under *protocol inheritance*, denoted $p \leq_{pt} q$, if and only if there exists an $H \subseteq E$ such that $(\mathrm{PA}^{\tau}_{\delta}+\mathrm{RN})(M) \vdash \partial_H(p) = q$.

*ii*) Projection inheritance:
For any object life cycles $p, q \in \mathcal{C}(\mathrm{PA}^{\tau}(M))$, life cycle $p$ is a subclass of $q$ under *projection inheritance*, denoted $p \leq_{pj} q$, if and only if there exists an $I \subseteq E$ such that $(\mathrm{PA}^{\tau}_{\delta}+\mathrm{RN})(M) \vdash \tau_I(p) = q$.

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

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

The above definitions are formulated in terms of equality of closed $(\mathrm{PA}^{\tau}_{\delta}+\mathrm{RN})(M)$ terms. The completeness of theory $(\mathrm{PA}^{\tau}_{\delta}+\mathrm{RN})(M)$ for rooted branching bisimilarity (Theorem 3.10) implies that this formulation is equivalent to a formulation in terms of rooted branching bisimilarity. Without completeness, the above definitions had been too restrictive. It would have been possible that an object life cycle $p$ after encapsulation and/or abstraction of the appropriate methods and life cycle $q$ would be rooted branching bisimilar, but that this equality would not be derivable from the axioms of $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M)$. 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 4.14 below). It is not clear whether it is meaningful to treat different calls of one method in a different way. In the current definition, 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. In the case study of Section 8, it proves not to be necessary to treat different calls of the same method differently when determining inheritance relationships between life cycles. However, the case study is still only one, relatively small, example. Therefore, in the conclusions of Section 9, it is briefly explained how the framework of this paper can be generalized in such a way that it is possible to treat different calls of one method in a different way in the construction of subclasses.

A final note about Definition 4.5 is that life-cycle inheritance is defined in terms of a composition of operators 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. It would also lack desirable properties such as transitivity.

In reasoning about object life cycles and inheritance relationships between them, it is often convenient to know the set of methods being invoked in a life cycle. For this purpose, the *alphabet* operator is introduced. The alphabet operator yields for each closed $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M)$ term the set of *external* actions that the corresponding process in the operational semantics may perform. The alphabet operator is defined inductively using the structure of basic $\mathrm{BPA}_\delta(M)$ terms, under the assumption that derivability is a congruence for the alphabet operator. In combination with the elimination result for $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M)$ of Property 3.4, this assumption means that it is possible to calculate the alphabet of arbitrary closed $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M)$ terms.

**Definition 4.6. (Alphabet)** The alphabet operator $\alpha : \mathcal{C}((\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M)) \rightarrow \mathcal{P}(E)$ is a function such that, for any closed terms $p, q \in \mathcal{C}((\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M))$, $((\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash p = q) \Rightarrow \alpha(p) = \alpha(q)$. For any $a \in E$ and $p, q \in \mathcal{C}((\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M))$, $\alpha(\delta) = \emptyset$, $\alpha(\tau) = \emptyset$, $\alpha(a) = \{a\}$, $\alpha(\tau \cdot p) = \alpha(p)$, $\alpha(a \cdot p) = \{a\} \cup \alpha(p)$, and $\alpha(p + q) = \alpha(p) \cup \alpha(q)$.

Note that it should be verified that the definition of the alphabet operator is consistent with the axioms of set theory. Inconsistencies arise when the combination of the congruence requirement and the inductive definition allows the derivation of an equality between sets that is not derivable from set theory. It is beyond the scope of this paper to prove that Definition 4.6 is consistent with set theory. A detailed study on the alphabet operator in process algebra can be found in [9]. The alphabet operator is particularly interesting in combination with encapsulation and abstraction. Most of the auxiliary lemmas on encapsulation, abstraction, and the alphabet operator presented in the remainder of this section also appear in [9].

$$\begin{tikzcd} & \leq_{pp} \arrow[dl] \arrow[dr] & \\ \leq_{pt} \arrow[dr] & & \leq_{pj} \arrow[dl] \\ & \leq_{lc} & \end{tikzcd}$$

Figure 4.7: An overview of the four inheritance relations for behavior.

Figure 4.7 gives an overview of the four inheritance relations defined above. The arrows depict strict inclusion relations. The correctness of the inclusion relations between protocol/projection inheritance, on the one hand, and protocol and projection inheritance, on the other hand, follows easily from their definitions. 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 4.8.** *For any closed term* $p \in \mathcal{C}((\mathrm{PA}_\delta^\tau + \mathrm{RN})(M))$ *and sets* $H, I \subseteq E$,

*i)* $\alpha(p) \cap H = \emptyset \Rightarrow (\mathrm{PA}_\delta^\tau + \mathrm{RN})(M) \vdash \partial_H(p) = p$ *and*

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

**Proof.** Only Lemma 4.8 *i)* is proven. The other proof is similar.

It follows from Property 3.4 (Elimination) that there is a basic $\mathrm{BPA}_\delta(M)$ term $t$ such that $(\mathrm{PA}_\delta^\tau + \mathrm{RN})(M) \vdash p = t$. Note that it follows from Definition 4.6 (Alphabet) that $\alpha(p) = \alpha(t)$. Hence, it suffices to show that $\alpha(t) \cap H = \emptyset \Rightarrow (\mathrm{PA}_\delta^\tau + \mathrm{RN})(M) \vdash \partial_H(t) = t$. Given this result, it follows that $\alpha(p) \cap H = \emptyset \Rightarrow (\mathrm{PA}_\delta^\tau + \mathrm{RN})(M) \vdash \partial_H(p) = \partial_H(t) = t = p$. The proof is by induction on the structure of basic $\mathrm{BPA}_\delta(M)$ terms. The symbol $\equiv$ is used to denote syntactical identity of terms.

*i)* Assume $t \equiv \delta$ or $t \equiv a$, for some method identifier $a \in M$. It follows from Axiom $D1$ that $(\mathrm{PA}_\delta^\tau + \mathrm{RN})(M) \vdash \partial_H(t) = t$.

*ii)* Assume $t \equiv a \cdot s$, for some $a \in M$ and basic term $s \in \mathcal{B}(\mathrm{BPA}_\delta(M))$. Then, $(\mathrm{PA}_\delta^\tau + \mathrm{RN})(M) \vdash \partial_H(a \cdot s) \overset{D4}{=} \partial_H(a) \cdot \partial_H(s) \overset{D1, \mathit{Induction}}{=} a \cdot s$.

*iii)* Assume $t \equiv u + v$, for some basic terms $u, v \in \mathcal{B}(\mathrm{BPA}_\delta(M))$. Then, $(\mathrm{PA}_\delta^\tau + \mathrm{RN})(M) \vdash \partial_H(u + v) \overset{D3}{=} \partial_H(u) + \partial_H(v) \overset{\mathit{Induction}\,(2\times)}{=} u + v$.

□

By means of a few examples, it is straightforward to show that the inclusion relations in Figure 4.7 are indeed strict and that there are no inclusion relations between protocol inheritance and projection inheritance.

**Example 4.9.** Consider the three production units $U$, $U_1$, and $U_2$ introduced in Examples 4.2, 4.3, and 4.4, respectively. The arguments in the last two of these three examples show that $U_1 \leq_{pt} U$ and $U_2 \leq_{pj} U$. It is not difficult to see that there exist no $I \subseteq E$ such that $\tau_I(U_1) = U$ and no $H \subseteq E$ such that $\partial_H(U_2) = U$. Hence, there are no inclusion relations between protocol and projection inheritance. In addition, it follows that the inclusions between protocol/projection inheritance, on the one hand, and protocol and projection inheritance, on the other hand, are strict.

In order to show that protocol/projection inheritance is not an empty relation, consider the following example. Unit $U_3$ is a production unit that, as all the units in the other examples, first receives a command. Depending on the command, it continues immediately with its main processing step or it performs a preprocessing step followed by the main processing step. After processing is completed, the processed material is delivered to the environment. The life cycle of $U_3$ is specified as follows: $\mathit{rcmd} \cdot (\mathit{ppmat} \cdot \mathit{pmat} + \mathit{pmat}) \cdot \mathit{omat}$. Unit $U_3$ is a subclass under protocol/projection inheritance of unit $U$ of Example 4.2. It follows simply from the axioms of $(\mathrm{PA}_\delta^\tau + \mathrm{RN})(M)$ that, for $H = I = \{\mathit{ppmat}\}$, $\partial_H(U_3) = \tau_I(U_3) = U$.

Finally, to show that the inclusions between protocol, projection, and life-cycle inheritance are strict, consider a production unit $U_4$ with life cycle $\mathit{rcmd} \cdot (\mathit{ppmat} \cdot \mathit{pmat} \cdot \mathit{omat} + \mathit{error})$. For $H = \{\mathit{error}\}$ and $I = \{\mathit{ppmat}\}$, it easily follows that $\tau_I \circ \partial_H(U_4) = U$. Hence, $U_4 \leq_{lc} U$. It is not difficult to see that there is no relation between $U_4$ and $U$ under any of the other inheritance relations.

Figure 4.7 shows that life-cycle inheritance is more general than any of the other three inheritance relations. The reason for studying all four relations instead of only life-cycle inheritance is one of separation of concerns. Protocol and projection inheritance each characterize a different type of extension of the behavior of life cycles. In a design process, it can be useful to know in what way a life cycle is extended. In the definition of life-cycle inheritance, the type of extension is lost in generality. Moreover, for certain applications it may be useful to restrict inheritance to protocol, projection, or protocol/projection inheritance.

The remainder of this subsection is devoted to studying some basic properties of the four inheritance relations. 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 conforms to the intuitive definitions of protocol and projection inheritance given earlier. For life-cycle inheritance, canonical sets of methods that can be encapsulated and methods that can be hidden do not exist, but it is always possible to choose a partitioning of the methods new in the subclass. To prove these properties, the following two lemmas are needed. They state simple results about encapsulation and abstraction.

**Lemma 4.10.** *For any closed term* $p \in \mathcal{C}((\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M))$ *and sets* $H, I \subseteq E$,

*i)* $\alpha(\partial_H(p)) \cap H = \emptyset$ *and*

*ii)* $\alpha(\tau_I(p)) \cap I = \emptyset$.

**Proof.** Structural induction on basic $\mathrm{BPA}_\delta(M)$ terms. □

**Lemma 4.11.** *For any closed term* $p \in \mathcal{C}((\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M))$ *and sets* $H, H', I, I' \subseteq E$,

*i)* $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \partial_{H' \cup H}(p) = \partial_{H'} \circ \partial_H(p)$ *and*

*ii)* $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \tau_{I' \cup I}(p) = \tau_{I'} \circ \tau_I(p)$.

**Proof.** Structural induction on basic $\mathrm{BPA}_\delta(M)$ terms. □

**Property 4.12.** *For any object life cycles* $p, q \in \mathcal{C}(\mathrm{PA}^\tau(M))$,

*i)* $p \le_{pt} q \Leftrightarrow (\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \partial_{\alpha(p)\backslash\alpha(q)}(p) = q$,

*ii)* $p \le_{pj} q \Leftrightarrow (\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \tau_{\alpha(p)\backslash\alpha(q)}(p) = q$,

*iii)* $p \le_{pp} q \Leftrightarrow (\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \partial_{\alpha(p)\backslash\alpha(q)}(p) = q \wedge (\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \tau_{\alpha(p)\backslash\alpha(q)}(p) = q$, *and*

*iv)* $p \le_{lc} q \Leftrightarrow (\exists H, I : H, I \subseteq \alpha(p)\backslash\alpha(q) \wedge H \cup I = \alpha(p)\backslash\alpha(q) \wedge H \cap I = \emptyset :$
$(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \tau_I \circ \partial_H(p) = q)$.

**Proof.** Only the proof of Property *i*) is given. The proofs of *ii*) and *iv*) are similar; Property *iii*) follows from *i*) and *ii*).

It follows from Definition 4.5 *i*) (Protocol inheritance) that $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \partial_{\alpha(p)\backslash\alpha(q)}(p) = q$ implies $p \le_{pt} q$. To prove the other implication, assume $p \le_{pt} q$. It follows from Lemma 4.11 *i*) and Lemma 4.8 *i*) that there exists a subset $H$ of $\alpha(p)$ such that $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \partial_H(p) = q$. The congruence requirement in Definition 4.6 for the alphabet operator and Lemma 4.10 *i*) show that $H$ cannot contain any methods in $\alpha(q)$. Again Lemmas 4.11 *i*) and 4.8 *i*) yield that $H$ can be extended to all elements of $\alpha(p)$ which are not elements of $\alpha(q)$. □

Property 4.12 *iv*) cannot be strengthened any further. For object life cycles $p$ and $q$ in $\mathcal{C}(\mathrm{PA}^\tau(M))$ such that $p \le_{lc} q$, there does not exist a canonical partitioning of $\alpha(p)\backslash\alpha(q)$ into $H$ and $I$ such that $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \tau_I \circ \partial_H(p) = q$. Assume, for example, that $p \le_{pp} q$. It follows from Property 4.12 *iii*) that for $H = \alpha(p)\backslash\alpha(q)$ and $I = \emptyset$, as well as for $H = \emptyset$ and $I = \alpha(p)\backslash\alpha(q)$, $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \tau_I \circ \partial_H(p) = q$.

Two basic properties that any inheritance relation should satisfy are reflexivity and transitivity. That is, a meaningful inheritance relation should be a *preorder*. It is easy to see that all four inheritance relations are reflexive. Except for life-cycle inheritance, it is also straightforward to show that they are transitive. Showing that life-cycle inheritance is transitive is slightly more involved. The crucial point is the observation that, given two life cycles $p$ and $q$ such that $p \le_{lc} q$ and $q \le_{lc} r$, 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. The following lemmas are needed to formally prove that life-cycle inheritance is transitive.

**Lemma 4.13.** *For any closed term* $p \in \mathcal{C}((\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M))$ *and sets* $H, I \subseteq E$,

*i)* $\alpha(\partial_H(p)) \subseteq \alpha(p)$ *and*

*ii)* $\alpha(\tau_I(p)) \subseteq \alpha(p)$.

**Proof.** Structural induction on basic $\mathrm{BPA}_\delta(M)$ terms. □

**Lemma 4.14.** *For any closed term* $p \in \mathcal{C}((\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M))$ *and sets* $H, I \subseteq E$,
$H \cap I = \emptyset \Rightarrow (\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \tau_I \circ \partial_H(p) = \partial_H \circ \tau_I(p)$.

**Proof.** Structural induction on basic $\mathrm{BPA}_\delta(M)$ terms. □

**Property 4.15.** *Protocol, projection, protocol/projection, and life-cycle inheritance are preorders.*

**Proof.** It follows from Lemma 4.8 that, for any $p \in \mathcal{C}(\mathrm{PA}^\tau(M))$, $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \partial_\emptyset(p) = p$ and $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \tau_\emptyset(p) = p$. Hence, $\leq_{pt}$, $\leq_{pj}$, $\leq_{pp}$, and $\leq_{lc}$ are reflexive.

To show that $\leq_{pt}$ is transitive, let $p$, $q$, and $r$ be object life cycles in $\mathcal{C}(\mathrm{PA}^\tau(M))$ such that $p \leq_{pt} q$ and $q \leq_{pt} r$. Assume that $H, H' \subseteq E$ are sets such that $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \partial_H(p) = q$ and $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \partial_{H'}(q) = r$. Using Lemma 4.11 *i*), $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \partial_{H' \cup H}(p) = \partial_{H'} \circ \partial_H(p) = \partial_{H'}(q) = r$. Hence, $p \leq_{pt} r$, which proves transitivity of protocol inheritance. The proofs for $\leq_{pj}$ and $\leq_{pp}$ are very similar.

To show that $\leq_{lc}$ is transitive, assume $p$, $q$, and $r$ are object life cycles in $\mathcal{C}(\mathrm{PA}^\tau(M))$ such that $p \leq_{lc} q$ and $q \leq_{lc} r$. Hence, there are subsets $H$, $H'$, $I$, and $I'$ of $E$ such that $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \tau_I \circ \partial_H(p) = q$ and $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \tau_{I'} \circ \partial_{H'}(q) = r$. It follows from Property 4.12 *iv*) that these sets can be chosen such that $H$ and $I$ are subsets of $\alpha(p) \backslash \alpha(q)$ and such that $H'$ and $I'$ are subsets of $\alpha(q) \backslash \alpha(r)$. The congruence requirement in Definition 4.6 for the alphabet operator and Lemma 4.13 yield that $\alpha(r) \subseteq \alpha(q) \subseteq \alpha(p)$, and hence that $(H \cup I) \cap (H' \cup I') = \emptyset$. Using Lemmas 4.11 and 4.14, it follows that $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \tau_{I' \cup I} \circ \partial_{H' \cup H}(p) = \tau_{I'} \circ \tau_I \circ \partial_{H'} \circ \partial_H(p) = \tau_{I'} \circ \partial_{H'} \circ \tau_I \circ \partial_H(p) = \tau_{I'} \circ \partial_{H'}(q) = r$. Hence, $p \leq_{lc} r$. □

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 $\mathrm{PA}^\tau(M)$.

**Definition 4.16. (Subclass equivalence)** Let $\approx_*$, where $* \in \{pp, pt, pj, lc\}$, be the equivalence relation induced by $\leq_*$. That is, for any object life cycles $p$ and $q$ in $\mathcal{C}(\mathrm{PA}^\tau(M))$, $p \approx_* q \Leftrightarrow p \leq_* q \wedge q \leq_* p$. Processes $p$ and $q$ are said to be *subclass equivalent* under $*$ inheritance.

The four subclass-equivalence relations indeed all coincide with derivability from the axioms of $\mathrm{PA}^\tau(M)$.

**Property 4.17.** *For any object life cycles* $p, q \in \mathcal{C}(\mathrm{PA}^\tau(M))$ *and* $* \in \{pp, pt, pj, lc\}$,
$p \approx_* q \Leftrightarrow \mathrm{PA}^\tau(M) \vdash p = q$.

**Proof.** The result is only proven for protocol inheritance. The other proofs are similar. First, assume that $\mathrm{PA}^\tau(M) \vdash p = q$. It follows from Lemma 4.8 *i*) and Property 3.6 (Conservative extension) that $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \partial_\emptyset(p) = p = q$ and $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \partial_\emptyset(q) = q = p$. 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 $q \leq_{pt} p$. Property 4.12 *i*) yields that $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \partial_{\alpha(p)\backslash\alpha(q)}(p) = q$ and $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \partial_{\alpha(q)\backslash\alpha(p)}(q) = p$. It follows from the congruence requirement in Definition 4.6 for the alphabet operator and Lemma 4.13 *i*) that $\alpha(q) \subseteq \alpha(p)$ and $\alpha(p) \subseteq \alpha(q)$. Hence, $\alpha(p) = \alpha(q)$, which means that $\alpha(p) \backslash \alpha(q) = \alpha(q) \backslash \alpha(p) = \emptyset$. Lemma 4.8 *i*) yields that $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash \partial_{\alpha(p)\backslash\alpha(q)}(p) = p$ and hence that $(\mathrm{PA}_\delta^\tau{+}\mathrm{RN})(M) \vdash p = q$. Property 3.6 (Conservative extension) implies that $\mathrm{PA}^\tau(M) \vdash p = q$. □

Note that, as a consequence of Property 3.6, Theorem 3.10, and Property 4.17, two life cycles are subclass equivalent under any form of inheritance if and only if they are rooted branching bisimilar.

It is important to note that the inheritance preorders are not precongruences for the operators of $\mathrm{PA}^{\tau}(M)$. That is, it is not possible to apply them in arbitrary contexts.

**Example 4.18.** It is easy to see that, for any distinct $a, b \in E$, $a+b \leq_{pt} a$, because $(\mathrm{PA}_{\delta}^{\tau}+\mathrm{RN})(M) \vdash \partial_{\{b\}}(a+b) = a$. 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 $(\mathrm{PA}_{\delta}^{\tau}+\mathrm{RN})(M) \vdash \partial_{\{b\}}(b \cdot (a+b)) = \delta$, $b \cdot (a+b) \not\leq_{pt} b \cdot a$. A similar example can be constructed for projection inheritance.

Another instructive example is the following, where first a subclass of a single method call $a$ under protocol inheritance is constructed and, subsequently, this subclass is further refined to a more specialized subclass. For any distinct $a, b, c, d \in E$, $a+b \leq_{pt} a$ 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 is easy to prove 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 the remainder of this subsection, it is formalized under which conditions it is allowed to refine a subclass to a more specialized subclass. Before going into more details, the notion of a context is defined. A context is a closed $\mathrm{PA}^{\tau}(M)$ term that contains a "hole."

**Definition 4.19. (Context)** The most simple context is simply a hole, denoted by an underscore "_". Furthermore, for any closed term $p \in \mathcal{C}(\mathrm{PA}^{\tau}(M))$ and context $C$, $p \oplus C$ and $C \oplus p$ are contexts, where $\oplus \in \{\cdot, +, \|, \mathbin{\|\!\_}\}$.

**Example 4.20.** Let $a, b$, and $c$ be three method identifiers in $M$. The following are simple examples of contexts: $a + \_$, $\_ \cdot a$, and $a \parallel b \cdot \_$. Substitution of a closed $\mathrm{PA}^{\tau}(M)$ term $p$ in a context $C$ is denoted $C[p]$. Thus, for context $C$ defined as $a + \_$, $C[c]$ equals $a + c$. Substituting method $c$ in any of the other two contexts yields the closed $\mathrm{PA}^{\tau}(M)$ terms $c \cdot a$ and $a \parallel b \cdot c$.

Property 4.22 given below defines the conditions that must be satisfied when refining a subclass. To prove this property, the following lemma is needed. It states that the encapsulation and abstraction operators distribute over merge and left merge. Hence, it follows from the encapsulation and abstraction axioms given in Table 3.1 that they distribute over all operators that may appear in a context or in a term representing a life cycle. Note that the lemma does not carry over to an ACP-style algebraic setting with communication (see [10, 11, 33]).

**Lemma 4.21.** *For any closed terms $p, q \in \mathcal{C}((\mathrm{PA}_{\delta}^{\tau}+\mathrm{RN})(M))$, $H, I \subseteq E$,*

*i)* $(\mathrm{PA}_{\delta}^{\tau}+\mathrm{RN})(M) \vdash \partial_H(p \mathbin{\|\!\_} q) = \partial_H(p) \mathbin{\|\!\_} \partial_H(q)$,

*ii)* $(\mathrm{PA}_{\delta}^{\tau}+\mathrm{RN})(M) \vdash \partial_H(p \parallel q) = \partial_H(p) \parallel \partial_H(q)$,

*iii)* $(\mathrm{PA}_{\delta}^{\tau}+\mathrm{RN})(M) \vdash \tau_I(p \mathbin{\|\!\_} q) = \tau_I(p) \mathbin{\|\!\_} \tau_I(q)$, *and*

*iv)* $(\mathrm{PA}_{\delta}^{\tau}+\mathrm{RN})(M) \vdash \tau_I(p \parallel q) = \tau_I(p) \parallel \tau_I(q)$.

**Proof.** Property *i)* is proven by induction on the sum of the number of symbols in $p$ and the number of symbols in $q$. Property *ii)* follows immediately from *i)* and Axioms $M1$ and $D3$ in Table 3.1. Properties *iii)* and *iv)* are proven in a similar way. □

Let $p, q, r, C, H, H', I$, and $I'$ be as in Property 4.22. Informally, the condition $\alpha(r) \cap H = \emptyset$ in Property 4.22 *i)* means that it is not allowed to encapsulate methods in $p$ that are *not* encapsulated in $C[q]$;

methods being encapsulated in $p$ are those in $H$ and methods not encapsulated in $C[q]$ are those 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 4.22.** *Let $p$, $q$, and $r$ be object life cycles in $\mathcal{C}(\mathrm{PA}^{\tau}(M))$ and let $C$ be a context. Let $H$, $H'$, $I$, and $I'$ be subsets of $E$.*

*i*) *If $p \leq_{pt} q$, with $H$ such that $(\mathrm{PA}^{\tau}_{\delta}+\mathrm{RN})(M) \vdash \partial_H(p) = q$, and $C[q] \leq_{pt} r$, then*
$\alpha(r) \cap H = \emptyset \Rightarrow C[p] \leq_{pt} r$.

*ii*) *If $p \leq_{pj} q$, with $I$ such that $(\mathrm{PA}^{\tau}_{\delta}+\mathrm{RN})(M) \vdash \tau_I(p) = q$, and $C[q] \leq_{pj} r$, then*
$\alpha(r) \cap I = \emptyset \Rightarrow C[p] \leq_{pj} r$.

*iii*) *If $p \leq_{pp} q$, with $H$ and $I$ such that $(\mathrm{PA}^{\tau}_{\delta}+\mathrm{RN})(M) \vdash \partial_H(p) = q$ and $(\mathrm{PA}^{\tau}_{\delta}+\mathrm{RN})(M) \vdash \tau_I(p) = q$, and if $C[q] \leq_{pp} r$, then*
$\alpha(r) \cap (H \cup I) = \emptyset \Rightarrow C[p] \leq_{pp} r$.

*iv*) *If $p \leq_{lc} q$ with $H$ and $I$ such that $(\mathrm{PA}^{\tau}_{\delta}+\mathrm{RN})(M) \vdash \tau_I \circ \partial_H(p) = q$, and if $C[q] \leq_{lc} r$ with $H'$ and $I'$ such that $(\mathrm{PA}^{\tau}_{\delta}+\mathrm{RN})(M) \vdash \tau_{I'} \circ \partial_{H'}(C[q]) = r$, 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, Property *i*) is proven. Let $H' \subseteq E$ be such that $(\mathrm{PA}^{\tau}_{\delta}+\mathrm{RN})(M) \vdash \partial_{H'}(C[q]) = r$. A consequence of the axioms of the encapsulation operator and Lemma 4.21 above is that encapsulation distributes over all operators of $\mathrm{PA}^{\tau}(M)$. This result is used in the second step of the following derivation. In the final step, the requirement that $\alpha(r) \cap H = \emptyset$ is used.

$$\partial_{H' \cup H}(C[p]) \stackrel{4.11}{=} \partial_{H'} \circ \partial_H(C[p]) = \partial_{H'} \circ \partial_H(C[\partial_H(p)]) = \partial_{H'} \circ \partial_H(C[q]) \stackrel{4.11}{=}$$
$$\partial_{H' \cup H}(C[q]) = \partial_{H \cup H'}(C[q]) \stackrel{4.11}{=} \partial_H \circ \partial_{H'}(C[q]) = \partial_H(r) \stackrel{4.8}{=} r.$$

This derivation shows that $C[p] \leq_{pt} r$.

Properties *ii*) and *iii*) are proven similarly.

Property *iv*) is shown as follows. The distributivity of encapsulation and abstraction over all operators of $\mathrm{PA}^{\tau}(M)$ is used in the third step of the following derivation. 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.

$$\tau_{I' \cup I} \circ \partial_{H' \cup H}(C[p]) \stackrel{4.11}{=} \tau_{I'} \circ \tau_I \circ \partial_{H'} \circ \partial_H(C[p]) \stackrel{4.14}{=} \tau_{I'} \circ \partial_{H'} \circ \tau_I \circ \partial_H(C[p]) =$$
$$\tau_{I'} \circ \partial_{H'} \circ \tau_I \circ \partial_H(C[\tau_I \circ \partial_H(p)] = \tau_{I'} \circ \partial_{H'} \circ \tau_I \circ \partial_H(C[q]) \stackrel{4.11,4.14}{=}$$
$$\tau_I \circ \partial_H \circ \tau_{I'} \circ \partial_{H'}(C[q]) = \tau_I \circ \partial_H(r) \stackrel{4.8}{=} r. \qquad \square$$

**Example 4.23.** It is easy to check that the examples of Example 4.18 are consistent with Property 4.22 *i*). The following example shows an application of Property 4.22 *iv*). Let $a, b, c \in E$ be distinct methods. Consider the context $C$ defined as $\_ + b$. By encapsulating method $b$, it is easy to show that $C[a] \leq_{lc} a$. In addition, $a \cdot c \leq_{lc} a$, 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 4.22 *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, it follows 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, Property 4.22 *iv*) cannot be applied. 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 subsection show that the definitions of the four inheritance relations are sound. All four relations are preorders. The induced equivalence relations coincide with rooted branching bisimilarity. 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 with practical examples has to show whether they are not too restrictive.

## 4.2 Axioms of inheritance

To get a better understanding of the four inheritance relations, it is useful to know what general subclass relations are valid under each form of inheritance. The *axioms of inheritance* given in this subsection show that extending a life cycle with an alternative behavior yields a subclass under protocol inheritance. They also show that extending a life cycle by inserting some behavior in between sequential parts of the original life cycle, as well as putting some 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 extensions.

Note that no attempt is made to find a complete set of axioms characterizing the inheritance relations. The theory developed in this section is not intended for use in practical applications. The main goal is to learn to understand inheritance of behavior. The axioms presented in this section are helpful in achieving this goal. In addition, in Section 7, they form a source of inspiration for transformation rules in a framework based on Petri nets, which is a framework much closer to the object-oriented methods used in practice than process algebra.

**Property 4.24.** *For any closed terms* $p, q \in \mathcal{C}(\mathrm{PA}^{\tau}(M))$ *and method* $b \in E \backslash \alpha(q)$,

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

**Proof.** Let $H = \{b\}$.
$(\mathrm{PA}_{\delta}^{\tau}+\mathrm{RN})(M) \vdash \partial_H(q + b \cdot p) \overset{D3,D4}{=} \partial_H(q) + \partial_H(b) \cdot \partial_H(p) \overset{D2,4.8}{=} q + \delta \cdot \partial_H(p) \overset{A7,A6}{=} q.$
Hence, $q + b \cdot p \leq_{pt} q$. □

Method $b$ functions as some sort of a "*guard*." Blocking the guard means that the environment cannot choose the behavior $b \cdot p$. For this reason, $b$ may not appear in the alphabet of $q$, since blocking $b$ would otherwise change the behavior of $q$. Axiom *PT* shows that encapsulation is sufficient to capture inheritance by means of the choice operator. It also shows that it is sufficient to encapsulate a single method, whereas the canonical set of Property 4.12, $\alpha(q+b\cdot p)\backslash\alpha(q)$, might be much larger. Finally, note that Axiom *PT* has a variant in which the term $p$ is absent. This variant states that it is allowed to extend a life cycle with the alternative of a single method call. This extension is not captured by *PT*, because the theory $\mathrm{PA}^{\tau}(M)$ does not have a means to express the empty process. The subclass relationship between production units $U$ and $U_1$ discussed in Example 4.3 can be proven by means of this variant of *PT* in combination with the context rule of Property 4.22 *i*). Axiom *PT* is the most characteristic example of protocol inheritance.

Property 4.27 given below lists several axioms that are characteristic for projection inheritance. The following two lemmas are needed to prove the correctness of these axioms for projection inheritance.

**Lemma 4.25.** *For any closed term* $p \in \mathcal{C}(\mathrm{PA}^{\tau}(M))$ *and* $I \subseteq E$,
$\alpha(p) \subseteq I \Rightarrow (\mathrm{PA}_{\delta}^{\tau}+\mathrm{RN})(M) \vdash \tau_I(p) = \tau.$

**Proof.** Structural induction on basic $\mathrm{BPA}(M)$ terms. □

Note that this lemma cannot be proven for arbitrary closed $(\mathrm{PA}_{\delta}^{\tau}+\mathrm{RN})(M)$ terms, because it is not possible to hide constant $\delta$.

**Lemma 4.26.** *For any closed term* $p \in \mathcal{C}((\mathrm{PA}_{\delta}^{\tau}+\mathrm{RN})(M))$,

*i*) $(\mathrm{PA}_{\delta}^{\tau}+\mathrm{RN})(M) \vdash p \mathbin{\|\!\!\_} \tau = p$,
*ii*) $(\mathrm{PA}_{\delta}^{\tau}+\mathrm{RN})(M) \vdash p \parallel \tau = \tau \cdot p + p$.

**Proof.** Property *i)* is proven by induction on the structure of basic $\mathrm{BPA}_\delta(M)$ terms. Property *ii)* follows immediately from *i)* and Axioms $M1$ and $M2$. □

**Property 4.27.** *For any $q, q_0, q_1$, and $r$ in $\mathcal{C}(\mathrm{PA}^\tau(M))$ such that $\alpha(r) \subseteq E\backslash(\alpha(q) \cup \alpha(q_0) \cup \alpha(q_1))$,*

| | |
|---|---|
| $q \cdot r \leq_{pj} q$ | $PJ1$ |
| $q \cdot (r \cdot (q_0 + q_1) + q_0) \leq_{pj} q \cdot (q_0 + q_1)$ | $PJ2$ |
| $q_0 \cdot (q_1 \parallel r) \leq_{pj} q_0 \cdot q_1$ | $PJ3$ |

**Proof.** Let $I$ be equal to $\alpha(r)$.

$(\mathrm{PA}_\delta^\tau + \mathrm{RN})(M) \vdash \tau_I(q \cdot r) \overset{TI4}{=} \tau_I(q) \cdot \tau_I(r) \overset{4.8}{=} q \cdot \tau_I(r) \overset{4.25}{=} q \cdot \tau \overset{B1}{=} q,$

which proves $PJ1$. The proof for $PJ2$ is similar; only Axiom $B2$ is used instead of $B1$.

$(\mathrm{PA}_\delta^\tau + \mathrm{RN})(M) \vdash \tau_I(q \cdot (r \cdot (q_0 + q_1) + q_0)) \overset{TI3,TI4}{=} \tau_I(q) \cdot (\tau_I(r) \cdot (\tau_I(q_0) + \tau_I(q_1)) + \tau_I(q_0)) \overset{4.8,4.25}{=}$
$q \cdot (\tau \cdot (q_0 + q_1) + q_0) \overset{B2}{=} q \cdot (q_0 + q_1).$

Axiom $PJ3$ is proved as follows.

$(\mathrm{PA}_\delta^\tau + \mathrm{RN})(M) \vdash \tau_I(q_0 \cdot (q_1 \parallel r)) \overset{TI4,4.21}{=} \tau_I(q_0) \cdot (\tau_I(q_1) \parallel \tau_I(r)) \overset{4.8,4.25}{=} q_0 \cdot (q_1 \parallel \tau) \overset{4.26}{=}$
$q_0 \cdot (\tau \cdot q_1 + q_1) \overset{A6,B2}{=} q_0 \cdot q_1.$ □

Axioms $PJ1$ and $PJ2$ are inspired by the two 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. Axiom $PJ3$ shows that putting alternative behavior in parallel with part of the original life cycle also yields a subclass under projection inheritance. The inheritance relationship between production units $U$ and $U_2$ discussed in Example 4.4 follows from Axiom $PJ1$ and the context rule of Property 4.22 *ii)*.

**Example 4.28.** Consider production unit $U$ of Example 4.2. Production unit $U_5$ sends a processing-ready signal when the main processing step is completed. The signal is sent in parallel with the delivery of output material. The life cycle of $U_5$ is described as follows: $rcmd \cdot pmat \cdot (omat \parallel sprdy)$. It follows from Axiom $PJ3$ that $U_5 \leq_{pj} U$.

**Property 4.29.** *For any $q_0, q_1$, and $r$ in $\mathcal{C}(\mathrm{PA}^\tau(M))$ such that $\alpha(r) \subseteq E\backslash(\alpha(q_0) \cup \alpha(q_1))$, and $b \in E\backslash(\alpha(q_0) \cup \alpha(q_1))$,*

| | |
|---|---|
| $q_0 \cdot (b \cdot r \cdot q_1 + q_1) \leq_{pp} q_0 \cdot q_1$ | $PP$ |

**Proof.** Let $H$ be equal to $\{b\}$ and $I$ be equal to $\alpha(r) \cup \{b\}$. Note that $b$ is not equal to $\tau$.

$(\mathrm{PA}_\delta^\tau + \mathrm{RN})(M) \vdash \partial_H(q_0 \cdot (b \cdot r \cdot q_1 + q_1)) \overset{D4,D3}{=} \partial_H(q_0) \cdot (\partial_H(b) \cdot \partial_H(r \cdot q_1) + \partial_H(q_1)) \overset{D2,4.8}{=}$
$q_0 \cdot (\delta \cdot \partial_H(r \cdot q_1) + q_1) \overset{A7,A6}{=} q_0 \cdot q_1.$

Furthermore,

$(\mathrm{PA}_\delta^\tau + \mathrm{RN})(M) \vdash \tau_I(q_0 \cdot (b \cdot r \cdot q_1 + q_1)) \overset{TI4,TI3}{=} \tau_I(q_0) \cdot (\tau_I(b \cdot r) \cdot \tau_I(q_1) + \tau_I(q_1)) \overset{4.8,4.25}{=}$
$q_0 \cdot (\tau \cdot q_1 + q_1) \overset{A6,B2}{=} q_0 \cdot q_1.$

It follows from the above two derivations that $q_0 \cdot ((b \cdot r) \cdot q_1 + q_1) \leq_{pp} q_0 \cdot q_1$. □

Axiom $PP$ shows that under protocol/projection inheritance it is allowed to *postpone* behavior. In case it is possible to specify iterative behavior, a nice variant of $PP$ is an axiom in which the behavior $b \cdot r$ is iterated arbitrarily many times before continuing with $q_1$. The inheritance relationship between production units $U$ and $U_3$ shown in Example 4.9 can also be proven by means of a variant of Axiom $PP$ without term $r$ in combination with the context rule of Property 4.22 *iii)*.

The final property of this section gives a few axioms of life-cycle inheritance.

**Property 4.30.** *For any* $p, q_0, q_1$, *and* $r$ *in* $\mathcal{C}(\mathrm{PA}^\tau(M))$ *such that* $\alpha(r) \subseteq E \backslash (\alpha(q_0) \cup \alpha(q_1))$, *and* $b \in E \backslash (\alpha(r) \cup \alpha(q_0) \cup \alpha(q_1))$,

$$\begin{array}{ll}
q_0 \cdot (r \cdot (q_1 + b \cdot p)) \leq_{lc} q_0 \cdot q_1 & LC1 \\
q_0 \cdot ((q_1 + b \cdot p) \parallel r) \leq_{lc} q_0 \cdot q_1 & LC2 \\
q_0 \cdot (q_1 \parallel (r + b \cdot p)) \leq_{lc} q_0 \cdot q_1 & LC3
\end{array}$$

**Proof.** Let $H$ be equal to $\{b\}$ and $I$ be equal to $\alpha(r)$.

$(\mathrm{PA}_\delta^\tau + \mathrm{RN})(M) \vdash \tau_I \circ \partial_H(q_0 \cdot (r \cdot (q_1 + b \cdot p))) \overset{D4, D3, TI4, TI3}{=}$
$\tau_I \circ \partial_H(q_0) \cdot (\tau_I \circ \partial_H(r) \cdot (\tau_I \circ \partial_H(q_1) + \tau_I \circ \partial_H(b) \cdot \tau_I \circ \partial_H(p))) \overset{D2, 4.8, 4.25}{=}$
$q_0 \cdot (\tau \cdot (q_1 + \delta \cdot \tau_I \circ \partial_H(p))) \overset{A7, A6, B1}{=} q_0 \cdot q_1$,

which proves $LC1$. Axiom $LC2$ is proven as follows:

$(\mathrm{PA}_\delta^\tau + \mathrm{RN})(M) \vdash \tau_I \circ \partial_H(q_0 \cdot ((q_1 + b \cdot p) \parallel r)) \overset{D4, 4.21, D3}{=}$
$\tau_I(\partial_H(q_0) \cdot (\partial_H(q_1) + \partial_H(b) \cdot \partial_H(p)) \parallel \partial_H(r))) \overset{D2, 4.8}{=} \tau_I(q_0 \cdot ((q_1 + \delta \cdot \partial_H(p)) \parallel r)) \overset{A7, A6}{=}$
$\tau_I(q_0 \cdot (q_1 \parallel r))$,

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

Axioms $LC1$ through $LC3$ are combinations of the axioms for protocol and projection inheritance. They are illustrative for life-cycle inheritance. It is not difficult to find several more of such combinations. It is possible to prove the above axioms by means of the context rule of Property 4.22 *iv*). The proof above uses basic lemmas and axioms, because they yield a more readable proof.

**Example 4.31.** Consider a production unit $U_6$ with life cycle $rcmd \cdot ((pmat + ppmat \cdot pmat) \parallel ssps) \cdot omat$. Upon receipt of a command, unit $U_6$ sends a start-processing signal. In parallel, it starts its main processing step, possibly preceded by a preprocessing step. It follows from Axiom $LC2$ and the context rule of Property 4.22 *iv*) that $U_6$ is a subclass under life-cycle inheritance of unit $U$ of Example 4.2.

### 4.3 Concluding remarks

This section has presented a characterization of inheritance of behavior in a simple process-algebraic setting. Four inheritance relations have been defined, all in terms of the algebraic concepts of encapsulation and abstraction. Several basic properties of the relations show that they are theoretically sound. The axioms of inheritance given in the previous subsection show that life-cycle inheritance captures three basic operators to construct a subclass from a superclass, namely choice, sequential composition, and parallel composition. Since these three operators are fundamental to any (concurrent) object-oriented language, it appears that encapsulation and abstraction capture the essence of inheritance of behavior. The axioms of inheritance show how to create subclasses of some given class in a *constructive* way at design time. Thus, the axioms stimulate the *reuse* of object life cycles during the design process.

## 5 Petri Nets

The popularity of Petri-net theory as a formalism for modeling processes is due to both its easy-to-understand graphical representation of Petri-net models and its potential as a technique for formally analyzing concurrent processes. Petri nets were introduced in 1962 by Carl Adam Petri [60]. Since then, Petri-net theory has been extended in many ways and applied to many kinds of problems. This section contains an introduction to the framework of so-called Place/Transition nets, abbreviated P/T nets.

Section 5.1 introduces the notion of bags, which play an important role in any Petri-net formalism. Section 5.2 presents the class of *labeled* P/T nets. Section 5.3 reviews some theoretical results about P/T nets that are the basis for some of the analysis techniques available to analyze Petri-net models. In Section 5.4, the subclass of free-choice P/T nets is introduced. The reason to consider this subclass of labeled P/T nets is that analysis techniques for free-choice P/T nets are more efficient than the techniques for the general class of labeled P/T nets.

The introduction to labeled P/T nets and free-choice P/T nets given in this section is not exhaustive. Its only goal is to provide a basis for the remainder of this paper. As mentioned in the introduction, for readers already familiar with Petri-net theory, it is sufficient to browse through this section to get acquainted with notation. Good starting points for further reading on Petri nets are [19, 27, 41, 55, 59, 64, 65, 66].

## 5.1 Notations for bags

In this paper, bags are defined as finite multi-sets of elements from some alphabet $A$. A bag over alphabet $A$ can be considered as a function from $A$ to the natural numbers $\mathbb{N}$ such that only a finite number of elements from $A$ is assigned a non-zero function value. For some bag $X$ over alphabet $A$ and $a \in A$, $X(a)$ denotes the number of occurrences of $a$ in $X$, often called the cardinality of $a$ in $X$. The set of all bags over $A$ is denoted $\mathcal{B}(A)$. Note that any finite set of elements from $A$ also denotes a unique bag over $A$, namely the function yielding 1 for every element in the set and 0 otherwise. The empty bag, which is the function yielding 0 for any element in $A$, is denoted $\mathbf{0}$. For the explicit enumeration of a bag, a notation similar to the notation for sets is used, but using square brackets instead of curly brackets and using superscripts to denote the cardinality of the elements. For example, $[a^2, b, c^3]$ denotes the bag with two elements $a$, one $b$, and three elements $c$; the bag $[a^2 \mid P(a)]$ contains two elements $a$ for every $a$ such that $P(a)$ holds, where $P$ is some predicate on symbols of the alphabet under consideration. To denote individual elements of a bag, the same symbol "$\in$" is used as for sets: For any bag $X$ over alphabet $A$ and element $a \in A$, $a \in X$ if and only if $X(a) > 0$. The sum of two bags $X$ and $Y$, denoted $X \uplus Y$, is defined as $[a^n \mid a \in A \wedge n = X(a) + Y(a)]$. The difference of $X$ and $Y$, denoted $X - Y$, is defined as $[a^n \mid a \in A \wedge n = (X(a) - Y(a)) \max 0]$. The binding of sum and difference is left-associative. The restriction of $X$ to some domain $D \subseteq A$, denoted $X \upharpoonright D$, is defined as $[a^{X(a)} \mid a \in D]$. Restriction binds stronger than sum and difference. The notion of subbags is defined as expected: Bag $X$ is a subbag of $Y$, denoted $X \leq Y$, if and only if for all $a \in A$, $X(a) \leq Y(a)$.

## 5.2 Labeled P/T nets

Let $U$ be some universe of identifiers; let $L$ be some set of labels.

**Definition 5.1. (Labeled P/T net)** An $L$-labeled Place/Transition net, or simply labeled P/T net, is a tuple $(P, T, F, \ell)$ where

*i)* $P \subseteq U$ is a finite set of *places*;
*ii)* $T \subseteq U$ is a finite set of *transitions* such that $P \cap T = \emptyset$;
*iii)* $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the *flow relation*;
*iv)* $\ell : T \rightarrow L$ is a *labeling function*.

Let $N = (P, T, F, \ell)$ be a labeled P/T net. Elements of $P \cup T$ are referred to as *nodes*. A node $x \in P \cup T$ is called an *input node* of another node $y \in P \cup T$ if and only if there exists a directed arc from $x$ to $y$; that is, if and only if $xFy$. Node $x$ is called an *output node* of $y$ if and only if there exists a directed arc from $y$ to $x$. If $x$ is a place in $P$, it is called an input place or an output place; if it is a transition, it is called an input or an output transition. The set of all input nodes of some node $x$ is called the *preset* of $x$; its set of output nodes is called the *postset*. Two auxiliary functions $\boldsymbol{i}_N\_, \boldsymbol{o}_N\_ : (P \cup T) \rightarrow \mathcal{P}(P \cup T)$ are defined that assign to each node its preset and postset, respectively. For any node $x \in P \cup T$, $\boldsymbol{i}_N x = \{y \mid yFx\}$ and

$\boldsymbol{o}_N x = \{y \mid xFy\}$. In an unambiguous context, the subscript identifying the P/T net may be omitted from the preset and postset functions.

**Example 5.2.** Besides a mathematical definition, labeled P/T nets also have an unambiguous graphical representation. Figure 5.3 shows a labeled P/T net representing a simple production unit. Places are depicted by circles, transitions by rectangles, and the flow relation by arcs. The small black dots residing in the places are called tokens, which are introduced below. Attached to each place is its identifier. Attached to each transition are its identifier and its label. In the example of Figure 5.3, it is assumed that the labeling function is the identity function.

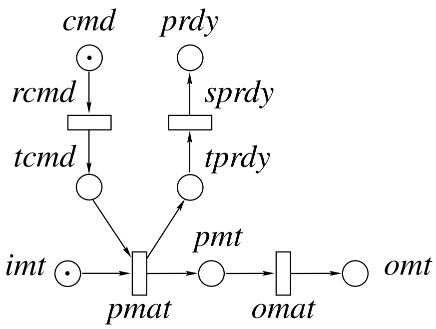



Figure 5.3: The graphical representation of a labeled P/T net.

The class of Petri nets introduced in Definition 5.1 is sometimes referred to as the class of *ordinary* (labeled) P/T nets to distinguish it from the class of Petri nets that allows more than one arc between a pair of nodes. In the context of this paper, it is reasonable to allow at most one arc between any two nodes of a P/T net.

A labeled P/T net as defined above defines the static structure of a process. However, labeled P/T nets also have a behavior. The behavior of a P/T net is determined by its *state*. To express the state of a net, its places may contain *tokens*. In labeled P/T nets, tokens are nothing more than simple markers (see the graphical representation of a labeled P/T net in Figure 5.3). The distribution of tokens over the places is often called the *marking* of the net.

**Definition 5.4. (Marked, labeled P/T net)** A *marked*, $L$-labeled P/T net is a pair $(N, s)$, where $N = (P, T, F, \ell)$ is an $L$-labeled P/T net and where $s$ is a bag over $P$ denoting the marking of the net. The set of all marked, $L$-labeled P/T nets is denoted $\mathcal{N}(L)$.

The behavior of marked, labeled P/T nets is defined by a so-called *firing rule*, which is simply a transition relation defining the change in the state of a marked net when executing an action. To define the firing rule, it is necessary to formalize when a net is allowed to execute a certain action.

**Definition 5.5. (Transition enabling)** Let $(N, s)$ be a marked, labeled P/T net in $\mathcal{N}(L)$, where $N = (P, T, F, \ell)$. A transition $t \in T$ is *enabled*, denoted $(N, s)[t\rangle$, if and only if each of its input places contains at least one token. That is, $(N, s)[t\rangle \Leftrightarrow \boldsymbol{i}t \leq s$. (Note that the set $\boldsymbol{i}t$ is interpreted as a bag.)

**Example 5.6.** In the marked net of Figure 5.3, transition *rcmd* is enabled. None of the other transitions is enabled.

When a transition $t$ of a labeled P/T net is enabled, the net can *fire* this transition. Upon firing, $t$ removes a token from each of its input places and it adds a token to each of its output places. This means that upon firing $t$, the marked net $(N, s)$ changes into another marked net $(N, s - \boldsymbol{i}t \uplus \boldsymbol{o}t)$. When firing a transition, the labeled P/T net executes an *action*, which is represented by its label.

**Definition 5.7. (Firing rule)** The firing rule $\_\,[\_\rangle\,\_ \subseteq \mathcal{N}(L) \times L \times \mathcal{N}(L)$ is the smallest relation satisfying for any $(N, s)$ in $\mathcal{N}(L)$, with $N = (P, T, F, \ell)$, and any $t \in T$,
$(N, s)[t\rangle \Rightarrow (N, s)\ [\ell(t)\rangle\ (N, s - \boldsymbol{it} \uplus \boldsymbol{ot})$.

Tokens that are removed from the marking when firing a transition are referred to as *consumed* tokens or the *consumption* of a transition; tokens that are added to the marking are referred to as *produced* tokens or the *production* of a transition.

**Example 5.8.** In the labeled P/T net of Figure 5.3, firing transition *rcmd* changes the marking from [*cmd*, *imt*] to [*tcmd*, *imt*], thus enabling transition *pmat*. Firing transition *pmat* yields the marking [*tprdy*, *pmt*]. This new marking enables both transitions *sprdy* and *omat*. These two transitions are independent and can, therefore, be fired in arbitrary order, yielding the final marking [*prdy*, *omt*]. Concurrency is represented in a very natural way in Petri-net models. Since, in the net of Figure 5.3, the labeling function is simply the identity function, upon firing a transition $t$ action $t$ is performed.

In Section 2, process spaces have been introduced as the basic semantic framework of this paper. The definitions given so far in this subsection constitute a process space, namely the process space $(\mathcal{N}(L), L, [\ \rangle, \emptyset)$. Labeled P/T nets represent processes, labels correspond to actions, and the firing rule defines a transition relation; the termination predicate is the empty set. The latter means that the semantics for labeled P/T nets defined by the above process space does not distinguish successful termination and deadlock. This corresponds to the usual semantics for Petri nets (see, for example, [55, 59, 64]). In the next section, a subclass of labeled P/T nets is introduced for modeling object life cycles; the semantics for this class of labeled P/T nets does distinguish successful termination and deadlock.

Since the semantics of labeled P/T nets is captured by means of a process space, we automatically obtain a well-defined notion of equivalence of P/T nets, namely (rooted) branching bisimilarity as defined in Definition 2.8. The fact that rooted branching bisimilarity is also the semantic equivalence in the semantics of the equational theory of Section 3 eases the translation of the inheritance notions of the previous section to the Petri-net framework. In the rich literature on Petri nets, many different semantics appear for many different purposes. In particular, so-called step semantics allow that transitions that are concurrently enabled can fire simultaneously; other semantics define the behavior of Petri nets in terms of partial orders. Step semantics and partial-order semantics are usually meant for studying concurrency and causality. In this paper, the role of concurrency and causality is less important. Therefore, we have chosen the same semantic framework for both the process-algebraic theory of Section 3 and the Petri-net framework of this section. Good starting points for readers interested in concurrency- and causality-related semantics for Petri nets are [19, 61]. A study of concurrency and causality in an ACP-style process-algebraic setting inspired by the treatment of these notions in Petri-net theory can be found in [8].

## 5.3 Analysis of labeled P/T nets

P/T-net models can be analyzed in many different ways. An important class of analysis techniques focuses on properties of such models. The basic idea is that properties of a system or process can be phrased in terms of properties of its P/T-net model. This subsection presents a selection of properties that can be used to analyze labeled P/T nets. This selection is not exhaustive. However, it is sufficient for the remainder of this paper. Note that the labeling function in labeled P/T nets is often only used for modeling purposes. It usually does not affect analysis techniques. Nevertheless, the techniques presented in the remainder of this section are defined for labeled P/T nets, because labeling plays a role in the next section.

Some properties of labeled P/T nets are defined on the structure of a net, whereas other properties concern their behavior. The first property defined in this subsection is a simple structural property which is often convenient in the analysis of P/T nets. The reflexive and transitive closure of a relation $R$ is denoted $R^*$; the inverse of relation $R$ is denoted $R^{-1}$.

**Definition 5.9. (Connectedness)** A labeled P/T net $N = (P, T, F, \ell)$ is *weakly connected*, or simply *connected*, if and only if, for every two nodes $x$ and $y$ in $P \cup T$, $x(F \cup F^{-1})^* y$. Net $N$ is *strongly connected* if and only if, for every two nodes $x$ and $y$ in $P \cup T$, $x F^* y$.

**Example 5.10.** The P/T net in Figure 5.3 is connected, but not strongly connected.

The next definition is fundamental in the analysis of P/T nets. It defines the set of markings that a P/T net can reach from its initial marking. The set of reachable markings of a P/T net is also called the state space of the net. Recall that the semantics of labeled P/T nets is captured in the process space $(\mathcal{N}(L), L, [\,\rangle\,, \emptyset)$. Thus, the reachability relation $\_ \stackrel{*}{\Longrightarrow} \_ \subseteq \mathcal{N}(L) \times \mathcal{N}(L)$ can be defined as in Definition 2.2.

**Definition 5.11. (Reachable markings)** The set of *reachable markings* of a marked, labeled P/T net $(N, s) \in \mathcal{N}(L)$ with $N = (P, T, F, \ell)$, denoted $[N, s\rangle$, is defined as the set $\{s' \in \mathcal{B}(P) \mid (N, s) \stackrel{*}{\Longrightarrow} (N, s')\}$.

Sometimes it is convenient to know the sequence of transitions that are fired in order to reach some given marking. This paper uses the following notations for sequences. Let $A$ be some alphabet of identifiers. A *sequence of length n*, for some natural number $n \in \mathbb{N}$, over alphabet $A$ is a function $\sigma : \{0, \ldots, n-1\} \to A$. The sequence of length zero is called the empty sequence and written $\varepsilon$. For the sake of readability, a sequence of positive length is usually written by juxtaposing the function values: For example, a sequence $\sigma = \{(0, a), (1, a), (2, b)\}$, for $a, b \in A$, is written $aab$. The set of all sequences of arbitrary length over alphabet $A$ is written $A^*$.

**Definition 5.12. (Firing sequence)** Let $(N, s_0)$ with $N = (P, T, F, \ell)$ be a marked, labeled P/T net in $\mathcal{N}(L)$. A sequence $\sigma \in T^*$ is called a *firing sequence* of $(N, s_0)$ if and only if, for some natural number $n \in \mathbb{N}$, there exist markings $s_1, \ldots, s_n \in \mathcal{B}(P)$ and transitions $t_1, \ldots, t_n \in T$ such that $\sigma = t_1 \ldots t_n$ and, for all $i$ with $0 \leq i < n$, $(N, s_i)[t_{i+1}\rangle$ and $s_{i+1} = s_i - \boldsymbol{i}t_{i+1} \uplus \boldsymbol{o}t_{i+1}$. (Note that $n = 0$ implies that $\sigma = \varepsilon$ and that $\varepsilon$ is a firing sequence of $(N, s_0)$.) Sequence $\sigma$ is said to be *enabled* in marking $s_0$, denoted $(N, s_0)[\sigma\rangle$. Firing the sequence $\sigma$ results in the unique marking $s_n$, denoted $(N, s_0)\ [\sigma\rangle\ (N, s_n)$.

The following property is a direct result of the definitions given so far. It states that a marking of a labeled P/T net is reachable if and only if there is a firing sequence leading to that marking.

**Property 5.13.** *Let $(N, s)$ with $N = (P, T, F, \ell)$ be a marked, labeled P/T net in $\mathcal{N}(L)$. For any marking $s' \in \mathcal{B}(P)$, $s' \in [N, s\rangle$ if and only if there exists a firing sequence $\sigma$ of $(N, s)$ such that $(N, s)\ [\sigma\rangle\ (N, s')$.*

**Proof.** Definitions 5.11 (Reachable markings) and 5.12 (Firing sequence). □

**Example 5.14.** It is straightforward to verify that the set of reachable markings of the P/T net in Figure 5.3 equals $\{[cmd, imt], [tcmd, imt], [tprdy, pmt], [prdy, pmt], [tprdy, omt], [prdy, omt]\}$. Marking $[prdy, pmt]$ is the result of firing sequence *rcmd pmat sprdy*.

Property 5.13 is very often used in reasoning about reachable markings of P/T nets. Since it is so fundamental, in the remainder, it is not explicitly referenced each time it is used in a proof.

The following property is often useful when analyzing the set of reachable markings of a P/T net.

**Property 5.15. (Monotonicity of reachable markings)** *Let $(N, s)$ with $N = (P, T, F, \ell)$ be a marked, labeled P/T net in $\mathcal{N}(L)$. Let $s'$ and $s''$ in $\mathcal{B}(P)$ be two markings of $N$. If $s'$ is a reachable marking in $[N, s\rangle$, then marking $s' \uplus s''$ is reachable from $s \uplus s''$; that is, $s' \uplus s'' \in [N, s \uplus s''\rangle$.*

**Proof.** It is a straightforward consequence of the observation that any firing sequence of $(N, s)$ is also a firing sequence of $(N, s \uplus s'')$ that does not affect the tokens in $s''$. □

Example 5.14 shows that it can be straightforward to calculate the set of reachable markings of a P/T net. However, in general, the state space of a non-trivial P/T net can be very large or even infinite.

**Definition 5.16. (Boundedness)** A marked, labeled P/T net $(N, s) \in \mathcal{N}(L)$ is *bounded* if and only if the set of reachable markings $[N, s\rangle$ is finite.

The following property characterizes boundedness; it is particularly useful to prove that a P/T net is unbounded.

**Property 5.17. (Characterization of boundedness)** *A net $(N, s) \in \mathcal{N}(L)$ is bounded if and only if, for any markings $s' \in [N, s\rangle$ and $s'' \in [N, s'\rangle$, $s'' \geq s'$ implies $s'' = s'$.*

**Proof.** See [59, Section 4.2.1.1], where the property is formulated as a property of the coverability tree (called the reachability tree in [59]). The property is a direct result of Property 5.15 (Monotonicity of reachable markings) and the fact that the coverability tree of a labeled P/T net is always finite. □

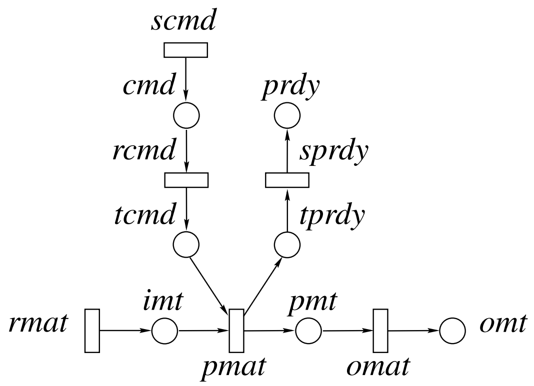



Figure 5.18: An example of a live and unbounded P/T net.

**Example 5.19.** The P/T net of Figure 5.3 is bounded. Example 5.14 shows that its state space is finite. Figure 5.18 shows a variant of the P/T net of Figure 5.3. It contains two new transitions, modeling an operator that sends commands to the production unit (*scmd*) and replenishes input material (*rmat*). The initial marking is the empty bag **0**. As before, it is assumed that the labeling function is the identity function. It is not difficult to see that the net of Figure 5.18 is unbounded. Since the two abovementioned transitions have no input places, they are continuously enabled. Therefore, the number of tokens in places *cmd* and *imt*, and thus all other places, can increase indefinitely. Property 5.17 can be used to prove unboundedness formally. Firing transition *scmd* leads to marking [*cmd*], which is strictly larger than the initial marking **0**.

Another property of a P/T-net model, which is often meaningful from a design point of view, is that it contains no so-called dead transitions. A dead transition is a transition that is never able to fire.

**Definition 5.20. (Dead transition)** Let $(N, s)$ be a marked, labeled P/T net in $\mathcal{N}(L)$. A transition $t \in T$ is *dead* in $(N, s)$ if and only if there is no reachable marking $s' \in [N, s\rangle$ such that $(N, s')[t\rangle$.

A property stronger than the absence of dead transitions is liveness. A P/T net is live if and only if, no matter what marking has been reached, it is always possible to enable an *arbitrary* transition of the net by firing any number of other transitions.

**Definition 5.21. (Liveness)** A marked, labeled P/T net $(N, s) \in \mathcal{N}(L)$ with $N = (P, T, F, \ell)$ is *live* if and only if, for every reachable marking $s' \in [N, s\rangle$ and transition $t \in T$, there is a reachable marking $s'' \in [N, s'\rangle$ such that $(N, s'')[t\rangle$.

**Property 5.22.** *A live marked, labeled P/T net does not have any dead transitions.*

**Proof.** Definitions 5.20 and 5.21. □

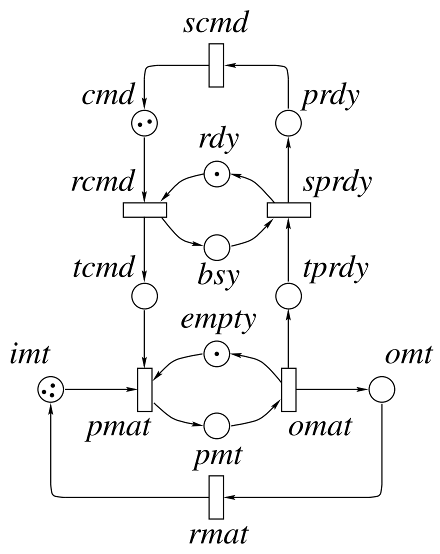



Figure 5.23: An example of a live and bounded P/T net.

**Example 5.24.** The P/T net of Figure 5.18 is live. Figure 5.23 shows another example of a live P/T net. It is a variant of the production unit of Figure 5.18. It contains three extra places. A token in place *rdy* means that the unit is ready to receive a command; a token in *bsy* means that it is busy processing; a token in *empty* models that the unit contains no material. Another difference with the unit of Figure 5.18 is that the processing-ready signal is not sent immediately after processing, but only after the output material has been delivered. A final difference concerns the commands and the input material for the unit. The operator only sends a new command to the unit after a processing-ready signal has been received. Material is only replenished after output material has been delivered. Initially, two commands have been sent to the unit and three pieces of input material are available.

It is possible to prove that the two P/T nets of Figures 5.18 and 5.23 are live by analyzing their state spaces. From such an analysis, it also follows that the net of Figure 5.23 is bounded, whereas we have already seen that the net of Figure 5.18 is unbounded.

The P/T net of Figure 5.3 is not live. Again, this follows easily from the state space (see Example 5.14).

A marking of a marked P/T net is called a home marking if and only if it is reachable from every marking reachable from the initial marking. Home markings play an important role in the analysis of termination behavior and iterative behavior. Consider a marked P/T net that is defined to terminate successfully when some given marking is reached. If this marking is a home marking, it means that the P/T net always has the option to terminate successfully. If the initial marking is a home marking of the net, then its behavior is iterative.

**Definition 5.25. (Home marking)** Let $(N, s)$ be a marked, labeled P/T net in $\mathcal{N}(L)$. A reachable marking $s' \in [N, s\rangle$ is a *home marking* of $(N, s)$ if and only if, for any reachable marking $s'' \in [N, s\rangle$, $s' \in [N, s''\rangle$.

**Example 5.26.** The initial marking of the P/T net of Figure 5.23 is a home marking.

The following definition defines the notion of a *subnet* of a P/T net.

**Definition 5.27. (Subnet)** Let $N = (P, T, F, \ell)$ be an $L$-labeled P/T net. A P/T net $S = (P_0, T_0, F_0, \ell_0)$ with $P_0 \subseteq P$ and $T_0 \subseteq T$ is a *subnet* of $N$ if and only if $F_0 = F \cap ((P_0 \times T_0) \cup (T_0 \times P_0))$ and $\ell_0 = \ell \cap (T_0 \times L)$. Subnet $S$ is said to be generated by the set of nodes $P_0 \cup T_0$.

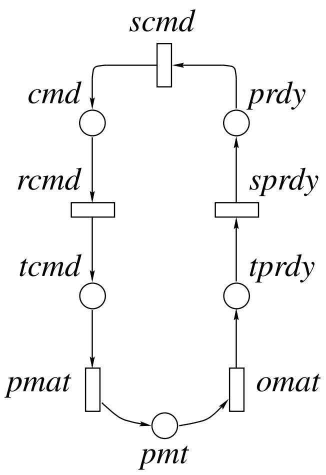



Figure 5.28: A subnet of the P/T net of Figure 5.23.

**Example 5.29.** Consider the P/T net of Figure 5.23. Figure 5.28 shows its subnet generated by the set of nodes $\{cmd, rcmd, tcmd, pmat, pmt, omat, tprdy, sprdy, prdy, scmd\}$.

The following definition defines a very specific kind of subnet.

**Definition 5.30. (S-component)** Let $N = (P, T, F, \ell)$ be an $L$-labeled P/T net. Subnet $S = (P_0, T_0, F_0, \ell_0)$ of $N$ with $P_0 \neq \emptyset$ is an *S-component* of $N$ if and only if *i)* $S$ is strongly connected (see Definition 5.9), *ii)* for every transition $t \in T_0$, $|\boldsymbol{i}_S t| = |\boldsymbol{o}_S t| = 1$, and *iii)* for every place $p \in P_0$, $\boldsymbol{i}_N p \cup \boldsymbol{o}_N p \subseteq T_0$.

**Example 5.31.** The P/T net of Figure 5.28 is an S-component of the P/T net of Figure 5.23.

A characteristic property of an S-component of some P/T net is that the number of tokens in the places of the S-component is constant for all reachable markings of the P/T net. Recall that, for some bag $Y$ over alphabet $A$ and element $a \in A$, $Y(a)$ denotes the number of occurrences of $a$ in $Y$. For any $X \subseteq A$, $Y(X)$ denotes the total number of occurrences of elements of $X$ in $Y$; that is, $Y(X) = (+a : a \in X : Y(a))$.

**Property 5.32.** *Let $(N, s)$ be a marked, labeled P/T net in $\mathcal{N}(L)$; let $(P, T, F, \ell)$ be an S-component of $N$. For any reachable marking $s' \in [N, s\rangle$, $s'(P) = s(P)$.*

**Proof.** Induction on the length of any firing sequence needed to reach marking $s'$ from marking $s$. □

**Example 5.33.** Consider again the P/T net of Figure 5.23 and its S-component of Figure 5.28. It is not difficult to see that the number of tokens in the S-component is two for every reachable marking.

Property 5.32 has many interesting consequences. The following property states that the marking of a live connected P/T net marks all of its S-components.

**Property 5.34.** *Let $(N, s)$ with $N = (P, T, F, \ell)$ be a live marked, connected P/T net in $\mathcal{N}(L)$ such that $T \neq \emptyset$. For every S-component $(P_0, T_0, F_0, \ell_0)$ of $N$, $s(P_0) > 0$.*

**Proof.** If $N$ has no places the property is trivial. Therefore, assume that $P \neq \emptyset$. Let $(P_0, T_0, F_0, \ell_0)$ be an S-component of $N$ such that $P_0 \neq \emptyset$. Assume that $s(P_0) = 0$. It follows from the assumptions that $N$ is connected and $T \neq \emptyset$ and Definitions 5.9 (Connectedness) and 5.30 (S-component) that $T_0 \neq \emptyset$. Property 5.32 implies that any of the places in $P_0$ is unmarked in any marking reachable from $s$. Thus, it follows from Definitions 5.20 (Dead transition) and again 5.30 (S-component) that all transitions in $T_0$ are dead in $(N, s)$. However, this contradicts Property 5.22, which states that a live P/T net cannot have dead transitions. Hence, $s(P_0) > 0$. □

This subsection ends with two properties that a set of places of a labeled P/T net may exhibit. A set of places $X$ of a P/T net is called a trap if every transition that needs a token from $X$ to fire also returns a token to $X$. The preset and postset of a *set* of nodes $X$ of a P/T net are defined as follows: $\boldsymbol{i}X = (\cup x : x \in X : \boldsymbol{i}x)$ and $\boldsymbol{o}X = (\cup x : x \in X : \boldsymbol{o}x)$.

**Definition 5.35. (Trap)** Let $N = (P, T, F, \ell)$ be a labeled P/T net. A set of places $X \subseteq P$ is called a *trap* if and only if $\boldsymbol{o}X \subseteq \boldsymbol{i}X$. A trap is *proper* if and only if it is not the empty set.

A characteristic property of a trap is that, once it becomes marked, it remains marked.

**Property 5.36.** *Let $(N, s)$ be a labeled P/T net with $N = (P, T, F, \ell)$. Let $X \subseteq P$ be a trap of $N$. If $s(X) > 0$, then, for every reachable marking $s' \in [N, s\rangle$, $s'(X) > 0$.*

**Proof.** Definitions 5.35 (Trap), 5.11 (Reachable markings), and 5.7 (Firing rule). □

**Example 5.37.** Consider again the P/T net of Figure 5.23. The following sets of places are traps: {*rdy*, *bsy*}, {*empty*, *pmt*}, {*cmd*, *bsy*, *prdy*}, {*rdy*, *tcmd*, *pmt*, *tprdy*}, {*cmd*, *tcmd*, *pmt*, *tprdy*, *prdy*}, and {*imt*, *pmt*, *omt*}. By definition, the union of any number of traps is again a trap. The P/T net of Figure 5.23 has no other proper traps than the six mentioned above and the ones that consist of the union of any number of these six traps.

Note that the places of an S-component of a P/T net constitute a trap. (It is an interesting exercise to prove this formally.)

A set of places $X$ of a labeled P/T net is called a siphon if and only if every transition that puts a token in $X$ upon firing also consumes a token from $X$.

**Definition 5.38. (Siphon)** Let $N = (P, T, F, \ell)$ be a labeled P/T net. A set of places $X \subseteq P$ is called a *siphon* if and only if $\boldsymbol{i}X \subseteq \boldsymbol{o}X$. A siphon is *proper* if and only if it is not the empty set.

A characteristic property of a siphon is that, once it becomes unmarked, it always remains unmarked.

**Property 5.39.** *Let $(N, s)$ be a labeled P/T net with $N = (P, T, F, \ell)$. Let $X \subseteq P$ be a siphon of $N$. If $s(X) = 0$, then, for every reachable marking $s' \in [N, s\rangle$, $s'(X) = 0$.*

**Proof.** Definitions 5.38 (Siphon), 5.11 (Reachable markings), and 5.7 (Firing rule). □

**Example 5.40.** Consider again Figure 5.23. In this particular example, the set of siphons of the P/T net is identical to its set of traps given in Example 5.37.

Note that the places of an S-component of a P/T net form a siphon of the net.

In this subsection, a non-exhaustive selection of properties of labeled P/T nets has been presented. However, some important aspects have not yet been addressed. Is it always possible to decide whether some marking of a labeled P/T net is reachable from its initial marking? Or whether it is a home marking? And is it always possible to decide whether some given labeled P/T net is bounded or live? The answer to all these questions is affirmative. However, the algorithms are very complex and inefficient. The reader interested in these algorithms, and in decidability and complexity results for P/T nets in general, is referred to the literature. Good starting points are [31, 32]. A solution to improve the efficiency of analysis techniques for P/T nets is to consider *subclasses* of P/T nets. As mentioned, the class of so-called free-choice P/T nets is particularly interesting in this context.

## 5.4 Free-choice P/T nets

Free-choice P/T nets are characterized by the fact that two transitions sharing an input place always share all their input places. The class of free-choice P/T nets combines a reasonable expressive power with strong analysis techniques. Consequently, free-choice P/T nets have been studied extensively in the literature. The most important results on free-choice P/T nets have been brought together in [27]. Except for Property 5.54, all results in this subsection appear in [27]. The main theorems are given without proof. Some other results are accompanied with proofs, because the proofs illustrate the use of the main theorems. The results in this subsection show that traps, siphons, and S-components play an important role in analyzing liveness, boundedness, and home markings of free-choice P/T nets.

**Definition 5.41. (Free-choice P/T net)** A *free-choice* P/T net is a P/T net $(P, T, F, \ell)$ as in Definition 5.1 such that, for all transitions $t, u \in T$, either $\boldsymbol{it} \cap \boldsymbol{iu} = \emptyset$ or $\boldsymbol{it} = \boldsymbol{iu}$.

Note that, in Definition 5.41, free-choice P/T nets are labeled. As mentioned, the labeling function is included for modeling purposes. It is not present in the standard definition of free-choice P/T nets as given in, for example, [27]. The labeling function does not affect any of the results presented in the remainder of this section.

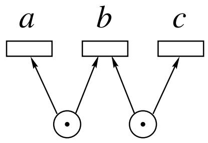



Figure 5.42: A non-free-choice construct: confusion.

**Example 5.43.** The P/T nets of Figures 5.3, 5.18, 5.23, and 5.28 are all free-choice. Figure 5.42 shows a typical non-free-choice construct, called confusion. In the initial state, all three transitions $a$, $b$, and $c$ are enabled. Transitions $a$ and $c$ are independent; they do not share any input places. However, both transitions compete for a token with transition $b$ and are thus in conflict with $b$. The construct is called confusion, because firing, for example, transition $a$ solves the conflict between the other two transitions $b$ and $c$ in favor of $c$. After $a$ has fired only $c$ is enabled.

The first main theorem of this subsection states that a connected free-choice P/T net is live if and only if every proper siphon includes an initially marked trap.

**Theorem 5.44. (Commoner's theorem)** *Let $(N, s)$ with $N = (P, T, F, \ell)$ be a marked, connected free-choice P/T net in $\mathcal{N}(L)$ such that $T$ is not empty. Net $(N, s)$ is live if and only if every proper siphon $X \subseteq P$ contains a trap $Y \subseteq X$ such that $s(Y) > 0$.*

**Proof.** [27, Section 4.3] □

**Example 5.45.** Consider the free-choice P/T net of Figure 5.18. It is not difficult to see that this P/T net does not have any siphons. Hence, it trivially satisfies the condition of Theorem 5.44, which means it is live. Figure 5.23 shows another free-choice P/T net. Its traps and siphons are given in Examples 5.37 and 5.40. It follows immediately that the net satisfies the condition of Commoner's theorem, which means that also this net is live. Finally, consider the free-choice P/T net of Figure 5.3. The set consisting of the single place *cmd* is a siphon. Since it is not a trap, the condition of Commoner's theorem is not satisfied. As a result, the P/T net of Figure 5.3 is not live. Note that the results of this example conform to the conclusions drawn earlier in Example 5.24.

Theorem 5.44 (Commoner's theorem) can be used to prove the following interesting property of live free-choice P/T nets. It says that a live free-choice P/T net remains live when an arbitrary number of tokens is added to its marking. The basic idea of the proof is that adding tokens to a marking cannot invalidate the requirements of Commoner's theorem.

**Property 5.46. (Monotonicity of liveness)** *Let $(N, s)$ with $N = (P, T, F, \ell)$ be a live marked, free-choice P/T net in $\mathcal{N}(L)$; let $s' \in \mathcal{B}(P)$ be a marking of $N$. The marked, free-choice P/T net $(N, s \uplus s')$ is live.*

**Proof.** Since Commoner's theorem is only valid for *connected* free-choice P/T nets, $N$ is partitioned into connected subnets without any connections between them. Formally, let $N_0 = (P_0, T_0, F_0, \ell_0), \ldots, N_n = (P_n, T_n, F_n, \ell_n)$, for some $n \in \mathbb{N}$, be *connected* subnets of $N$ such that, for all $i$ with $0 \leq i \leq n$, $N_i$ is generated by $P_i \cup T_i$. Furthermore, assume that $P_0$ through $P_n$, $T_0$ through $T_n$, and $F_0$ through $F_n$ are partitionings of $P$, $T$, and $F$, respectively. Clearly, it follows from Definition 5.21 (Liveness) that, for any marking $s \in \mathcal{B}(P)$, $(N, s)$ is live if and only if, for all $i$ with $0 \leq i \leq n$, $(N_i, s \upharpoonright P_i)$ is live.

Let $N_i$, for some $i$ with $0 \leq i \leq n$, be an arbitrary subnet of $N$ as defined above. To prove Property 5.46, it suffices to show that, for arbitrary markings $s, s' \in \mathcal{B}(P_i)$, $(N_i, s \uplus s')$ is live from the assumption that $(N_i, s)$ is live. If $T_i = \emptyset$, $(N_i, s \uplus s')$ is live. Therefore, assume that $T_i \neq \emptyset$. Note that, since $N$ is free-choice, also $N_i$ is free-choice. Thus, it follows from the assumption that $(N_i, s)$ is live and Theorem 5.44 (Commoner's theorem) that every proper siphon $X \subseteq P_i$ of $N_i$ contains a trap $Y \subseteq X$ such that $s(Y) > 0$. Consequently, every proper siphon $X \subseteq P_i$ contains a trap $Y \subseteq X$ such that $(s \uplus s')(Y) > 0$. Again Theorem 5.44 yields that $(N_i, s \uplus s')$ is live. $\square$

Note that liveness is not monotone for ordinary labeled P/T nets. It is a nice exercise to find a live P/T net that is no longer live when one or more tokens are added to the initial marking. For the impatient reader, Figure 7.26 in Section 7.6 contains an example of such a net.

Recall Property 5.34 which states that the marking of a live connected P/T net marks all of its S-components. It can be strengthened for a certain class of free-choice P/T nets.

**Property 5.47.** *Let $(N, s)$ with $N = (P, T, F, \ell)$ be a live and bounded marked, connected free-choice P/T net such that $T$ is not empty. Let $s' \in \mathcal{B}(P)$ be a marking of $N$. The marked P/T net $(N, s')$ is live if and only if, for every S-component $(P_0, T_0, F_0, \ell_0)$ of $N$, $s'(P_0) > 0$. That is, $(N, s')$ is live if and only if $s'$ marks every S-component of $N$.*

**Proof.** The implication from left to right follows immediately from Property 5.34. The other implication is a consequence of Theorem 5.44 (Commoner's theorem) and several results concerning siphons and traps, see [27, Section 5.2] for details. $\square$

An S-cover of a labeled P/T net is a set of S-components such that each place of the net is contained by at least one S-component.

**Definition 5.48. (S-cover)** Let $N = (P, T, F, \ell)$ be an $L$-labeled P/T net; let $C$ be a set of S-components of $N$. Set $C$ is called an *S-cover* of $N$ if and only if, for every place $p \in P$, there exists an S-component $(P_0, T_0, F_0, \ell_0)$ in $C$ such that $p \in P_0$.

The second important theorem of this subsection is the S-coverability theorem.

**Theorem 5.49. (S-coverability theorem)** *Let $(N, s)$ be a live and bounded marked, free-choice P/T net. Net $N$ has an S-cover.*

**Proof.** [27, Section 5.1] □

**Example 5.50.** In Example 5.24, it was argued that the P/T net of Figure 5.23 is live and bounded. Since this P/T net is free-choice, it must have an S-cover. It is not difficult to verify that the four subnets generated by the sets of nodes $\{cmd, rcmd, tcmd, pmat, pmt, omat, tprdy, sprdy, prdy, scmd\}$, $\{rdy, rcmd, bsy, sprdy\}$, $\{empty, pmat, pmt, omat\}$, and $\{imt, pmat, pmt, omat, omt, rmat\}$ are S-components that form an S-cover. The first one of these S-components is the one shown in Figure 5.28. Note that the marking in Figure 5.23 marks all S-components, which conforms to Property 5.47.

The S-coverability theorem can be used to prove the following property of live and bounded free-choice P/T nets.

**Property 5.51.** *Let $(N, s)$ with $N = (P, T, F, \ell)$ be a live and bounded marked, free-choice P/T net in $\mathcal{N}(L)$. For every marking $s' \in \mathcal{B}(P)$, net $(N, s')$ is bounded.*

**Proof.** According to Definition 5.16 (Boundedness), it must be shown that the set of reachable markings $[N, s'\rangle$ is finite. First, note that the number of tokens in marking $s'$, $s'(P)$, is finite. Second, it follows from Theorem 5.49 (S-coverability theorem) that $N$ has an S-cover $C$. Since the number of places of a P/T net is finite, it follows from Definition 5.48 (S-cover) that the number of S-components in $C$, $|C|$, is also finite. Since every S-component initially contains at most $s'(P)$ tokens, it follows from Property 5.32 that every reachable marking $s'' \in [N, s'\rangle$ contains at most $s'(P) \cdot |C|$ tokens; that is, for every $s'' \in [N, s'\rangle$, $s''(P) \leq s'(P) \cdot |C|$. Since the number of places of $N$ is finite, it follows that $[N, s'\rangle$ is finite. □

The third and final theorem of this subsection states that a reachable marking of a live and bounded connected free-choice P/T net is a home marking if and only if it marks every proper trap of the net.

**Theorem 5.52. (Home-marking theorem)** *Let $(N, s)$ with $N = (P, T, F, \ell)$ be a live and bounded marked, connected free-choice P/T net in $\mathcal{N}(L)$ such that $T$ is not empty. A reachable marking $s' \in [N, s\rangle$ is a home marking of $(N, s)$ if and only if, for every proper trap $X \subseteq P$, $s'(X) > 0$.*

**Proof.** [27, Section 8.2] □

**Example 5.53.** Consider again the P/T net of Figure 5.23 and its traps given in Example 5.37. Since the initial marking marks every proper trap of the net, it is a home marking, which conforms to the claim of Example 5.26.

The Home-marking theorem can be used to prove monotonicity of home markings for live and bounded free-choice P/T nets.

**Property 5.54. (Monotonicity of home markings)** *Let $(N, s)$ with $N = (P, T, F, \ell)$ be a live and bounded marked, free-choice P/T net in $\mathcal{N}(L)$. Let $s'$ and $s''$ in $\mathcal{B}(P)$ be two markings of $N$. If $s'$ is a home marking of $(N, s)$, then $s' \uplus s''$ is a home marking of $(N, s \uplus s'')$.*

**Proof.** If $N$ has no transitions, the property is trivial. Therefore assume that $T$ is not empty. Without loss of generality, it may also be assumed that $N$ is connected. (If $N$ is not connected, it is possible to consider the partitioning of connected subnets of $N$, similar to the proof of Property 5.46 (Monotonicity of liveness).)

Assume that $s'$ is a home marking of $(N, s)$. The goal is to use the Home-marking theorem to prove that $s' \uplus s''$ is a home marking of $(N, s \uplus s'')$. Thus, it must be shown that $(N, s \uplus s'')$ is live and bounded, that $s' \uplus s'' \in [N, s \uplus s''\rangle$, and that $s' \uplus s''$ marks every proper trap of $N$.

Since $(N, s)$ is live and bounded, Properties 5.46 and 5.51 yield that $(N, s \uplus s'')$ is live and bounded. Since $s'$ is a home marking of $(N, s)$, Definition 5.25 (Home marking) and Property 5.15 (Monotonicity of reachable markings) imply that $s' \uplus s'' \in [N, s \uplus s''\rangle$. It follows from the Home-marking theorem that $s'$ marks every proper trap of $N$ and, thus, that $s' \uplus s''$ marks every proper trap of $N$. As a result, the conditions of the Home-marking theorem are satisfied proving that $s' \uplus s''$ is a home marking of $(N, s \uplus s'')$. □

Similar to Property 5.46 (Monotonicity of liveness), Property 5.54 does not generalize to ordinary P/T nets as defined in Definition 5.1. Figure 7.26 in Section 7.6 shows a live and bounded P/T net with a home marking that does not satisfy the monotonicity requirement stated in Property 5.54.

The three main theorems of this subsection, Commoner's theorem, the S-coverability theorem, and the home-marking theorem, form the basis for several efficient analysis techniques for free-choice P/T nets. None of the results generalizes to ordinary P/T nets, although some of them (partially) generalize to subclasses of P/T nets that are slightly larger than free-choice P/T nets. The interested reader is referred to [27, Chapter 10].

# 6 Inheritance in the Petri-net Framework

The main goal of this and the following section is to translate the results of Section 4 to a framework that is close to practical object-oriented methods such as UML. Petri nets are well suited for this purpose for several reasons. First, they provide a graphical description technique that is easy to understand. Second, Petri nets have an explicit representation of states. Third, it is natural to model concurrency in Petri nets, which is an advantage when modeling distributed systems. Fourth, they have a sound theoretical basis and many techniques and tools are available for the analysis of Petri nets. Finally, they are close to the state-based graphical techniques used in practical object-oriented methods for specifying object life cycles, such as the statechart diagrams of UML.

Section 6.1 formalizes the notion of an object life cycle in the framework of Petri nets. In Section 6.2, the four inheritance relations defined in Section 4.1 are translated to this framework. Section 7 presents several transformation rules on object life cycles. These transformation rules are based on the axioms of inheritance of Section 4.2. They can be used to transform a class into a subclass, thus, reusing life-cycle designs.

The Petri-net framework of this and the following section is more expressive and more powerful than the process-algebraic framework of Section 4. The price to be paid is that the definitions, the theorems, and the proofs are more complex. The translation from process algebra to Petri nets shows how the development of a concept in one formalism can inspire the development in another formalism. The formalization of inheritance of behavior in process algebra has led to a clear conceptual understanding; the study in Petri nets yields a framework close to practical object-oriented methods.

## 6.1 Object life cycles

An object life cycle specifies the order in which the methods of an object may be executed. When modeling a life cycle with a Petri net, a transition firing corresponds to the execution of a method. Since the emphasis is on the execution order of methods and not on their implementation details, the formalism of labeled P/T nets as introduced in the previous section is well suited as the basic Petri-net framework for modeling life cycles and studying inheritance of behavior. Transition labels correspond to method identifiers. At this point, it is clear why transition labeling is included in our P/T-net framework. It is necessary, because a single method may occur several times in the life cycle of an object. As in Section 4, the set of methods is denoted $M$. Recall that $M$ includes the identifier $\tau$ to model internal methods. The set of external methods $M\backslash\{\tau\}$ is denoted $E$. However, not every $M$-labeled P/T net is an object life cycle.

It is important to see that a life cycle refers to a *single object*. It suffices to consider just one object, because multiple objects of the same class interact via the execution of methods and not directly via the life cycle. A P/T net defining a life cycle has exactly one *initial* or *input* place $i$. Place $i$ has no input transitions. A token in place $i$ models the fact that the corresponding object has not yet been created. When considering the behavior of an object as it is specified by a P/T net with an input place $i$, in the initial marking, place $i$ is the only place marked and it contains only a single token.

To model object termination, an object life cycle has a unique *final* or *output* place $o$. An object terminates when, and only when, it reaches the marking of a single token in $o$. In addition, if a marking has a token in $o$, it must be the only token in the marking. This means that, upon termination of an object, all information about the object is removed. Furthermore, it is assumed that it is always possible to terminate. However, this does not mean that an object is *forced* to terminate. In technical terms, marking $[o]$ must be a home marking of the P/T net modeling an object life cycle, as defined in Definition 5.25.

In addition to the above requirements, a P/T net modeling a life cycle is assumed to be connected, as defined in Definition 5.9. Furthermore, a life cycle may not have any dead transitions, as defined in Definition 5.20. These requirements are technically convenient. They are also meaningful from a design point of view. Places in a part of the P/T net that is not connected to the initial place $i$ will remain unmarked when starting from the initial marking $[i]$, no matter what transitions are fired. Dead transitions correspond to methods that cannot be executed. Hence, it is not meaningful to model a life cycle with an unconnected P/T net or a P/T net with dead transitions.

The above considerations are partly related to the static structure of an object life cycle and partly to its behavior, in particular, its *termination* behavior. Therefore, it is important to fix the semantic framework for $M$-labeled P/T nets that is used throughout the remainder. Assume that the universe of identifiers $U$ contains the special identifiers $i$ and $o$.

**Definition 6.1. (Semantics of $M$-labeled P/T nets)** The semantics of marked, $M$-labeled P/T nets is defined by the process space $(\mathcal{N}(M), M, [\,\rangle, \downarrow)$. The set of processes $\mathcal{N}(M)$ is the set of all marked, $M$-labeled P/T nets as defined in Definition 5.4. The transition relation $[\,\rangle$ is the firing rule of Definition 5.7. Predicate $\downarrow$ is the set of all marked nets $(N, [o])$ in $\mathcal{N}(M)$ with $N = (P, T, F, \ell)$ and $o \in P$, where $o$ is the special output place in $U$ introduced above.

As before, the consequence of defining the semantics of $M$-labeled P/T nets in terms of a process space is that we obtain a notion of equivalence of P/T nets, namely (rooted) branching bisimilarity. As explained in Section 2.2, the root condition in the definition of rooted branching bisimilarity is crucial in a process-algebraic context. However, it is not needed in a framework of P/T nets. On the contrary, it is often tedious to have to take into account the root condition. Therefore, in the remainder, branching bisimilarity is chosen as the basic equivalence. However, all the results in this section carry over to rooted branching bisimilarity.

Having defined the basic semantic framework, it is possible to formally define the notion of an object life cycle. The definition uses Definition 5.11 that defines the set of reachable markings of a P/T net. Note that the behavioral properties in the definition are all defined with respect to the marking consisting of a single token in the special place $i$. In the remainder, we often implicitly assume that the initial marking of an object life cycle equals $[i]$.

**Definition 6.2. (Object life cycle)** Let $N = (P, T, F, \ell)$ be an $M$-labeled P/T net. Net $N$ is an *object life cycle* if and only if the following conditions are satisfied:

*i)* *Connectedness*: $N$ is weakly connected;
*ii)* *Object creation*: $P$ contains an input place $i \in U$ such that $\bullet i = \emptyset$;
*iii)* *Object termination*: $P$ contains an output place $o \in U$ such that $o\bullet = \emptyset$;
*iv)* *Proper termination*: for any reachable marking $s \in [N, [i]\rangle$, if $o \in s$, then $s = [o]$;
*v)* *Termination option*: marking $[o]$ is a home marking of $(N, [i])$;
*vi)* *Dead transitions*: $(N, [i])$ contains no dead transitions.

The set of all object life cycles is denoted $\mathcal{O}$.

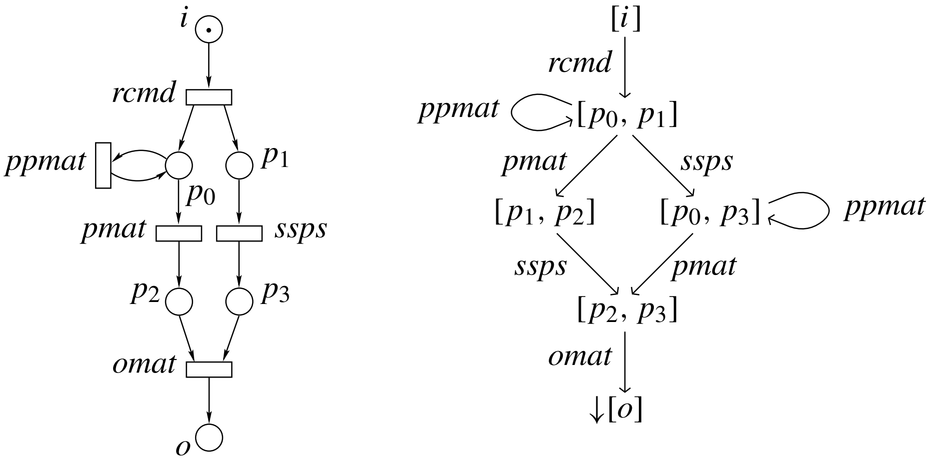



Figure 6.3: An example of an object life cycle.

**Example 6.4.** Figure 6.3 shows an example of a labeled P/T net modeling the object life cycle of a production unit and its semantics. The firing rule is depicted by arrows. The states in the semantics are represented by the markings of the net. The life cycle shows concurrent as well as iterative behavior. After receiving a command, the unit starts its processing phase. In parallel, it sends a start-processing signal to the operator. The processing phase consists of zero or more preprocessing steps, followed by the main processing action. The unit finishes with delivering the output material. It is clear that the object life cycle of Figure 6.3 satisfies the requirements of Definition 6.2. In particular, it has always the option to terminate and termination is always proper.

Definition 6.2 of an object life cycle in the framework of P/T nets is more involved than the corresponding definition in the equational theory $\mathrm{PA}^{\tau}(M)$ (Definition 4.1). The main reason for this difference is that the theory $\mathrm{PA}^{\tau}(M)$ does not contain the inaction constant nor does it allow the specification of iterative behavior. A consequence of these limitations is that the termination of an object life cycle specified in $\mathrm{PA}^{\tau}(M)$ is always guaranteed. Labeled P/T nets, on the other hand, inherently allow iterations, as the above example shows. It is also not difficult to give P/T nets which will never terminate due to a deadlock. In other words, labeled P/T nets are more expressive than closed terms over the signature of the equational theory $\mathrm{PA}^{\tau}(M)$. One of the consequences is that it is necessary to explicitly define the termination requirements in the definition of an object life cycle above.

An interesting question is what properties object life cycles exhibit. An example of a useful property is the following.

**Property 6.5.** *Let $N$ be an object life cycle in $\mathcal{O}$. The marked life cycle $(N, [i])$ is bounded.*

**Proof.** Assume $(N, [i])$ is unbounded. According to Property 5.17 (Characterization of boundedness), there exist markings $s' \in [N, [i]\rangle$ and $s'' \in [N, s'\rangle$ such that $s'' > s'$. It follows from Requirement *v*) (Termination option) of Definition 6.2 (Object life cycle) that $[o] \in [N, s'\rangle$. Property 5.15 (Monotonicity of reachable markings) yields that $[o] \uplus (s'' - s') \in [N, s''\rangle$. Since $s'' - s'$ is not empty, the latter contradicts Requirement *iv*) (Proper termination) of Definition 6.2 (Object life cycle). Hence, $(N, [i])$ is bounded. □

This property is one step towards a very interesting result, namely that object life cycles can be characterized in terms of liveness and boundedness. The characterization is taken from [2], where it is given for so-called sound workflow nets. Sound workflow nets are almost identical to object life cycles. The following auxiliary definition is needed to formulate the desired theorem. Given a net with an input place and an output place as in Definition 6.2 (Object life cycle), it defines the extension of the net with an extra transition connecting the output place to the input place.

**Definition 6.6.** Let $N = (P, T, F, \ell)$ be an $M$-labeled P/T net satisfying the first three requirements of Definition 6.2. Assume that $\bar{t}$ is an identifier in $U$ that does not appear in $P$ or $T$. The labeled P/T net $\bar{N} = (\bar{P}, \bar{T}, \bar{F}, \bar{\ell})$ is defined as follows: $\bar{P} = P$, $\bar{T} = T \cup \{\bar{t}\}$, $\bar{F} = F \cup \{(o, \bar{t}), (\bar{t}, i)\}$, and $\bar{\ell} = \ell \cup \{(\bar{t}, \tau)\}$.

The label of the new transition in the extended net does not play a role in the characterization of object life cycles. For the sake of convenience, the label is set to $\tau$.

**Theorem 6.7. (Characterization of object life cycles)** *Let $N$ be an $M$-labeled P/T net satisfying Requirements i) through iii) of Definition 6.2 (Object life cycle). Net $N$ is an object life cycle if and only if $(\bar{N}, [i])$ is live and bounded.*

**Proof.** The proof is identical to the proof of Theorem 11 in [2], which characterizes soundness of workflow nets in terms of liveness and boundedness. There are two small differences between the two theorems. First, object life cycles are labeled nets whereas workflow nets are unlabeled. However, labeling does not play a role in the proof. Second, the definition of workflow nets requires that the extension with an extra transition connecting the output place to the input place is strongly connected. However, this extra requirement is not needed in the proof. □

Another interesting question is whether it is possible to decide efficiently whether a labeled P/T net is an object life cycle.

**Theorem 6.8. (Decidability of object-life-cycle properties)** *It is decidable whether an $M$-labeled P/T net is an object life cycle.*

**Proof.** Requirements *i*) through *iii*) of Definition 6.2 are simple structural properties that can be checked for any P/T net in a straightforward way. Requirements *iv*), *v*), and *vi*) can be derived from the coverability tree of $(N, [i])$ (see [59], where it is called the reachability tree). Since the coverability tree of a P/T net is always finite, it is decidable whether $N$ is an object life cycle. □

The exact complexity of deciding the object-life-cycle properties is still an open question. However, some known complexity results for P/T nets (see, for example, [31, 32]) suggest that they cannot be decided efficiently. First, deciding the life-cycle properties by means of the coverability tree, as suggested in the proof of Theorem 6.8, requires in the worst case non-primitive recursive space. Second, Requirements *iv*), *v*), and *vi*) of Definition 6.2 are all closely related to the question whether a marking is reachable from some other marking. This question is decidable, but the exact complexity is unknown. It is only known that the space requirements of an algorithm to decide reachability are at least exponential in the number of nodes of the net, which is not particularly efficient. The question whether some given marking is a home marking is decidable, but the complexity is unknown. Third, Theorem 6.7 shows that object life cycles can be characterized in terms of liveness and boundedness. Deciding boundedness for labeled P/T nets is known to be EXPSPACE-hard. The complexity of deciding liveness is unknown. It is unlikely that deciding the combination of liveness and boundedness requires less than exponential space.

Although the complexity of deciding the life-cycle properties is discouraging, it is still possible to verify the properties by means of the coverability tree in a relatively straightforward way for object life cycles that are not too large. In addition, [46] gives conditions for a marked P/T net that guarantee that the net is live and bounded. It also presents an algorithm which is linear in the number of nodes of the net to verify these conditions. Based on Theorem 6.7, the algorithm of [46] provides an efficient procedure to prove that a P/T net is an object life cycle, although failure of the algorithm does not mean that the net is not a life cycle. Experience with case studies is needed to find out the practical limitations. In case the P/T nets in an object-oriented design become too complex for the automatic verification of life-cycle properties or if the algorithm of [46] fails, there are two solutions. First, it is possible to build life cycles in a constructive way by means of transformation rules preserving the life-cycle properties. Second, it is possible to restrict the class of P/T nets used in the design in such a way that the verification of the life-cycle properties becomes more efficient. Both solutions are discussed briefly in the remainder of this subsection.

The characterization of object life cycles in terms of liveness and boundedness of Theorem 6.7 already provides the basis for a useful set of transformation rules preserving life-cycle properties. The theorem implies that transformation rules preserving liveness and boundedness of P/T nets also preserve life-cycle properties. Starting with an object life cycle, the application of any such transformation rule yields another object life cycle. Using this approach, time-consuming verification of life-cycle properties is not necessary. The literature contains several studies on transformation rules preserving liveness and boundedness of P/T nets. In [2], a set of transformation rules is given that is designed for the purpose of constructing sound workflow nets. They cover design constructs such as the addition of sequential, alternative, parallel, and iterative behavior to a workflow net. Since these rules preserve liveness and boundedness, they can also be used to construct object life cycles. In [72], a very similar set of transformations preserving liveness and boundedness of P/T nets is presented, although the context differs from the context of [2]. Other liveness-and-boundedness-preserving transformation rules can be found in [18, 26, 27, 30, 45, 55]. However, these rules are developed for the analysis of net models instead of their construction. Consequently, from the design point of view, they do not always have an intuitive meaning. Of course, this does not mean that they cannot be used during the construction of an object-oriented system design. Since the literature already contains so many studies on transformation rules preserving liveness and boundedness of P/T nets, they are not further discussed in this paper.

Another source of transformation rules preserving object-life-cycle properties are the transformation rules introduced in Section 7, which are developed for the purpose of constructing *subclasses* of object life cycles. For more details, the reader is referred to Section 7.

As mentioned, another solution to make the automatic verification of life-cycle properties feasible is to restrict the class of P/T nets allowed in the design. In Section 5.4, the set of free-choice P/T nets has been introduced as a class of nets that combines expressiveness with strong analysis techniques. It is known that deciding the combination of liveness and boundedness for marked, free-choice P/T nets can be done in linear time [46]. In combination with Theorem 6.7, this leads immediately to the following result.

**Theorem 6.9. (Complexity of deciding life-cycle properties for free-choice P/T nets)** *It is decidable in linear time whether an $M$-labeled, free-choice P/T net is an object life cycle as defined in Definition 6.2.*

**Proof.** Theorem 6.7 and the results of [46]. □

This last theorem is a small improvement of a result in [2] for free-choice workflow nets. It is possible to relax the free-choice requirement in Theorem 6.9 slightly while maintaining the same efficiency in deciding the life-cycle properties (see [2]).

Another class of P/T nets that might be an interesting candidate for defining object life cycles is the class of well-handled P/T nets, as defined in [1, 3]. For well-handled P/T nets, it is also possible to verify the object-life-cycle properties efficiently. As for free-choice P/T nets, this result is based on Theorem 6.7 (Characterization of object life cycles). For more details, the interested reader is referred to [1, 3].

## 6.2 Inheritance relations

Definition 6.2 formalizes the notion of an object life cycle in terms of P/T nets. The next step in the translation of the process-algebraic framework of Section 4 to P/T nets is the formalization of the four inheritance relations of Definition 4.5. Recall that these relations are all defined in terms of two operators, namely encapsulation and abstraction. Encapsulation is used to block method calls; abstraction is used to hide method calls. These two operators can be defined on labeled P/T nets as follows. Note that they only affect the structure of a P/T net. For the sake of readability, they are defined on marked nets.

**Definition 6.10. (Encapsulation)** Let $(N, s)$ be a marked, $M$-labeled P/T net in $\mathcal{N}(M)$, where $N = (P, T_0, F_0, \ell_0)$. For any $H \subseteq E$, the encapsulation operator $\partial_H : \mathcal{N}(M) \to \mathcal{N}(M)$ is a function that removes from a given P/T net all transitions with a label in $H$. Formally, $\partial_H(N, s) = ((P, T_1, F_1, \ell_1), s)$ such that $T_1 = \{t \in T_0 \mid \ell_0(t) \notin H\}$, $F_1 = F_0 \cap ((P \times T_1) \cup (T_1 \times P))$, and $\ell_1 = \ell_0 \cap (T_1 \times M)$.

**Definition 6.11. (Abstraction)** Let $(N, s)$ be a marked, $M$-labeled P/T net in $\mathcal{N}(M)$, where $N = (P, T, F, \ell_0)$. For any $I \subseteq E$, the abstraction operator $\tau_I : \mathcal{N}(M) \to \mathcal{N}(M)$ is a function that renames all transition labels in $I$ to the silent action $\tau$. Formally, $\tau_I(N, s) = ((P, T, F, \ell_1), s)$ such that, for any $t \in T$, $\ell_0(t) \in I$ implies $\ell_1(t) = \tau$ and $\ell_0(t) \notin I$ implies $\ell_1(t) = \ell_0(t)$.

The following property shows that the definition of encapsulation and abstraction is sound with respect to our standard notion of equivalence. It states that two nets with the same behavior also exhibit the same behavior after encapsulating or abstracting one or more actions.

**Property 6.12.** *Branching bisimilarity, $\sim_b$, is a congruence for the encapsulation and abstraction operators.*

**Proof.** Theorem 2.14 proves that branching bisimilarity is an equivalence relation. It remains to be shown that, for any two marked P/T nets $(N_0, s_0)$ and $(N_1, s_1)$ in $\mathcal{N}(M)$ and any sets $H, I \subseteq E$, $(N_0, s_0) \sim_b (N_1, s_1)$ implies that $\partial_H(N_0, s_0) \sim_b \partial_H(N_1, s_1)$ and $\tau_I(N_0, s_0) \sim_b \tau_I(N_1, s_1)$. Let $\mathcal{R} \subseteq \mathcal{N}(M) \times \mathcal{N}(M)$ be a branching bisimulation between $(N_0, s_0)$ and $(N_1, s_1)$. Based on $\mathcal{R}$, a relation $\mathcal{Q} \subseteq \mathcal{N}(M) \times \mathcal{N}(M)$ is defined as the set $\{(\partial_H(N_0, u), \partial_H(N_1, v)) \mid (N_0, u)\mathcal{R}(N_1, v)\}$. It is not difficult to verify that $\mathcal{Q}$ is a branching bisimulation between $(N_0, s_0)$ and $(N_1, s_1)$. Hence, $\sim_b$ is a congruence for the encapsulation operator $\partial_H$. In a similar way, it can be shown that $\sim_b$ is a congruence for the abstraction operator $\tau_I$. □

The introduction of encapsulation and abstraction on P/T nets is sufficient to translate the inheritance relations of Definition 4.5 to the framework of this section.

**Definition 6.13. (Inheritance relations)**

*i)* *Protocol inheritance*:
For any object life cycles $N_0$ and $N_1$ in $\mathcal{O}$, life cycle $N_1$ is a subclass of $N_0$ under protocol inheritance, denoted $N_1 \leq_{pt} N_0$, if and only if there is an $H \subseteq E$ such that $\partial_H(N_1, [i]) \sim_b (N_0, [i])$.

*ii)* *Projection inheritance*:
For any object life cycles $N_0$ and $N_1$ in $\mathcal{O}$, life cycle $N_1$ is a subclass of $N_0$ under projection inheritance, denoted $N_1 \leq_{pj} N_0$, if and only if there is an $I \subseteq E$ such that $\tau_I(N_1, [i]) \sim_b (N_0, [i])$.

*iii)* *Protocol/projection inheritance*:
For any object life cycles $N_0$ and $N_1$ in $\mathcal{O}$, life cycle $N_1$ is a subclass of $N_0$ under protocol/projection inheritance, denoted $N_1 \leq_{pp} N_0$, if and only if there is an $H \subseteq E$ such that $\partial_H(N_1, [i]) \sim_b (N_0, [i])$ and an $I \subseteq E$ such that $\tau_I(N_1, [i]) \sim_b (N_0, [i])$.

*iv)* *Life-cycle inheritance*:
For any object life cycles $N_0$ and $N_1$ in $\mathcal{O}$, life cycle $N_1$ is a subclass of $N_0$ under life-cycle inheritance, denoted $N_1 \leq_{lc} N_0$, if and only if there are an $I \subseteq E$ and an $H \subseteq E$ such that $I \cap H = \emptyset$ and $\tau_I \circ \partial_H(N_1, [i]) \sim_b (N_0, [i])$.

**Example 6.15.** Figure 6.14 shows five object life cycles, each of them modeling a variant of a production unit. Unit $N_0$ is the basic production unit that receives a command, processes material, and delivers output material. Unit $N_1$ extends $N_0$ with error control. If upon completion of the processing phase an error is

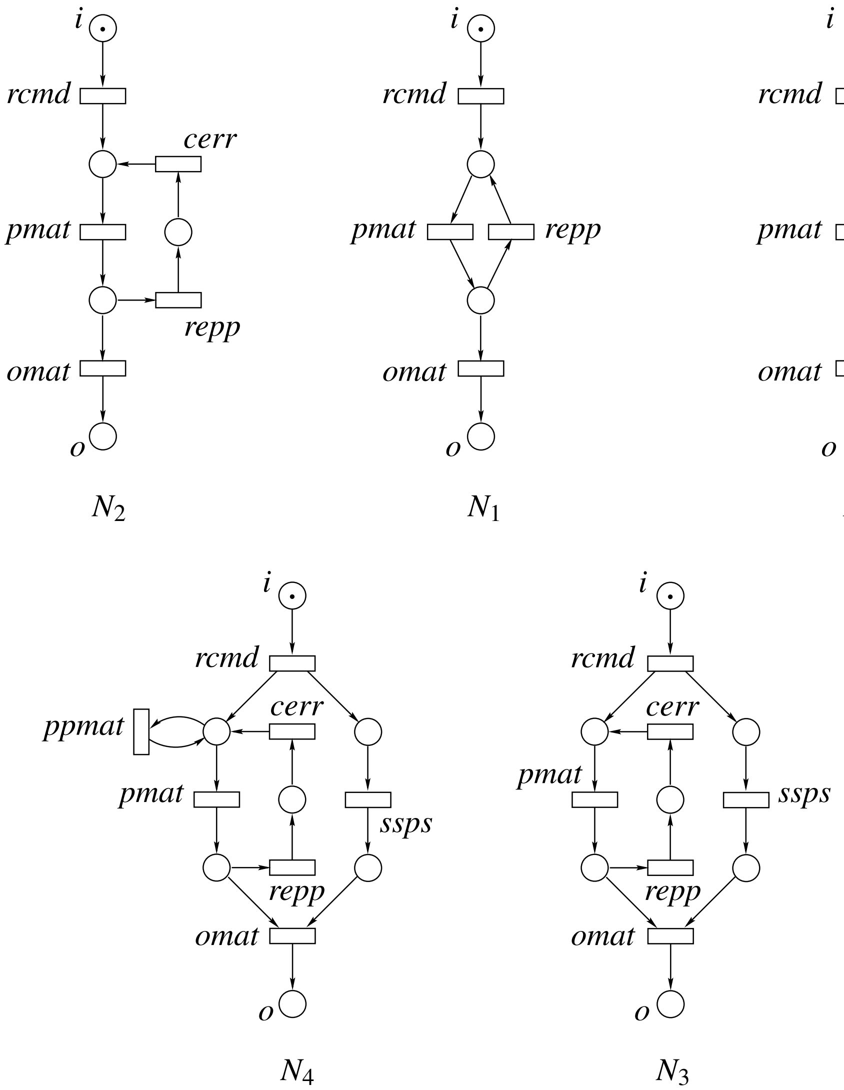



Figure 6.14: Some examples of inheritance relationships.

detected, method *repp* is executed to repeat the processing step. By encapsulating method *repp*, it is easily shown that unit $N_1$ is a subclass of unit $N_0$ under protocol inheritance. Unit $N_2$ introduces a new method *cerr* that an operator may use to correct an error. As unit $N_1$, unit $N_2$ is a subclass under protocol inheritance of unit $N_0$. Again, encapsulating method *repp* is sufficient to prove this relationship. Unit $N_2$ is a subclass under projection inheritance of unit $N_1$. This can be shown by hiding method *cerr*. Unit $N_3$ sends a start-processing signal to the operator in parallel with its processing phase. Hiding method *ssps* shows that it is a subclass under projection inheritance of unit $N_2$. Finally, unit $N_4$ is a subclass under protocol/projection inheritance of unit $N_3$. Encapsulating the preprocessing action *ppmat* yields a life cycle identical to unit $N_3$. Hiding method *ppmat* yields a life cycle which is branching bisimilar to $N_3$. Finally, all units $N_j$, where $0 \leq j \leq 4$, are subclasses of unit $N_0$ under life-cycle inheritance.

In the process-algebraic framework of Section 4, several properties of the four inheritance relations have been proven. Most of these properties carry over to the setting of this section. To facilitate reasoning about life cycles, the alphabet operator is defined on P/T nets.

**Definition 6.16. (Alphabet)** The alphabet operator is a function $\alpha : \mathcal{N}(M) \to \mathcal{P}(E)$. Let $(N, s)$ be a marked, $M$-labeled P/T net in $\mathcal{N}(M)$, with $N = (P, T, F, \ell)$. The alphabet of $(N, s)$ is defined as the set of visible labels of all transitions of the net that are not dead: $\alpha(N, s) = \{\ell(t) \mid t \in T \wedge \ell(t) \neq \tau \wedge t$ is not dead in $(N, s)\}$.

An important property of the alphabet operator is that two equivalent P/T nets have the same alphabet. To prove this property, the following lemma is needed. For every two processes related by some branching bisimulation in a given process space, it says that the branching bisimulation relates any process reachable from one of these two processes to a process reachable from the other process.

**Lemma 6.17.** *Assume that $(\mathcal{P}, \mathcal{A}, \longrightarrow, \downarrow)$ is a process space as defined in Definition 2.1. Furthermore, assume that $\_ \stackrel{*}{\Longrightarrow} \_ \subseteq \mathcal{P} \times \mathcal{P}$ is the reachability relation of Definition 2.2. Let $p$ and $q$ be two processes in $\mathcal{P}$; let $\mathcal{R}$ be a branching bisimulation between $p$ and $q$, as defined in Definition 2.8. For any $p', q' \in \mathcal{P}$,*

*i)* $p \stackrel{*}{\Longrightarrow} p' \Rightarrow (\exists\, q' : q' \in \mathcal{P} : q \stackrel{*}{\Longrightarrow} q' \wedge p'\mathcal{R}q')$ *and*
*ii)* $q \stackrel{*}{\Longrightarrow} q' \Rightarrow (\exists\, p' : p' \in \mathcal{P} : p \stackrel{*}{\Longrightarrow} p' \wedge p'\mathcal{R}q')$.

**Proof.** It is straightforward to prove the two properties by induction on the number of actions needed to reach $p'$ from $p$ and $q'$ from $q$, respectively. □

Given Lemma 6.17 and two branching bisimilar P/T nets, it is not difficult to prove that any non-dead transition in one net corresponds to a non-dead transition with the same label in the other net. This leads easily to the desired congruence property.

**Property 6.18.** *For any two P/T nets $(N_0, s_0)$ and $(N_1, s_1)$ in $\mathcal{N}(M)$,*
$(N_0, s_0) \sim_b (N_1, s_1) \Rightarrow \alpha(N_0, s_0) = \alpha(N_1, s_1)$.

**Proof.** Let $N_0 = (P_0, T_0, F_0, \ell_0)$ and $N_1 = (P_1, T_1, F_1, \ell_1)$. Assume that $(N_0, s_0) \sim_b (N_1, s_1)$. Let $\mathcal{R}$ be a branching bisimulation between $(N_0, s_0)$ and $(N_1, s_1)$. Recall that $\_ \Longrightarrow \_ \subseteq \mathcal{N}(M) \times \mathcal{N}(M)$ is the relation expressing reachability via silent actions, defined in Definition 2.6.

Assume that $a \in \alpha(N_0, s_0)$. It follows from Definitions 6.16 (Alphabet) and 5.20 (Dead transition) that there must exist a reachable marking $u \in [N_0, s_0\rangle$ and a transition $t_0 \in T_0$ such that $(N_0, u)[t_0\rangle$, $a \neq \tau$, and $\ell(t_0) = a$. Hence, Lemma 6.17 *i)* and Definition 5.11 (Reachable markings) yield that there must exist a reachable marking $v \in [N_1, s_1\rangle$ such that $(N_0, u)\mathcal{R}(N_1, v)$. Since $\mathcal{R}$ is a branching bisimulation and $a \neq \tau$, there exist a marking $w$ of $N_1$ such that $(N_1, v) \Longrightarrow (N_1, w)$ and a transition $t_1 \in T_1$ such that $\ell(t_1) = a$ and $(N_1, w)[t_1\rangle$. It follows that $w \in [N_1, s_1\rangle$. Hence, Definition 5.20 (Dead transition) implies that $t_1$ is not dead. Consequently, Definition 6.16 (Alphabet) yields that $a \in \alpha(N_1, s_1)$.

A symmetrical argument proves that any $a \in \alpha(N_1, s_1)$ is also an element of $\alpha(N_0, s_0)$, which completes the proof. □

Since an object life cycle does not have any dead transitions, it is straightforward to calculate its alphabet, as the following property shows.

**Property 6.19.** *For any object life cycle $N = (P, T, F, \ell)$ in $\mathcal{O}$,*
$\alpha(N, [i]) = \{\ell(t) \mid t \in T \wedge \ell(t) \neq \tau\}$.

**Proof.** Definitions 6.2 (Object life cycle) and 6.16 (Alphabet). □

Property 6.19 shows that the alphabet of an object life cycle does not depend on its marking. Therefore, the alphabet of an object life cycle $N \in \mathcal{O}$ is abbreviated $\alpha(N)$.

The properties of the alphabet, encapsulation, and abstraction operators given in Section 4.1, namely Lemmas 4.8, 4.10, 4.11, 4.13, and 4.14, are easily translated to labeled P/T nets. They can be proven straightforwardly from Definitions 6.10 (Encapsulation), 6.11 (Abstraction), and 6.16 (Alphabet).

Properties 4.12, 4.15, and 4.17 of Section 4.1 can now be proven within the framework of this section. The proofs use the basic lemmas for the alphabet, encapsulation, and abstraction operators mentioned above, as well as the congruence results of Properties 6.12 and 6.18. They go along the same lines as the proofs in Section 4.1 and are, hence, omitted.

The first property shows the existence of canonical sets of methods for proving relationships under protocol, projection, and protocol/projection inheritance. Furthermore, to prove a life-cycle-inheritance relationship between two life cycles, it is always possible to choose a partitioning of exactly all the methods new in the subclass into two sets, one containing the methods that are encapsulated and the other one containing methods that are hidden.

**Property 6.20.** *For any object life cycles* $N_0, N_1 \in \mathcal{O}$,

*i)* $N_1 \leq_{pt} N_0 \Leftrightarrow \partial_{\alpha(N_1)\backslash\alpha(N_0)}(N_1, [i]) \sim_b (N_0, [i])$,

*ii)* $N_1 \leq_{pj} N_0 \Leftrightarrow \tau_{\alpha(N_1)\backslash\alpha(N_0)}(N_1, [i]) \sim_b (N_0, [i])$,

*iii)* $N_1 \leq_{pp} N_0 \Leftrightarrow \partial_{\alpha(N_1)\backslash\alpha(N_0)}(N_1, [i]) \sim_b (N_0, [i]) \wedge \tau_{\alpha(N_1)\backslash\alpha(N_0)}(N_1, [i]) \sim_b (N_0, [i])$, *and*

*iv)* $N_1 \leq_{lc} N_0 \Leftrightarrow (\exists H, I : H, I \subseteq \alpha(N_1)\backslash\alpha(N_0) \wedge H \cup I = \alpha(N_1)\backslash\alpha(N_0) \wedge H \cap I = \emptyset :$
$\tau_I \circ \partial_H(N_1, [i]) \sim_b (N_0, [i]))$.

The second property states that the inheritance relations are reflexive and transitive.

**Property 6.21.** *Protocol, projection, protocol/projection, and life-cycle inheritance, as defined in Definition 6.13, are preorders.*

The third property taken from Section 4.1 shows that subclass equivalence under any form of inheritance corresponds to branching bisimilarity.

**Definition 6.22. (Subclass equivalence)** Let $\approx_*$, where $* \in \{pp, pt, pj, lc\}$, be the equivalence relation induced by $\leq_*$. For any object life cycles $N_0$ and $N_1$ in $\mathcal{O}$, $N_0 \approx_* N_1 \Leftrightarrow N_0 \leq_* N_1 \wedge N_1 \leq_* N_0$. The two life cycles are said to be *subclass equivalent* under $*$ inheritance.

**Property 6.23.** *For any object life cycles* $N_0, N_1 \in \mathcal{O}$ *and* $* \in \{pp, pt, pj, lc\}$,
$N_0 \approx_* N_1 \Leftrightarrow (N_0, [i]) \sim_b (N_1, [i])$.

Another result of Section 4.1, namely Property 4.22, describes when a subclass can be refined to a more specialized subclass. It does not have a straightforward translation to the framework of this section. However, the transformation rules introduced in the next section can be used for this purpose.

A final aspect that needs to be studied is the decidability of the four inheritance relations.

**Theorem 6.24. (Decidability of inheritance)** *For any two object life cycles* $N_0$ *and* $N_1$ *in* $\mathcal{O}$, *it is decidable whether* $N_1$ *is a subclass of* $N_0$ *under any of the four inheritance relations of Definition 6.13.*

**Proof.** The first step in proving any inheritance relationship between two object life cycles is to choose appropriate sets of methods that must be encapsulated or hidden. For protocol, projection, and protocol/projection inheritance, Property 6.20 shows that canonical sets defined in terms of the alphabets of $N_0$ and $N_1$ can be chosen. Property 6.19 implies that it is straightforward to calculate these two alphabets. For life-cycle inheritance, Property 6.20 shows that it is always possible to choose a partitioning of the methods in the alphabet of $N_1$ and not in the alphabet of $N_0$. Since the number of transitions of a P/T net is finite, there only exists a finite number of possibilities.

The second step involves checking branching bisimilarity. Observe that, according to Property 6.5, $(N_0, [i])$ and $(N_1, [i])$ are bounded. It follows from Definitions 5.16 (Boundedness), 6.10 (Encapsulation), and 6.11 (Abstraction) that any of the modified object life cycles used in the definition of the four inheritance relations, namely $\partial_H(N_1, [i])$, $\tau_I(N_1, [i])$, and $\tau_I \circ \partial_H(N_1, [i])$ with $H, I \subseteq E$, are also bounded. Obviously, the process corresponding to a bounded P/T net, defined by the semantics of Definition 6.1 and Definition 2.3 (Process), has a finite number of states and transitions. Hence, the processes corresponding to any of the life cycles or modified life cycles mentioned above are all finite. Consequently, checking any of the branching-bisimilarity relationships occurring in Definition 6.13 is decidable. As a result, it is also decidable whether $N_1$ is a subclass of $N_0$ under any of the four inheritance relations. □

The key in the above decidability result is that all the P/T nets in the problem are bounded and, hence, all processes finite. As a result, branching bisimilarity and thus all the inheritance relations are decidable. The exact complexity of deciding branching bisimilarity on bounded P/T nets is unknown. It is known that deciding whether two finite processes are branching bisimilar can be done in polynomial time, where the size of the problem is defined as the number of states and transitions of the two processes [37]. However, constructing the process corresponding to a P/T net requires in the worst case at least exponential space, where the size of the problem is the number of nodes of the P/T net. This space requirement is an immediate consequence of the known lowerbound on the complexity of the reachability problem [32]. Hence, deciding any of the inheritance relations on two object life cycles cannot be done efficiently. This is one of the reasons to study in the next section transformation rules to construct subclasses from a given life cycle.

# 7 Inheritance Rules

An important goal of object-oriented design is to stimulate the *reuse* of software components. One of the aims of this paper is to develop support for the reuse of object life cycles. Therefore, this section proposes a number of *inheritance rules*. Inheritance rules are *transformation* rules on object life cycles that can be used to construct subclasses from a given object life cycle under specific forms of inheritance.

As long as object life cycles are not too complex, it is straightforward to verify whether there exists a specific inheritance relationship between them, both from a computational point of view and from a design point of view. However, the discussion at the end of the previous section shows that computation time might become unacceptable when the object life cycles become too large. In addition, even if there exists an inheritance relationship between two object life cycles, it might not always be meaningful from a design point of view. Therefore, an inheritance rule for the construction of subclasses of object life cycles should preferably satisfy two criteria. First, it should be efficient in computation time. Second, it should represent a meaningful design construct. Unfortunately, these two criteria are often in conflict. Usually, the more general an inheritance rule is, the more useful it is in practical design. The price to be paid is almost always the computation time needed to verify whether a specific transformation satisfies the requirements of the rule. An inheritance rule that satisfies both the abovementioned criteria can be a useful aid in object-oriented design by stimulating the reuse of object life cycles.

This section presents four different inheritance rules. Each one of them corresponds to a design construct which is often used in practice, namely choice, sequential composition, parallel composition, and iteration. The rules are inspired by the axioms of inheritance presented in Section 4.2. Each rule is a compromise between the two criteria mentioned above. The rules are presented in the order of increasing complexity.

The following auxiliary definition is useful in the definition of the inheritance rules.

**Definition 7.1. (Union of labeled P/T nets)** Let $N_0 = (P_0, T_0, F_0, \ell_0)$ and $N_1 = (P_1, T_1, F_1, \ell_1)$ be two $M$-labeled P/T nets such that $(P_0 \cup P_1) \cap (T_0 \cup T_1) = \emptyset$ and such that, for all $t \in T_0 \cap T_1$, $\ell_0(t) = \ell_1(t)$. The union $N_0 \cup N_1$ of $N_0$ and $N_1$ is the labeled P/T net $(P_0 \cup P_1, T_0 \cup T_1, F_0 \cup F_1, \ell_0 \cup \ell_1)$. If two P/T nets satisfy the abovementioned two conditions, their union is said to be *well defined*.

## 7.1 Inheritance rule *PP*

The four inheritance rules of this section are all based on the same principles. The rule that is the easiest one to understand is presented first. It preserves both protocol and projection inheritance. The rule is inspired by Axiom *PP*, as defined in Property 4.29. It is illustrated in Figure 7.2. Let $N_0$ be an object life cycle. Let $N$ be a *connected, free-choice* P/T net such that the union $N_1 = N_0 \cup N$ is well defined. P/T net $N_1$ is a subclass of life cycle $N_0$ under protocol/projection inheritance if the following four conditions are satisfied: *i*) $N_0$ and $N$ only share a single place $p$; *ii*) all transitions of $N$ have a label which does not appear in the alphabet of $N_0$; *iii*) each transition of $N$ with $p$ as one of its input places has a *visible* label, and *iv*) $(N, [p])$ is live and bounded. Inheritance rule *PP* shows that under protocol/projection inheritance, it is allowed to *postpone* behavior. When $N_1$ reaches a state in which place $p$ is marked, it is possible to iterate the behavior defined by $N$ an arbitrary number of times before continuing with the original behavior. The requirement that $(N, [p])$ is free-choice, live, and bounded guarantees that every token consumed from place $p$ by a transition of $N$ can always be returned to $p$. This property of $N$ is crucial for the correctness of rule *PP*.

Inheritance rule *PP* is fairly general. The only requirement that might seem out of place is the requirement that $N$ is free-choice. However, as already explained in Section 5.4, free-choice P/T nets exhibit many interesting properties that do not carry over to general P/T nets. Two examples of such properties are monotonicity of liveness and monotonicity of home markings. These and other properties form the basis of inheritance rule *PP*. In Section 7.6, the role of the free-choice requirement in rule *PP* (and the other inheritance rules of this section) is discussed in more detail.

A positive consequence of the free-choice requirement is that it can be verified very efficiently whether a specific transformation satisfies the conditions of the inheritance rule. Note that rule *PP* does not require that $N_1$ is an object life cycle. The fact that $N_1$ is a life cycle follows from the other requirements. As already explained, the alphabet of an object life cycle is simply the set of its visible transition labels. It has also been mentioned that liveness and boundedness of free-choice P/T nets can be verified in linear time [46]. This means that all requirements of rule *PP* can be verified efficiently.

The correctness of inheritance rule *PP* is proven by showing that it is a special case of two other inheritance rules, namely rule *PJ* and rule *PT*, both given further on in this section. The advantage of such a proof is that it is relatively short. The disadvantage is that it does not provide much insight in the exact working of the rule. On a first reading, the reader is advised to skip the correctness proof of rule *PP*. To understand the details of rule *PP*, it is best to study the lemmas which form the basis of the proof of the remaining inheritance rules. These lemmas are given in the next subsection.

Finally, observe that, when Requirement *iii*) of rule *PP* is dropped, it does no longer preserve protocol inheritance, but that it still preserves projection inheritance.

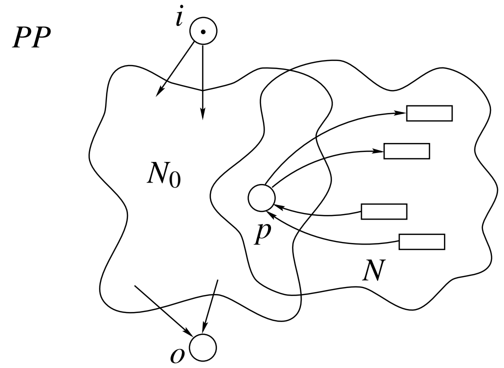



Figure 7.2: A protocol/projection-inheritance rule.

**Theorem 7.3. (Protocol/projection-inheritance rule** *PP*) *Let $N_0 = (P_0, T_0, F_0, \ell_0)$ be an object life cycle in $\mathcal{O}$. If $N = (P, T, F, \ell)$ is a* connected, *$M$-labeled,* free-choice *P/T net with place $p \in P$ such that*

*i)* $p \notin \{i, o\}$, $P_0 \cap P = \{p\}$, $T_0 \cap T = \emptyset$,
*ii)* $(\forall t : t \in T : \ell(t) \notin \alpha(N_0))$,
*iii)* $(\forall t : t \in T \wedge p \in \boldsymbol{i}_N t : \ell(t) \neq \tau)$,
*iv)* $(N, [p])$ *is live and bounded, and*
*v)* $N_1 = N_0 \cup N$ *is well defined,*

*then $N_1$ is an object life cycle in $\mathcal{O}$ such that $N_1 \leq_{pp} N_0$.*

**Proof.** To prove Theorem 7.3, it is shown that it is a special case of both Theorem 7.13 and Theorem 7.17 in Sections 7.3 and 7.4, respectively. Thus, it is shown that the inheritance rule preserves both projection and protocol inheritance, which according to Definition 6.13 (Inheritance relations) means that it also preserves protocol/projection inheritance. Let $N_0$, $N_1$, and $N$ be the three P/T nets satisfying the requirements of Theorem 7.3.

The first part of the proof starts by showing that $N_0$ contains a transition with place $p$ in its postset. Since $N_0$ is a life cycle, it follows from Definition 6.2 (Object life cycle) that $N_0$ is connected. It follows from Definition 5.9 (Connectedness) and the fact that places $i$ and $o$ are distinct that $T_0$ is not empty. Since a life cycle does not contain any dead transitions, Lemma 7.4 below yields that there exists a reachable marking $s \in [N_0, [i]\rangle$ such that $s \geq [p]$. Definitions 5.11 (Reachable markings) and 5.7 (Firing rule) imply that there exists a transition $t_p \in T_0$ such that $p \in \boldsymbol{o}_{N_0} t_p$.

Transition $t_p$ is used to construct a P/T net $N_2 = (P_2, T_2, F_2, \ell_2)$ as follows: $P_2 = P_0$, $T_2 = T \cup \{t_p\}$, $F_2 = F_0 \cup \{(p, t_p), (t_p, p)\}$, and $\ell_2 = \ell_0 \cup \{(t_p, \ell_0(t_p))\}$. It remains to be shown that $N_0$, $N_1$, and $N_2$ satisfy the requirements of Theorem 7.13, where $N_2$ plays the role of $N$.

First, it must be shown that $N_2$ is connected and free-choice. It follows immediately from Definition 5.9 (Connectedness), the construction of $N_2$, and the fact that $N$ is connected that also $N_2$ is connected. To prove that $N_2$ is free-choice, assume that $N_2$ is not free-choice. It follows from Definition 5.41 (Free-choice P/T net), the construction of $N_2$, and the fact that $N$ is free-choice that there must be a transition $t \in T$ such that $\{p\} \subset \boldsymbol{i}_N t$. Since $(N, [p])$ is live, there must be a reachable marking $s \in [N, [p]\rangle$ such that $(N, s)[t\rangle$, which means that $s > [p]$. However, by Property 5.17 (Characterization of boundedness), this contradicts the boundedness of $(N, [p])$. It follows that $N_2$ is free-choice.

Second, it is straightforward to see that $N_0$ and $N_2$ satisfy Requirements *i)* and *ii)* of Theorem 7.13.

Third, it must be shown that $(N_2, [p])$ is live and bounded. It follows from Definitions 5.35 (Trap) and 5.38 (Siphon) that the traps and siphons of $N_2$ are identical to the traps and siphons of $N$. It may be assumed that $T \neq \emptyset$. Otherwise, $N_0$ and $N_1$ are identical, which means that Theorem 7.3 is trivially true. It follows from the facts that both $N$ and $N_2$ are connected and that $(N, [p])$ is live and Theorem 5.44 (Commoner's theorem) that also $(N_2, [p])$ is live. To prove that $(N_2, [p])$ is bounded, observe that the sets of reachable markings of $(N, [p])$ and $(N_2, [p])$ are identical. Thus, since $(N, [p])$ is bounded, also $(N_2, [p])$ is bounded.

Finally, it is straightforward to verify that $N_1 = (P_0, T_0, F_0 \backslash \{(t_p, p)\}, \ell_0) \cup (P_2, T_2, F_2 \backslash \{(p, t_p)\}, \ell_2)$. Thus, also Requirement *iv)* of Theorem 7.13 is satisfied, completing the first part of the proof.

For the second part of the proof, assume that $y \in U$ is a fresh identifier not appearing in $P_0 \cup T_0 \cup P \cup T$ and that $b \in E$ is a fresh (visible) method identifier not in $\alpha(N_0)$ . Let P/T net $N^y = (P, T \cup \{y\}, F \cup \{(p, y), (y, p)\}, \ell \cup \{(y, b)\})$. It is straightforward to verify that $N_0$, $N_1$, and $N^y$ satisfy the requirements of Theorem 7.17, where $p_i = p_o = p$ and $N^y$ plays the role of $N$. The requirement that $N^y$ is connected and free-choice and the requirement that $(N^y, [p])$ is live and bounded (Requirement *iv)* of Theorem 7.17) are proven in the same way as above. Requirements *i)*, *ii)*, and *iii)* of Theorem 7.17 as well as Requirement *v)* are all trivially satisfied. Finally, also Requirement *vi)* is straightforward under the above assumption that $p_i = p_o$, which completes the proof. $\square$

The following lemma is used in the proof of Theorem 7.3. It shows that, in a connected P/T net without dead transitions, it is possible to put a token in an arbitrary place.

**Lemma 7.4.** *Let $N = (P, T, F, \ell)$ be a connected, $M$-labeled P/T net such that $T$ is not empty. Assume that $s \in \mathcal{B}(P)$ is a marking such that $(N, s)$ contains no dead transitions. For any place $p \in P$, there exists a reachable marking $s' \in [N, s\rangle$ such that $s' \geq [p]$.*

**Proof.** If $N$ contains no places, then the lemma is trivial. Therefore, assume that $P \neq \emptyset$. Let $p$ be a place in $P$. It follows from the assumption that $T$ is not empty and Definition 5.9 (Connectedness) that there exists a transition $t \in \boldsymbol{i}p \cup \boldsymbol{o}p$. Since $(N, s)$ has no dead transitions, Definition 5.20 (Dead transition) implies that there exists a marking $s' \in [N, s\rangle$ such that $(N, s')[t\rangle$. If $t \in \boldsymbol{o}p$, then $s' \geq [p]$. If $t \in \boldsymbol{i}p$, Definition 5.7 (Firing rule) implies that $s'' = s' - \boldsymbol{i}t \uplus \boldsymbol{o}t$ is a marking in $[N, s\rangle$ such that $s'' \geq [p]$. □

**Example 7.5.** Consider again the object life cycles in Example 6.15. It is not difficult to verify that the inheritance relationship between $N_4$ and $N_3$ can be proven by means of inheritance rule *PP*.

## 7.2 Live and bounded free-choice P/T nets and home markings

In the analysis of iterative behavior, home markings play an important role. Several results about home markings of live and bounded free-choice P/T nets are essential to the correctness of the inheritance rules in this section. These results are formulated in terms of a number of lemmas. If applicable, the lemmas are explained by means of inheritance rule *PP* of the previous subsection.

The first lemma is a technical result which characterizes the role of the empty marking in the analysis of home markings of live and bounded free-choice P/T nets.

**Lemma 7.6.** *Let $N = (P, T, F, \ell)$ be a connected, labeled, free-choice P/T net. Assume that there exists a marking $s \in \mathcal{B}(P)$ such that $(N, s)$ is live and bounded. Let $s', s'' \in \mathcal{B}(P)$ be two markings of $N$.*

*i)* *$\mathbf{0}$ is a home marking of $(N, \mathbf{0})$;*

*ii)* *if $s'$ is a home marking of $(N, s'')$, then $s' = \mathbf{0}$ if and only if $s'' = \mathbf{0}$.*

**Proof.** The first part of the lemma is proven as follows. Since there exists a marking $s \in \mathcal{B}(P)$ such that $(N, s)$ is bounded, it follows from Property 5.51 that also $(N, \mathbf{0})$ is bounded. Hence, Property 5.17 (Characterization of boundedness) implies that the empty marking is the only reachable marking of $(N, \mathbf{0})$. Thus, it follows trivially from Definition 5.25 (Home marking) that $\mathbf{0}$ is a home marking of $(N, \mathbf{0})$.

The second part of the lemma follows directly from the following two observations. First, since $s'$ is a home marking of $(N, s'')$, it follows from Property 5.32 and Definition 5.25 (Home marking) that $s'$ and $s''$ mark every S-component of $N$ with the same number of tokens. Second, it follows from the fact that there exists a marking $s \in \mathcal{B}(P)$ such that $(N, s)$ is live and bounded and Theorem 5.49 (S-coverability) that every place of $N$ is contained in the set of places of at least one S-component. □

The remaining lemmas all give results about a very specific kind of live and bounded free-choice P/T net, namely a connected free-choice P/T net $N$ with a place $p$ such that $(N, [p])$ is live and bounded. The extension $N$ in inheritance rule *PP* is such a P/T net. As mentioned earlier, the crucial property of $N$ is that any token consumed from place $p$ can always be returned. This property must remain valid under all circumstances. The following lemma states that a single token in place $p$ can always be returned. That is, if life cycle $N_0$ in inheritance rule *PP* puts a single token in place $p$, it is always possible to return this token when it is consumed by a transition of the extension $N$.

**Lemma 7.7.** *Let $N = (P, T, F, \ell)$ be a connected, labeled, free-choice P/T net. If $p \in P$ is a place such that $(N, [p])$ is live and bounded, then $[p]$ is a home marking of $(N, [p])$.*

**Proof.** If $N$ contains no transitions, the property is satisfied trivially. Therefore, assume that $T$ is not empty. Definitions 5.20 (Dead transition) and 5.21 (Liveness) imply that, for any reachable marking $s \in [N, [p]\rangle$ and any transition $t \in T$, $t$ is not dead in $(N, s)$. Lemma 7.4 yields that, for any reachable marking $s \in [N, [p]\rangle$, there exists a marking $s' \in [N, s\rangle$ such that $s' \geq [p]$. Since $(N, [p])$ is bounded, it follows from Property 5.17 (Characterization of boundedness) that $s' = [p]$. Hence, Definition 5.25 (Home marking) yields that $[p]$ is a home marking of $(N, [p])$. □

The following corollary generalizes Lemma 7.7 to an arbitrary number of tokens in place $p$. Considering inheritance rule *PP*, this result is needed because the original life cycle $N_0$ may put more than one token in place $p$.

**Corollary 7.8.** *Let $N = (P, T, F, \ell)$ be a connected, labeled, free-choice P/T net. If $p \in P$ is a place such that $(N, [p])$ is live and bounded, then, for all positive natural numbers $n \in \mathbb{N}$, $[p^n]$ is a home marking of $(N, [p^n])$.*

**Proof.** Property 5.54 (Monotonicity of home markings) and Lemma 7.7. □

Lemma 7.7 and Corollary 7.8 show that any number of tokens put into place $p$ by life cycle $N_0$ in inheritance rule *PP* can be returned when consumed by $N$, provided that no external effects disturb the process of returning tokens to $p$. Property 5.54 (Monotonicity of home markings) implies that this process is not disturbed when firing a transition of $N_0$ *adds* a token to $p$. This raises the question what happens if a transition of $N_0$ *removes* a token from $p$. Lemma 7.10 given below shows that the removal of a token from $p$ does not influence the process either. The following lemma is an auxiliary result needed in the proof of Lemma 7.10.

**Lemma 7.9.** *Let $N = (P, T, F, \ell)$ be a connected, labeled, free-choice P/T net. Assume that $p \in P$ is a place such that $(N, [p])$ is live and bounded. Let $s \in \mathcal{B}(P)$ be an arbitrary marking of $N$ and $n \in \mathbb{N}$ a positive natural number. If $(N, s)$ is live and, for every S-component $(P_0, T_0, F_0, \ell_0)$ of $N$, $s(P_0) = n$, then $[p^n] \in [N, s\rangle$. That is, if $(N, s)$ is live and $s$ marks every S-component of $N$ with $n$ tokens, then marking $[p^n]$ is reachable from $s$.*

**Proof.** If $N$ contains no transitions, it easily follows from the connectedness requirement that $P$ contains only place $p$. As a result, the lemma is trivially satisfied. Therefore, assume that $T$ is not empty. The proof is by induction on $n$.

*Base case*: Assume that $n = 1$. As a consequence, it must be shown that $[p] \in [N, s\rangle$ under the assumption that $s$ is a marking such that $(N, s)$ is live and every S-component of $N$ is marked with exactly one token.

It follows from Property 5.22 and Lemma 7.4 that there exists a marking $s' \in [N, s\rangle$ such that $s' \geq [p]$. It follows from the assumption that $s$ marks every S-component of $N$ with one token and Property 5.32 that also $s'$ marks every S-component of $N$ with one token. It follows from the fact that $(N, [p])$ is live and Property 5.34 that $[p]$ marks every S-component of $N$ with one token. Consequently, for every S-component $(P_0, T_0, F_0, \ell_0)$ of $N$, place $p$ is an element of $P_0$. Recall that Theorem 5.49 (S-coverability) implies that every place of $N$ is contained in the set of places of at least one S-component. Since $s'$ marks every S-component of $N$ with exactly one token and since $s' \geq [p]$, it follows that $s' = [p]$, which completes the proof.

*Inductive step*: The induction hypothesis states that, for some natural number $n \geq 1$ and any marking $s \in \mathcal{B}(P)$ such that $(N, s)$ is live and $s$ marks every S-component of $N$ with $n$ tokens, $[p^n] \in [N, s\rangle$. Assume that $s \in \mathcal{B}(P)$ is a marking such that $(N, s)$ is live and such that $s$ marks every S-component of $N$ with $n + 1$ tokens. It must be shown that $[p^{n+1}] \in [N, s\rangle$.

Property 5.22 and Lemma 7.4 show that there exists a marking $s' \in [N, s\rangle$ such that $s' \geq [p]$. It follows from the assumption and Property 5.32 that $s'$ marks every S-component of $N$ with $n + 1$ tokens. In the base case, it has been shown that $[p]$ marks every S-component of $N$ with exactly one token. Since $s' \geq [p]$, marking $s' - [p]$ marks every S-component of $N$ with $n$ tokens. It follows from the fact that $(N, [p])$ is connected, live, and bounded with $T \neq \emptyset$ and Property 5.47 that $(N, s' - [p])$ is live. The induction hypothesis yields that $[p^n] \in [N, s' - [p]\rangle$. Property 5.15 (Monotonicity of reachable markings) yields that $[p^{n+1}] \in [N, s'\rangle$. Since $s' \in [N, s\rangle$, it follows that $[p^{n+1}] \in [N, s\rangle$, which completes the proof.

□

**Lemma 7.10.** *Let $N = (P, T, F, \ell)$ be a connected, labeled, free-choice P/T net. Assume that $p \in P$ is a place such that $(N, [p])$ is live and bounded. Let $s \in \mathcal{B}(P)$ be an arbitrary marking of $N$ and $n \in \mathbb{N}$ a positive natural number. If $[p^n]$ is a home marking of $(N, [p] \uplus s)$, then $[p^{n-1}]$ is a home marking of $(N, s)$, where $[p^0]$ corresponds to the empty bag $\mathbf{0}$.*

**Proof.** If $T = \emptyset$, the lemma is trivially true. Therefore, assume $T \neq \emptyset$. Two cases must be distinguished.

*i)* First, let $n = 1$. Assume that $[p]$ is a home marking of $(N, [p] \uplus s)$. It must be shown that $\mathbf{0}$ is a home marking of $(N, s)$. Lemma 7.6 implies that it is necessary and sufficient to prove that $s = \mathbf{0}$.
It follows from the assumption and Definition 5.25 (Home marking) that $[p] \in [N, [p] \uplus s\rangle$. Since Property 5.54 (Monotonicity of home markings) and Lemma 7.7 imply that $[p] \uplus s$ is a home marking of $(N, [p] \uplus s)$, it follows that $[p] \uplus s \in [N, [p]\rangle$. However, according to Property 5.17 (Characterization of boundedness), this contradicts the boundedness of $(N, [p])$ unless $s = \mathbf{0}$, which completes this part of the proof.

*ii)* Second, let $n > 1$. Assume that $[p^n]$ is a home marking of $(N, [p] \uplus s)$. It must be shown that $[p^{n-1}]$ is a home marking of $(N, s)$. The aim is to use Theorem 5.52 (Home-marking theorem). This means that it must be shown that *a*) $(N, s)$ is bounded, *b*) $(N, s)$ is live, *c*) $[p^{n-1}] \in [N, s\rangle$, and *d*) $[p^{n-1}]$ marks every proper trap of $N$.
Requirement *a*) follows simply from the fact that $(N, [p])$ is live and bounded and Property 5.51.
Requirement *b*) is proven as follows. Since $(N, [p])$ is connected, live, and bounded with $T \neq \emptyset$, it follows from Property 5.47 that $[p]$ marks every S-component of $N$ with one token. Consequently, $[p^n]$ marks every S-component of $N$ with $n$ tokens. It follows from the assumption that $[p^n]$ is a home marking of $(N, [p] \uplus s)$ and Property 5.32 that $[p] \uplus s$ marks every S-component of $N$ also with $n$ tokens. Hence, $s$ marks every S-component with $n - 1$ tokens. Since $n > 1$, Property 5.47 yields that $(N, s)$ is live.
Requirement *c*) follows from Requirement *b*), the observation made in the proof of Requirement *b*) that $s$ marks every S-component of $N$ with $n - 1$ tokens, and Lemma 7.9.
Requirement *d*) is proven by means of Theorem 5.52 (Home-marking theorem) and Lemma 7.7. It follows from these two results and the fact that $(N, [p])$ is connected, live, and bounded with $T \neq \emptyset$ that $[p]$ marks every proper trap of $N$. Hence, since $n > 1$, $[p^{n-1}]$ also marks every proper trap of $N$.
Combining Requirements *a*) through *d*), the Home-marking theorem yields that $[p^{n-1}]$ is a home marking of $(N, s)$.

□

Let us return to inheritance rule *PP* one more time. The results given so far show that, under all circumstances, it is possible to return all tokens in places of $N$ to place $p$. Even the removal of a token from place $p$ does not disturb this property, as shown by Lemma 7.10 above. This result suggests that it must be possible to return tokens to $p$ without consuming any tokens from $p$ in the process. Lemma 7.11 shows that this is indeed possible.

Recall from Section 5.3 that a sequence of length $n$, for some natural number $n$, over some alphabet of identifiers $A$ is a function from $\{0, \ldots, n-1\}$ to $A$. An element $a \in A$ is said to be an element of a sequence $\sigma$ over $A$ of length $n$, denoted $a \in \sigma$, if and only if $a = \sigma(i)$ for some $0 \leq i < n$. The concatenation of two sequences $\sigma$ and $\sigma'$ over $A$ of length $n$ and $m$, respectively, denoted $\sigma\sigma'$, is the sequence of length $n + m$ defined as follows: for $0 \leq i < n$, $\sigma\sigma'(i) = \sigma(i)$ and, for $n \leq i < n + m$, $\sigma\sigma'(i) = \sigma'(i - n)$.

**Lemma 7.11.** *Let $N = (P, T, F, \ell)$ be a connected, labeled, free-choice P/T net. Assume that $p \in P$ is a place such that $(N, [p])$ is live and bounded. Let $s \in \mathcal{B}(P)$ be an arbitrary marking of $N$ and $n \in \mathbb{N}$ an arbitrary natural number. If $[p^n]$ is a home marking of $(N, s)$, then there exists a firing sequence $\sigma \in T^*$ such that $(N, s)\ [\sigma\rangle\ (N, [p^n])$ and, for all $t \in \boldsymbol{o}p$, $t \notin \sigma$.*

**Proof.** The lemma is trivial if $N$ contains no transitions. Therefore, assume that $T$ is not empty. The proof is by induction on $n$.

*Base case*: Assume that $n = 0$. It must be shown that there exists a firing sequence $\sigma \in T^*$ such that $(N, s)\ [\sigma\rangle\ (N, \mathbf{0})$ and, for all $t \in \boldsymbol{o}p$, $t \notin \sigma$, under the assumption that $\mathbf{0}$ is a home marking of $(N, s)$. Lemma 7.6 yields that $s = \mathbf{0}$. Clearly, the empty sequence $\varepsilon$ satisfies the requirements, completing the proof of the base case.

*Inductive step*: The induction hypothesis states that for some $n \geq 0$ and any marking $s \in \mathcal{B}(P)$ such that $[p^n]$ is a home marking of $(N, s)$, there exists a firing sequence $\sigma \in T^*$ such that $(N, s)\ [\sigma\rangle\ (N, [p^n])$ and, for all $t \in \boldsymbol{o}p$, $t \notin \sigma$. Assume that $s \in \mathcal{B}(P)$ is a marking of $N$ such that $[p^{n+1}]$ is a home marking of $(N, s)$. It must be shown that there exists a firing sequence $\sigma \in T^*$ such that $(N, s)\ [\sigma\rangle\ (N, [p^{n+1}])$ and, for all $t \in \boldsymbol{o}p$, $t \notin \sigma$.

It follows from the fact that $(N, [p])$ is connected, live, and bounded with $T \neq \emptyset$ and Property 5.47 that $[p]$ marks every S-component of $N$ with exactly one token. As a result, $[p^{n+1}]$ marks every S-component of $N$ with $n + 1$ tokens. Property 5.32 and the assumption that $[p^{n+1}]$ is a home marking of $(N, s)$ imply that also $s$ marks every S-component with $n + 1$ tokens. Thus, Property 5.47 yields that $(N, s)$ is live. Consequently, it follows from Property 5.22 and Lemma 7.4 that there exist a reachable marking $s' \in [N, s\rangle$, and thus a firing sequence $\sigma \in T^*$ such that $(N, s)\ [\sigma\rangle\ (N, s')$, such that $s' \geq [p]$. Clearly, $\sigma$ can be chosen such that, for all $t \in \boldsymbol{o}p$, $t \notin \sigma$. Since $s' \in [N, s\rangle$ and $[p^{n+1}]$ is a home marking of $(N, s)$, it follows from Definition 5.25 (Home marking) that $[p^{n+1}]$ is also a home marking of $(N, s')$. Since $s' \geq [p]$, Lemma 7.10 yields that $[p^n]$ is a home marking of $(N, s' - [p])$. Thus, it follows from the induction hypothesis that there exists a firing sequence $\sigma' \in T^*$ such that $(N, s' - [p])\ [\sigma'\rangle\ (N, [p^n])$ and, for all $t \in \boldsymbol{o}p$, $t \notin \sigma'$. Property 5.15 (Monotonicity of reachable markings) yields that $(N, s')\ [\sigma'\rangle\ (N, [p^{n+1}])$. Concatenating firing sequences $\sigma$ and $\sigma'$, yields the desired firing sequence. That is, $(N, s)\ [\sigma\sigma'\rangle\ (N, [p^{n+1}])$ and, for all $t \in \boldsymbol{o}p$, $t \notin \sigma\sigma'$, which completes the proof.

□

## 7.3 Inheritance rule *PJ*

The second inheritance rule of this section, *PJ*, builds upon the results given in the previous subsection. It is inspired by the algebraic axioms of inheritance *PJ*1 and *PJ*2 given in Property 4.27. Theorem 7.13 given below formalizes inheritance rule *PJ*. Figure 7.12 illustrates the rule. It shows that rule *PJ* corresponds to a sequential composition. New behavior may be inserted between sequential parts of a life cycle, yielding a subclass under projection inheritance. In contrast to inheritance rule *PP* of Theorem 7.3, the original life cycle is modified. Basically, inheritance rule *PJ* says that it is allowed to replace an arc in the original life cycle by an entire P/T net. The original life cycle $N_0$ contains a place $p$ which has a transition $t_p$ as one of its input transitions. The modification of $N_0$ is based upon a free-choice P/T net $N$ sharing place $p$ and transition $t_p$ with $N_0$. Place $p$ is the only input place of $t_p$ in $N$. The result of the inheritance rule is the P/T net $N_1$ obtained by taking the union of $N_0$ and $N$ after removing both the arc between $t_p$ and $p$ from $N_0$ and the arc between $p$ and $t_p$ from $N$. The requirement that $(N, [p])$ is live and bounded guarantees that $N_1$ always has the option to move every token that transition $t_p$ would normally have put into place $p$ to place $p$ by only firing transitions of $N$. The requirement that all transitions of $N$ other than $t_p$ are labeled with method identifiers not appearing in the alphabet of $N_0$ guarantees that hiding these methods does not influence the visible behavior of the original life cycle.

As already mentioned, inheritance rule *PP* is a special case of rule *PJ*. Similar to rule *PP*, the only compromise with respect to the generality of *PJ* is the requirement that $N$ is free-choice. The requirements of inheritance rule *PJ* can be verified with the same efficiency as the requirements of *PP*.

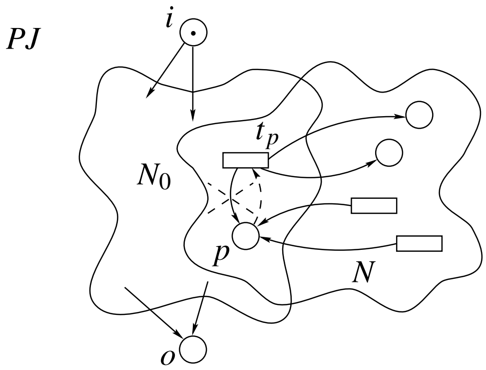



Figure 7.12: A projection-inheritance rule.

**Theorem 7.13. (Projection-inheritance rule** *PJ*) *Let* $N_0 = (P_0, T_0, F_0, \ell_0)$ *be an object life cycle in* $\mathcal{O}$. *If* $N = (P, T, F, \ell)$ *is a* connected, *M-labeled,* free-choice *P/T net with place* $p \in P$ *and transition* $t_p \in T$ *such that*

*i)* $P_0 \cap P = \{p\}$, $T_0 \cap T = \{t_p\}$, $(t_p, p) \in F_0$, $\boldsymbol{i}_N t_p = \{p\}$, *and* $p = o \Rightarrow \boldsymbol{o}_N p = \{t_p\}$,

*ii)* $(\forall t : t \in T \backslash T_0 : \ell(t) \notin \alpha(N_0))$,

*iii)* $(N, [p])$ *is live and bounded, and*

*iv)* $N_1 = (P_0, T_0, F_0 \backslash \{(t_p, p)\}, \ell_0) \cup (P, T, F \backslash \{(p, t_p)\}, \ell)$ *is well defined,*

*then* $N_1$ *is an object life cycle in* $\mathcal{O}$ *such that* $N_1 \leq_{pj} N_0$.

**Proof.** The proof consists of two parts. In the first part, a branching bisimulation is given showing that $(N_0, [i])$ and $(N_1, [i])$ satisfy the requirement of Definition 6.13 *ii*) (Projection inheritance). In the second part, the branching bisimulation is used to prove that $N_1$ is an object life cycle.

In the first part of the proof, it must be shown that there exists an $I \subseteq E$, such that $\tau_I(N_1, [i]) \sim_b (N_0, [i])$. Let $I$ be defined as the set $\alpha(N_1, [i]) \backslash \alpha(N_0)$. (Note that, at this point, it has not yet been shown that $N_1$ is an object life cycle, which means that we cannot omit the initial marking from the expression $\alpha(N_1, [i])$.) Let $\mathcal{R}$ be the relation $\{(\tau_I(N_1, u), (N_0, v)) \mid u \in [N_1, [i]\rangle \wedge v \in [N_0, [i]\rangle \wedge u \lceil (P_0 \backslash \{p\}) = v \lceil (P_0 \backslash \{p\}) \wedge v \lceil \{p\}$ is a home marking of $(N, u \lceil P)\}$. The remainder of this part of the proof is devoted to showing that $\mathcal{R}$ is a branching bisimulation between $\tau_I(N_1, [i])$ and $(N_0, [i])$, as defined in Definition 2.8. Note that the abstraction operator $\tau_I$ does not affect the set of markings reachable from $(N_1, [i])$. Any firing sequence, as defined in Definition 5.12, enabled in $(N_1, [i])$ is also enabled in $\tau_I(N_1, [i])$. Thus, the sets of reachable markings of $(N_1, [i])$ and $\tau_I(N_1, [i])$ are identical.

First, it must be shown that $\tau_I(N_1, [i])\mathcal{R}(N_0, [i])$. Since $N_0$ is an object life cycle and since $(t_p, p) \in F_0$, it follows from Requirement *ii*) (Object creation) of Definition 6.2 (Object life cycle) that $p \neq i$. Therefore, it suffices to show that $\mathbf{0}$ is a home marking of $(N, \mathbf{0})$, which follows immediately from Lemma 7.6.

Second, it must be shown that relation $\mathcal{R}$ is a branching bisimulation. Assume that $u \in [N_1, [i]\rangle$ and $v \in [N_0, [i]\rangle$ are markings such that $\tau_I(N_1, u)\mathcal{R}(N_0, v)$, which implies that $u \lceil (P_0\backslash\{p\}) = v \lceil (P_0\backslash\{p\})$ and $v \lceil \{p\}$ is a home marking of $(N, u \lceil P)$.

*i*) Assume that $t \in T_0 \cup T$ is a transition of $N_1$ such that $\tau_I(N_1, u)[t\rangle$ and such that $\tau_I(N_1, u)[\alpha\rangle\tau_I(N_1, u')$ where $\alpha$ is equal to the method identifier $(\ell_0 \cup \ell)(t)$ and where marking $u' \in [N_1, [i]\rangle$ is equal to $u - \boldsymbol{i}_{N_1}t \uplus \boldsymbol{o}_{N_1}t$. That is, we assume that $\tau_I(N_1, u)$ fires transition $t$. Requirement *i*) of Definition 2.8 states that it must be shown that there exist two markings $v', v'' \in \mathcal{B}(P_0)$ of $N_0$ such that *a*) $(N_0, v) \Longrightarrow (N_0, v'')\ [(\alpha)\rangle\ (N_0, v')$, *b*) $\tau_I(N_1, u)\mathcal{R}(N_0, v'')$, and *c*) $\tau_I(N_1, u')\mathcal{R}(N_0, v')$. Two cases must be distinguished.

First, assume that $t \in T\backslash T_0$. That is, $t$ is a transition of $N$ different from $t_p$. It follows from Requirement *ii*) of Theorem 7.13 and Definition 6.11 (Abstraction) that $\alpha = \tau$. Let $v'' = v' = v$.

Requirement *a*) follows easily from the fact that $\alpha = \tau$.

Requirement *b*) follows immediately from the assumption that $\tau_I(N_1, u)\mathcal{R}(N_0, v)$.

To prove Requirement *c*), it must be shown that $\tau_I(N_1, u')\mathcal{R}(N_0, v)$. By assumption, $u' \in [N_1, [i]\rangle$ and $v \in [N_0, [i]\rangle$. Since $t \in T\backslash T_0$, Requirement *i*) of Theorem 7.13 implies that $u' \lceil (P_0\backslash\{p\}) = u \lceil (P_0\backslash\{p\})$. Since $u \lceil (P_0\backslash\{p\}) = v \lceil (P_0\backslash\{p\})$, it follows that $u' \lceil (P_0\backslash\{p\}) = v \lceil (P_0\backslash\{p\})$. It remains to be shown that $v \lceil \{p\}$ is a home marking of $(N, u' \lceil P)$. The fact that $\tau_I(N_1, u)\ [t\rangle\ \tau_I(N_1, u')$ with $t \in T\backslash T_0$ implies that $(N, u \lceil P)\,[t\rangle\,(N, u' \lceil P)$. Thus, the desired result follows easily from Definition 5.25 (Home marking) and the observation that $v \lceil \{p\}$ is a home marking of $(N, u \lceil P)$.

Second, assume that $t \in T_0$. That is, $t$ is a transition of $N_0$. It can be shown that, in this case, $(N_0, v)$ can mimic the behavior of $\tau_I(N_1, u)$ by also firing transition $t$. Let $v'' = v$ and $v' = v - \boldsymbol{i}_{N_0}t \uplus \boldsymbol{o}_{N_0}t$.

To prove Requirement *a*) introduced above, it is needed that $u(p) \leq v(p)$. That is, marking $v$ marks place $p$ with at least as many tokens as $u$. This property is proven as follows. Since $T \neq \emptyset$ and $(N, [p])$ is connected, free-choice, live, and bounded, Property 5.34 implies that $[p]$ marks every S-component of $N$. Clearly, this means that place $p$ is contained in every S-component of $N$. Consequently, $v \lceil \{p\}$ marks every S-component of $N$ with $v(p)$ tokens. Since $v \lceil \{p\}$ is a home marking of $(N, u \lceil P)$, Property 5.32 proves that $u(p) \leq v(p)$. Since $\tau_I(N_1, u)[t\rangle$, $\alpha = (\ell_0 \cup \ell)(t)$, $\tau_I(N_1, u)\ [\alpha\rangle\ \tau_I(N_1, u')$, $u \lceil (P_0\backslash\{p\}) = v \lceil (P_0\backslash\{p\})$, $u(p) \leq v(p)$, and $t \in T_0$, it follows that $(N_0, v)[t\rangle$ and $(N_0, v)\ [\alpha\rangle\ (N_0, v - \boldsymbol{i}_{N_0}t \uplus \boldsymbol{o}_{N_0}t)$. At this point, Requirement *a*) easily follows from the fact that $v'' = v$ and $v' = v - \boldsymbol{i}_{N_0}t \uplus \boldsymbol{o}_{N_0}t$.

Requirement *b*) follows immediately from the assumptions that $\tau_I(N_1, u)\mathcal{R}(N_0, v)$ and that $v'' = v$.

Finally, Requirement *c*) is proven as follows. The assumptions and Requirement *a*) yield that $u' \in [N_1, [i]\rangle$ and $v' \in [N_0, [i]\rangle$. It follows from the fact that $u \lceil (P_0\backslash\{p\}) = v \lceil (P_0\backslash\{p\})$, and the definitions of $u'$ and $v'$ that $u' \lceil (P_0\backslash\{p\}) = v' \lceil (P_0\backslash\{p\})$. It remains to be shown that $v' \lceil \{p\}$ is a home marking of $(N, u' \lceil P)$. Recall that the markings $u$ and $v$ satisfy the property that $v \lceil \{p\}$ is a home marking of $(N, u \lceil P)$. Five cases can be distinguished. In this case analysis, Requirements *i*) and *iv*) of Theorem 7.13 are used to determine expressions for $v' \lceil \{p\}$ and $u' \lceil P$. Recall that $u' = u - \boldsymbol{i}_{N_1}t \uplus \boldsymbol{o}_{N_1}t$, $v' = v - \boldsymbol{i}_{N_0}t \uplus \boldsymbol{o}_{N_0}t$, and $t \in T_0$. First, if $p \notin \boldsymbol{i}_{N_1}t \cup \boldsymbol{o}_{N_1}t$ or if $p \in \boldsymbol{i}_{N_1}t \cap \boldsymbol{o}_{N_1}t$ and $t \neq t_p$, then $v' \lceil \{p\} = v \lceil \{p\}$ and $u' \lceil P = u \lceil P$. Hence, the desired result simply follows from the above property. Second, assume that $p \in \boldsymbol{i}_{N_1}t \cap \boldsymbol{o}_{N_1}t$ and $t = t_p$. It follows that $p \in \boldsymbol{i}_{N_0}t \cap \boldsymbol{o}_{N_0}t$. Hence, $v' \lceil \{p\} = v \lceil \{p\}$. It follows from the assumptions that $p \in \boldsymbol{i}_{N_1}t$ and $t = t_p$ and the fact that $\boldsymbol{i}_N t_p = \{p\}$ that $\boldsymbol{i}_{N_1}t \lceil P = \{p\}$. Since $p \in \boldsymbol{o}_{N_1}t$, it follows that $p \in \boldsymbol{o}_N t$. Since $\boldsymbol{i}_N t_p = \{p\}$ and $(N, [p])$ is bounded, Property 5.17 (Characterization of boundedness) yields that $\boldsymbol{o}_N t_p = \{p\}$. Consequently,

$\boldsymbol{o}_{N_1}t \upharpoonright P = \{p\}$. Thus, $u' \upharpoonright P = u \upharpoonright P$. As in the previous case, the desired result simply follows from the abovementioned property that $v \upharpoonright \{p\}$ is a home marking of $(N, u \upharpoonright P)$. Third, assume that $p \in \boldsymbol{i}_{N_1}t \backslash \boldsymbol{o}_{N_1}t$ and $t \neq t_p$. It follows from $p \in \boldsymbol{i}_{N_1}t$ and $t \in T_0$ that $p \in \boldsymbol{i}_{N_0}t$. It follows from $p \notin \boldsymbol{o}_{N_1}t$ and $t \neq t_p$ that $p \notin \boldsymbol{o}_{N_0}t$. Hence, $v' \upharpoonright \{p\} = v \upharpoonright \{p\} - [p]$ and $u' \upharpoonright P = u \upharpoonright P - [p]$. As a result, the desired result follows from the above property and Lemma 7.10. Fourth, assume that $p \in \boldsymbol{i}_{N_1}t \backslash \boldsymbol{o}_{N_1}t$ and $t = t_p$. It follows from $p \in \boldsymbol{i}_{N_1}t$ and $\boldsymbol{i}_N t_p = \{p\}$ that $\boldsymbol{i}_{N_1}t \upharpoonright P = \{p\} = \boldsymbol{i}_N t$. It follows from the requirements of Theorem 7.13 that $\boldsymbol{o}_{N_1}t_p \upharpoonright P = \boldsymbol{o}_N t_p$. Hence, $u' \upharpoonright P = u \upharpoonright P - \boldsymbol{i}_N t \uplus \boldsymbol{o}_N t$. It follows from $p \in \boldsymbol{i}_{N_1}t$ that $p \in \boldsymbol{i}_{N_0}t$. It follows from $t = t_p$ that $p \in \boldsymbol{o}_{N_0}t$. Hence, $v' \upharpoonright \{p\} = v \upharpoonright \{p\}$. Since, by assumption, $\tau_I(N_1, u)[t\rangle$ and $p \in \boldsymbol{i}_{N_1}t$, it follows from the fact that $\boldsymbol{i}_N t = \{p\}$ that also $(N, u \upharpoonright P)[t\rangle$. Consequently, $u' \upharpoonright P \in [N, u \upharpoonright P\rangle$. Thus, Definition 5.25 (Home marking), the fact that $v' \upharpoonright \{p\} = v \upharpoonright \{p\}$, and the assumption that $v \upharpoonright \{p\}$ is a home marking of $(N, u \upharpoonright P)$ yield that $v' \upharpoonright \{p\}$ is a home marking of $(N, u' \upharpoonright P)$. Finally, if $p \in \boldsymbol{o}_{N_1}t \backslash \boldsymbol{i}_{N_1}t$, then also $p \in \boldsymbol{o}_{N_0}t \backslash \boldsymbol{i}_{N_0}t$. Thus, $v' \upharpoonright \{p\} = v \upharpoonright \{p\} \uplus [p]$ and $u' \upharpoonright P = u \upharpoonright P \uplus [p]$. It follows from the assumption that $v \upharpoonright \{p\}$ is a home marking of $(N, u \upharpoonright P)$ and Lemma 7.6 that $v \upharpoonright \{p\} = \mathbf{0}$ if and only if $u \upharpoonright P = \mathbf{0}$. Hence, if $v \upharpoonright \{p\} = u \upharpoonright P = \mathbf{0}$, then the desired result follows from the fact that $(N, [p])$ is live and bounded and Lemma 7.7. If $v \upharpoonright \{p\} \neq \mathbf{0}$ and $u \upharpoonright P \neq \mathbf{0}$, it follows from the fact that $(N, [p])$ is live and Property 5.46 (Monotonicity of liveness) that $(N, v \upharpoonright \{p\})$ is live. Definitions 5.21 (Liveness) and 5.25 (Home marking) yield that $(N, u \upharpoonright P)$ is live. The fact that $(N, [p])$ is bounded and Property 5.51 yield that $(N, u \upharpoonright P)$ is also bounded. Hence, in this case, the desired result follows from the property that $v \upharpoonright \{p\}$ is a home marking of $(N, u \upharpoonright P)$ and Property 5.54 (Monotonicity of home markings).

*ii)* Assume that $t \in T_0$ is a transition of $N_0$ such that $(N_0, v)[t\rangle$ and such that $(N_0, v)\ [\alpha\rangle\ (N_0, v')$ where $\alpha$ is equal to the method identifier $\ell_0(t)$ and where marking $v' \in [N_0, [i]\rangle$ is equal to $v - \boldsymbol{i}_{N_0}t \uplus \boldsymbol{o}_{N_0}t$. Requirement *ii)* of Definition 2.8 states that it must be shown that there exist markings $u', u'' \in \mathcal{B}(P_0 \cup P)$ such that *a)* $\tau_I(N_1, u) \Longrightarrow \tau_I(N_1, u'')\ [(\alpha)\rangle\ \tau_I(N_1, u')$, *b)* $\tau_I(N_1, u'')\mathcal{R}(N_0, v)$, and *c)* $\tau_I(N_1, u')\mathcal{R}(N_0, v')$.

Let $u'' = v$ and $u' = v - \boldsymbol{i}_{N_1}t \uplus \boldsymbol{o}_{N_1}t$. This choice for $u''$ and $u'$ is inspired by the fact that it is possible to reach marking $v$ from $u$ by firing as many transitions of $N$ as necessary. Since marking $v$ enables transition $t$, it is straightforward to prove the desired requirements.

Requirement *a)* is proven as follows. The assumption that $v \upharpoonright \{p\}$ is a home marking of $(N, u \upharpoonright P)$, Requirements *i)* and *iii)* of Theorem 7.13, and Lemma 7.11 yield that there exists a firing sequence $\sigma \in (T \backslash \{t_p\})^*$ such that $(N, u \upharpoonright P)[\sigma\rangle(N, v \upharpoonright \{p\})$. The assumption that $u \upharpoonright (P_0 \backslash \{p\}) = v \upharpoonright (P_0 \backslash \{p\})$ and Requirement *iv)* of Theorem 7.13 imply that $\tau_I(N_1, u)\ [\sigma\rangle\ (N_1, v)$. Definition 6.11 (Abstraction), the definition of $I$, and Requirement *ii)* of Theorem 7.13 yield that $\tau_I(N_1, u) \Longrightarrow \tau_I(N_1, v)$. It remains to be shown that $\tau_I(N_1, v)\ [\alpha\rangle\ \tau_I(N_1, u')$. This result easily follows from the observation that, for all $t \in T_0$, $\boldsymbol{i}_{N_1}t = \boldsymbol{i}_{N_0}t$, the facts that $(N_0, v)[t\rangle$ and $\ell_0(t) = \alpha$, the definitions of $u'$ and $I$, and Definition 5.7 (Firing rule), thus completing the proof of Requirement *a)*.

To prove Requirement *b)*, it must be shown that $\tau_I(N_1, v)\mathcal{R}(N_0, v)$. By assumption, $v \in [N_0, [i]\rangle$. Since, also by assumption, $u \in [N_1, [i]\rangle$, Requirement *a)* shows that $v \in [N_1, [i]\rangle$. Clearly, also $v \upharpoonright (P_0 \backslash \{p\}) = v \upharpoonright (P_0 \backslash \{p\})$. Thus, since $v \upharpoonright P = v \upharpoonright \{p\}$, Requirement *b)* follows from Corollary 7.8 and Lemma 7.6 *i)*.

To prove Requirement *c)*, two cases must be distinguished. First, assume that $t \neq t_p$. It follows from the requirements of Theorem 7.13 that $\boldsymbol{i}_{N_1}t = \boldsymbol{i}_{N_0}t$ and $\boldsymbol{o}_{N_1}t = \boldsymbol{o}_{N_0}t$. Thus, Requirement *c)* simplifies to $\tau_I(N_1, v')\mathcal{R}(N_0, v')$, which is proven in a similar way as Requirement *b)*. Second, assume that $t = t_p$. It follows from the assumptions and the proof of Requirement *a)* that $v' \in [N_0, [i]\rangle$ and $u' \in [N_1, [i]\rangle$. It follows from the requirements of Theorem 7.13 that $\boldsymbol{i}_{N_1}t = \boldsymbol{i}_{N_0}t$ and $\boldsymbol{o}_{N_1}t = \boldsymbol{o}_{N_0}t \backslash \{p\} \cup \boldsymbol{o}_N t$. The definitions of $v'$ and $u'$ yield that $u' \upharpoonright (P_0 \backslash \{p\}) = v' \upharpoonright (P_0 \backslash \{p\})$. It remains to be shown that $v' \upharpoonright \{p\}$ is a home marking of $(N, u' \upharpoonright P)$. Again two cases need to be distinguished. If

$p \in \boldsymbol{i}_{N_0} t$, the fact that $t = t_p$ and Requirement *i*) of Theorem 7.13 yield that $v' \lceil \{p\} = v \lceil \{p\}$. Since $t = t_p$, $\boldsymbol{i}_{N_1} t = \boldsymbol{i}_{N_0} t$, $\boldsymbol{o}_{N_1} t = \boldsymbol{o}_{N_0} t \backslash \{p\} \cup \boldsymbol{o}_N t$, and $\boldsymbol{i}_N t_p = \{p\}$, it follows that $u' \lceil P = (v - \boldsymbol{i}_{N_1} t \uplus \boldsymbol{o}_{N_1} t) \lceil P = v \lceil \{p\} - [p] \uplus \boldsymbol{o}_N t = v \lceil \{p\} - \boldsymbol{i}_N t \uplus \boldsymbol{o}_N t$. Since $p \in \boldsymbol{i}_{N_0} t$ and $(N_0, v)[t\rangle$, it follows that $v \geq [p]$. Thus, Definition 5.7 (Firing rule) yields that $(N, v \lceil \{p\})\, [\ell(t)\rangle\, (N, u' \lceil P)$; in addition, Corollary 7.8 yields that $v \lceil \{p\}$ is a home marking of $(N, v \lceil \{p\})$. It follows from Definition 5.25 (Home marking) and the facts that $v' \lceil \{p\} = v \lceil \{p\}$ and that $(N, v \lceil \{p\})\, [\ell(t)\rangle\, (N, u' \lceil P)$ that $v' \lceil \{p\}$ is a home marking of $(N, u' \lceil P)$, which completes the proof in this case. If $p \notin \boldsymbol{i}_{N_0} t$, then $v' \lceil \{p\} = v \lceil \{p\} \uplus [p]$. Since $\boldsymbol{i}_{N_1} t = \boldsymbol{i}_{N_0} t$ and $\boldsymbol{o}_{N_1} t = \boldsymbol{o}_{N_0} t \backslash \{p\} \cup \boldsymbol{o}_N t$, it follows that $u' \lceil P = v \lceil \{p\} \uplus \boldsymbol{o}_N t$. Corollary 7.8 yields that $v \lceil \{p\} \uplus [p]$ is a home marking of $(N, v \lceil \{p\} \uplus [p])$. It follows from Definition 5.7 (Firing rule) that $(N, v \lceil \{p\} \uplus [p])\, [\ell(t)\rangle\, (N, u' \lceil P)$. Definition 5.25 (Home marking) yields that $v' \lceil \{p\}$ is a home marking of $(N, u' \lceil P)$, which completes the proof also in this case.

*iii*) Requirement *iii*) of Definition 2.8 (Branching bisimilarity) states that the following two implications must be shown:$\downarrow \tau_I(N_1, u) \Rightarrow \Downarrow (N_0, v)$ and $\downarrow (N_0, v) \Rightarrow \Downarrow \tau_I(N_1, u)$.

First, assume that $\downarrow \tau_I(N_1, u)$. It follows from Definition 6.1 (Semantics of $M$-labeled P/T nets) that $u = [o]$. If $p \neq o$, then $u \lceil (P_0 \backslash \{p\}) = [o]$ and $u \lceil P = \mathbf{0}$. Since $\tau_I(N_1, u) \mathcal{R} (N_0, v)$, it follows from the definition of $\mathcal{R}$ that $v \lceil (P_0 \backslash \{p\}) = [o]$ and that $v \lceil \{p\}$ is a home marking of $(N, \mathbf{0})$. Lemma 7.6 yields that $v \lceil \{p\} = \mathbf{0}$. Hence, $v = [o]$. Clearly, Definition 6.1 yields that $\downarrow (N_0, v)$. It follows from Definition 2.7 ($\Downarrow$) that $\Downarrow (N_0, v)$. If $p = o$, then $u \lceil (P_0 \backslash \{o\}) = v \lceil (P_0 \backslash \{o\}) = \mathbf{0}$ and $v \lceil \{o\}$ is a home marking of $(N, [o])$. Since $(N, [o])$ is live and bounded, Lemma 7.7 shows that $[o]$ is a home marking of $(N, [o])$. Lemma 7.6 *ii*) implies that $v \lceil \{o\}$ cannot be the empty bag. Furthermore, it follows from Property 5.17 (Characterization of boundedness) that $v \lceil \{o\} \not> [o]$. Combining these results implies that $v \lceil \{o\} = [o]$. Thus, we may conclude that $v = v \lceil (P_0 \backslash \{o\}) \uplus v \lceil \{o\} = [o]$, which by Definitions 6.1 and 2.7 completes the proof of the first implication.

Second, assume that $\downarrow (N_0, v)$. It follows from Definition 6.1 (Semantics of $M$-labeled P/T nets) that $v = [o]$. If $p \neq o$, then it follows from the definition of $\mathcal{R}$ that $u \lceil (P_0 \backslash \{p\}) = [o]$ and that $\mathbf{0}$ is a home marking of $(N, u \lceil P)$. Lemma 7.6 yields that $u \lceil P = \mathbf{0}$. As a result, $u = [o]$. Hence, $\downarrow \tau_I(N_1, u)$, which implies that $\Downarrow \tau_I(N_1, u)$. If $p = o$, then $u \lceil (P_0 \backslash \{o\}) = \mathbf{0}$ and $[o]$ is a home marking of $(N, u \lceil P)$. Lemma 7.11 and Requirement *ii*) of Theorem 7.13 imply that $\tau_I(N_1, u) \Longrightarrow \tau_I(N_1, [o])$. Hence, Definition 2.7 yields that $\Downarrow \tau_I(N_1, u)$. This last result proves the second implication, completing the proof that $\mathcal{R}$ satisfies Requirement *iii*) of Definition 2.8.

At this point, it has been shown that relation $\mathcal{R}$ as defined above is a branching bisimulation. As a result, $N_1$ is a subclass of $N_0$ under projection inheritance, provided that $N_1$ is a life cycle. The second part of the proof is devoted to this last requirement.

Net $N_1$ satisfies Requirement *i*) (Connectedness) of Definition 6.2 (Object life cycle), because both $N$ and $N_0$ are connected. The latter follows from the fact that $N_0$ is an object life cycle. The fact that $N_1$ satisfies Requirement *ii*) (Object creation) of Definition 6.2 follows from the observation that $(t_p, p) \in F_0$, which by the fact that $N_0$ is an object life cycle implies that $p \neq i$. Requirement *iii*) (Object termination) follows from the requirement in Theorem 7.13 that $p = o$ implies that $\boldsymbol{o}_N p = \{t_p\}$. The remainder of the proof shows that $N_1$ also satisfies Requirements *iv*), *v*), and *vi*) of Definition 6.2. It uses the fact that the relation $\mathcal{R}$ of the first part of the proof is a branching bisimulation between $\tau_I(N_1, [i])$ and $(N_0, [i])$, with $I = \alpha(N_1, [i]) \backslash \alpha(N_0)$.

Requirement *iv*) (Proper termination) means that the following must be shown: For every reachable marking $s \in [N_1, [i]\rangle$, $o \in s$ implies $s = [o]$. Let $u \in [N_1, [i]\rangle$ be a reachable marking of $(N_1, [i])$ such that $o \in u$. Since $\mathcal{R}$ is a branching bisimulation between $\tau_I(N_1, [i])$ and $(N_0, [i])$, Lemma 6.17 implies that there exists a marking $v \in [N_0, [i]\rangle$ such that $\tau_I(N_1, u) \mathcal{R} (N_0, v)$. Consequently, $u \lceil (P_0 \backslash \{p\}) = v \lceil (P_0 \backslash \{p\})$ and $v \lceil \{p\}$ is a home marking of $(N, u \lceil P)$. Assume that $p \neq o$. Since $o \in u$ and $o \in P_0 \backslash \{p\}$, this means that $o \in v$. It follows from the fact that $N_0$ is an object life cycle and Requirement *iv*) (Proper termination) of Definition 6.2 (Object life cycle) that $v = [o]$. This yields that $u \lceil (P_0 \backslash \{p\}) = [o]$. Another consequence of the fact that $v = [o]$ is that $v \lceil \{p\} = \mathbf{0}$, which means that $\mathbf{0}$ is a home marking of $(N, u \lceil P)$. Lemma 7.6 yields that $u \lceil P = \mathbf{0}$. Since $u \lceil (P_0 \backslash \{p\}) = [o]$, $u = u \lceil (P_0 \backslash \{p\}) \uplus u \lceil P = [o] \uplus \mathbf{0} = [o]$, which proves that $N_1$ satisfies Requirement *iv*) (Proper termination) of Definition 6.2 in this case. Assume that $p = o$. It follows that $v \lceil \{o\}$ is a home marking of $(N, u \lceil P)$ and, since $o \in u$, $u \lceil P \geq [o]$. Since $(N, [o])$ is live and bounded, Lemma 7.6 *ii*) implies that $v \lceil \{o\} \neq \mathbf{0}$; Requirement *iv*) (Proper termination) of Definition 6.2 and the fact that $N_0$ is an object life cycle with $v \in [N_0, [i]\rangle$ implies that $v \not> [o]$. Thus, $v \lceil \{o\} = v = [o]$. This result has two consequences. First, $u \lceil (P_0 \backslash \{o\}) = v \lceil (P_0 \backslash \{o\}) = \mathbf{0}$. Second, $u \lceil P = [o]$. The latter follows from the following observations. By assumption, $(N, [o])$ is free-choice, connected, live, and bounded. Property 5.34 implies that $[o]$ marks every S-component of $N$ with one token and, thus, that place $o$ is an element of every S-component of $N$. It follows from the fact that $[o]$ is a home marking of $(N, u \lceil P)$ and Property 5.32 that $u \lceil P$ marks every S-component of $N$ also with a single token. Recall that Theorem 5.49 (S-coverability) implies that every place of $N$ is contained in the set of places of at least one S-component. Since $u \lceil P$ and $[o]$ both mark every S-component of $N$ with exactly one token and since $u \lceil P \geq [o]$, it follows that $u \lceil P = [o]$, which is the desired result. Summarizing the results obtained so far yields that $u = u \lceil (P_0 \backslash \{o\}) \uplus u \lceil P = \mathbf{0} \uplus [o] = [o]$, which proves that $N_1$ satisfies Requirement *iv*) of Definition 6.2 also in this case.

To prove that $N_1$ satisfies Requirement *v*) (Termination option) of Definition 6.2, it must be shown that, for every reachable marking $s \in [N_1, [i]\rangle$, $[o] \in [N_1, s\rangle$. Let $u \in [N_1, [i]\rangle$ be a reachable marking of $(N_1, [i])$. As before, using the fact that $\mathcal{R}$ is a branching bisimulation between $\tau_I(N_1, [i])$ and $(N_0, [i])$, Lemma 6.17 yields that there exists a marking $v \in [N_0, [i]\rangle$ such that $\tau_I(N_1, u)\mathcal{R}(N_0, v)$. Consequently, $u \lceil (P_0 \backslash \{p\}) = v \lceil (P_0 \backslash \{p\})$ and $v \lceil \{p\}$ is a home marking of $(N, u \lceil P)$. Since $(N, [p])$ is live and bounded, Lemma 7.11 and Requirement *i*) of Theorem 7.13 yield that there exists a firing sequence $\sigma \in (T \backslash \{t_p\})^*$ such that $(N, u \lceil P)\,[\sigma\rangle\,(N, v \lceil \{p\})$. It follows from Requirements *i*) and *iv*) of Theorem 7.13 that $(N_1, u)\,[\sigma\rangle$ $(N_1, v)$. It remains to be shown that $[o]$ can be reached from marking $v$. It follows from the fact that $N_0$ is an object life cycle, the fact that $v \in [N_0, [i]\rangle$, and Requirement *v*) (Termination option) of Definition 6.2 (Object life cycle) that $[o] \in [N_0, v\rangle$. Let $\sigma_0 \in {T_0}^*$ be a firing sequence such that $(N_0, v)\ [\sigma_0\rangle\ (N_0, [o])$. It is possible to transform $\sigma_0$ into a firing sequence $\bar{\sigma_0} \in (T_0 \cup T)^*$ such that $(N_1, v)\ [\bar{\sigma_0}\rangle\ (N_1, [o])$. This transformation is inductively defined as follows. If $t_p \notin \sigma_0$, then $\bar{\sigma_0} = \sigma_0$. Clearly, this firing sequence satisfies the requirement. If $t_p \in \sigma_0$, then assume that $\sigma_0$ is of the form $\sigma_1 t_p \sigma_2$ with $t_p \notin \sigma_1$. Let $v' \in \mathcal{B}(P_0)$ be the marking of $N_0$ such that $(N_0, v)\ [\sigma_1 t_p\rangle\ (N_0, v')\ [\sigma_2\rangle\ (N_0, [o])$. It follows from the requirements of Theorem 7.13 that $(N_1, v)\ [\sigma_1 t_p\rangle\ (N_1, v' - [p] \uplus \boldsymbol{o}_N t_p)$. Note that $v' \geq [p]$. Corollary 7.8 implies that $v' \lceil \{p\}$ is a home marking of $(N, v' \lceil \{p\})$. Since $\boldsymbol{i}_N t_p = \{p\}$, it follows from Definition 5.7 (Firing rule) that $(N, v' \lceil \{p\})[\ell(t)\rangle(N, v' \lceil \{p\} - [p] \uplus \boldsymbol{o}_N t_p)$. Definition 5.25 (Home marking) implies that $v' \lceil \{p\}$ is also a home marking of $(N, v' \lceil \{p\} - [p] \uplus \boldsymbol{o}_N t_p)$. Lemma 7.11 yields that there exists a firing sequence $\sigma_3 \in (T \backslash \{t_p\})^*$ such that $(N, v' \lceil \{p\} - [p] \uplus \boldsymbol{o}_N t_p)\ [\sigma_3\rangle\ (N, v' \lceil \{p\})$. Consequently, also $(N_1, v' - [p] \uplus \boldsymbol{o}_N t_p)\ [\sigma_3\rangle\ (N_1, v')$. Thus, $(N_1, v)\ [\sigma_1 t_p \sigma_3\rangle\ (N_1, v')$. Choosing $\bar{\sigma_0} = \sigma_1 t_p \sigma_3 \bar{\sigma_2}$, it follows that $(N_1, v)\ [\bar{\sigma_0}\rangle\ (N_1, [o])$. Since $(N_1, u)\ [\sigma\rangle\ (N_1, v)$, it follows that $(N_1, u)\ [\sigma \bar{\sigma_0}\rangle\ (N_1, [o])$, proving that $[o] \in [N_1, u\rangle$. Thus, $(N_1, [i])$ satisfies Requirement *v*) (Termination option) of Definition 6.2.

It remains to prove Requirement *vi*) of Definition 6.2 saying that $(N_1, [i])$ contains no dead transitions, as defined in Definition 5.20. Since $N_1$ is defined as the union of $N_0$ and $N$, the set of transitions of $N_1$ is the set $T_0 \cup T$. The goal is to construct, for every transition of $N_1$, a firing sequence leading to a marking that enables that transition. First, let $t$ be a transition in $T_0$. It follows from the fact that $N_0$ is an object life cycle that $(N_0, [i])$ has no dead transitions. Consequently, there exist a firing sequence $\sigma \in {T_0}^*$ and a marking $s \in \mathcal{B}(P_0)$ such that $(N_0, [i])\ [\sigma\rangle\ (N_0, s)$ and $(N_0, s)[t\rangle$. Let $\bar{\sigma}$ be the firing sequence in $(T_0 \cup T)^*$ that is the result of transforming $\sigma$ according to the transformation introduced in the previous part of the proof. Recall that, for all $t \in T_0$, $\boldsymbol{i}_{N_1} t = \boldsymbol{i}_{N_0} t$. It follows that $(N_1, [i])\ [\bar{\sigma}\rangle\ (N_1, s)$ and $(N_1, s)[t\rangle$. Thus, $t$ is not dead in $(N_1, [i])$. Second, let $t$ be a transition in $T \backslash \{t_p\}$. Since $(N_0, [i])$ does not contain any dead transitions, there exists a firing sequence $\sigma \in (T_0 \backslash \{t_p\})^*$ and a marking $s \in \mathcal{B}(P_0)$ such that $(N_0, [i])\ [\sigma t_p\rangle\ (N_0, s)$. Note that $s \geq [p]$. It follows from the requirements of Theorem 7.13 that $(N_1, [i])\ [\sigma t_p\rangle\ (N_1, s - [p] \uplus \boldsymbol{o}_N t_p)$. Since $(N, [p])$ is live, it follows from Definitions 5.21 (Liveness) and 5.7 (Firing rule) and the fact that $\boldsymbol{i}_N t_p = \{p\}$ that also $(N, \boldsymbol{o}_N t_p)$ is live. As a result of Property 5.22, transition $t$ is not dead in $(N, \boldsymbol{o}_N t_p)$. It follows that there is a firing sequence $\sigma_0 \in T^*$ and a marking $s' \in \mathcal{B}(P)$ such that $(N, \boldsymbol{o}_N t_p)\ [\sigma_0\rangle\ (N, s')$ and $(N, s')[t\rangle$. Sequence $\sigma_0$ can be chosen such that $t_p \notin \sigma_0$. To prove this claim, assume that $\sigma_0 = \sigma_1 t_p \sigma_2$ with $t_p \notin \sigma_2$. Let $s'' \in \mathcal{B}(P)$ be the marking such that $(N, \boldsymbol{o}_N t_p)\ [\sigma_1 t_p\rangle\ (N, s'')\ [\sigma_2\rangle\ (N, s')$. It follows from Definition 5.7 (Firing rule) that $s'' \geq \boldsymbol{o}_N t_p$. Since $(N, [p])$ is bounded, it follows from Property 5.51 that also $(N, \boldsymbol{o}_N t_p)$ is bounded. Thus, Property 5.17 (Characterization of boundedness) yields that $s'' = \boldsymbol{o}_N t_p$. Consequently, $\sigma_2$ is a firing sequence, satisfying the desired property that $t_p \notin \sigma_2$ and $(N, \boldsymbol{o}_N t_p)\ [\sigma_2\rangle\ (N, s')$. Therefore, assume that $t_p \notin \sigma_0$. Since $s \geq [p]$, it follows that $(N_1, s - [p] \uplus \boldsymbol{o}_N t_p)\ [\sigma_0\rangle\ (N_1, s - [p] \uplus s')$. Since $(N, s')[t\rangle$, it follows that $(N_1, s - [p] \uplus s')[t\rangle$. From $(N_1, [i])\ [\sigma t_p\rangle\ (N_1, s - [p] \uplus \boldsymbol{o}_N t_p)$, it follows that $(N_1, [i])\ [\sigma t_p \sigma_0\rangle\ (N_1, s - [p] \uplus s')$. As a result, $t$ is not dead in $(N_1, [i])$. Combining the results of the two cases yields that $(N_1, [i])$ has no dead transitions, which proves Requirement *vi)* of Definition 6.2 and completes the proof of Theorem 7.13. $\square$

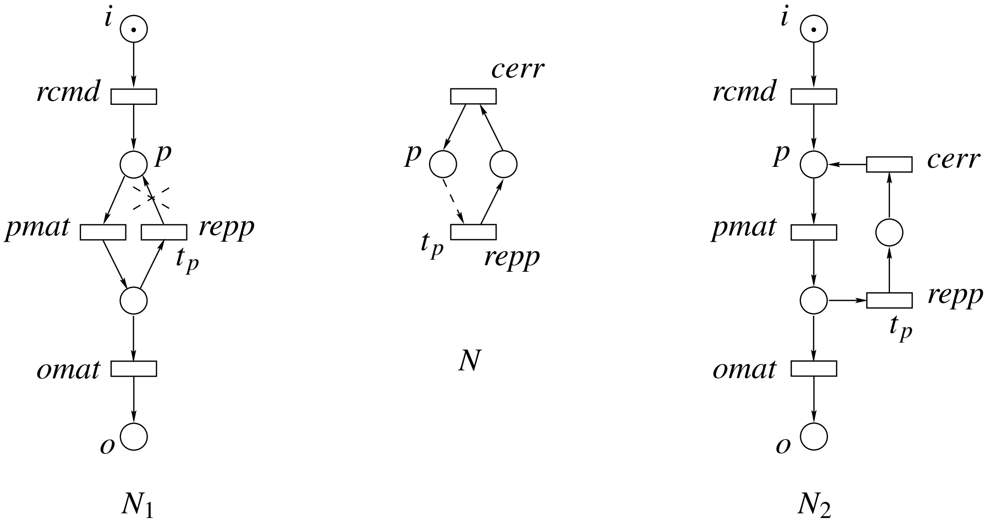



Figure 7.14: A transformation based on inheritance rule *PJ*.

**Example 7.15.** Consider again Example 6.15. The relationship between $N_2$ and $N_1$ can be proven by means of inheritance rule *PJ* given in Theorem 7.13. The transformation is illustrated in Figure 7.14. The arc in P/T net $N_1$ between transition $t_p$ and its output place $p$ is replaced by net $N$ minus the dashed arc between $p$ and $t_p$. The result is P/T net $N_2$. Since $(N, [p])$ is free-choice, live, and bounded and since method identifier *cerr* does not occur as a transition label in $N_1$, $N_2$ is a subclass under projection inheritance of $N_1$.

## 7.4 Inheritance rule *PT*

The third inheritance rule of this section preserves protocol inheritance. It is illustrated in Figure 7.16. The inheritance rule is inspired by Axiom *PT* which is the algebraic axiom of inheritance defined in Property 4.24. Inheritance rule *PT* can be used to extend a given life cycle with alternative branches of behavior. Let $N_0$ be an object life cycle. The extension of $N_0$ is based on a free-choice P/T net $N$ with a place $p_i$ such that

$(N, [p_i])$ is live and bounded. Nets $N_0$ and $N$ share two places $p_i$ and $p_o$ and no other nodes. Furthermore, $N$ contains a transition $y$ with $p_o$ as its only input place and $p_i$ as its only output place. The P/T net $N_1$ resulting from inheritance rule *PT* is defined as the union of $N_0$ and $N$ after the removal of transition $y$. Place $p_i$ functions as the entry point of the alternative branches of behavior added to $N_0$, whereas $p_o$ functions as the exit point. The requirement that $(N, [p_i])$ is live and bounded ensures that any token that transitions in $N$ consume from place $p_i$ is eventually returned to $p_o$. Two additional requirements guarantee that $N_1$ is an object life cycle. First, it is required that $N_0$ extended with a fresh transition $x$ with input place $p_i$ and output place $p_o$ is an object life cycle. Transition $x$ emulates the behavior of $N$ in $N_1$. Note that $x$ is only introduced to formulate the requirements of inheritance rule *PT*; it is not present in the original life cycle $N_0$, the extension $N$, or the subclass $N_1$. Second, recall that inheritance rule *PP* is a special case of inheritance rule *PT*. Rule *PT* reduces to rule *PP* when places $p_i$ and $p_o$ coincide. If places $p_i$ and $p_o$ are different, then it is assumed that the only input transition of place $p_i$ in $N$ is transition $y$ and that the only output transition of $p_o$ in $N$ is $y$. This assumption excludes the possibility of iterations beginning and ending in place $p_i$ or place $p_o$, thus guaranteeing that the modification of the original life cycle truly has place $p_i$ as its entry point and place $p_o$ as its exit point. A final requirement guarantees that $N_1$ is a subclass of $N_0$ under protocol inheritance: All transitions of $N$ with input place $p_i$ must have a visible label not appearing in the alphabet of $N_0$. This requirement means that transitions of $N$ with input places in $N_0$ act as guards, as explained in Section 4.2. Encapsulating the guards leads to a net whose behavior is identical to the behavior of the original life cycle, thus guaranteeing that $N_1$ is a subclass of $N_0$ under protocol inheritance.

The generality of inheritance rule *PT* is similar to the generality of inheritance rules *PP* and *PJ*. However, not all requirements of *PT* can be verified with the same efficiency as the requirements of the two earlier rules. The requirement that the extension of $N_0$ with the extra transition $x$ yields a life cycle is, theoretically, hard to verify. However, it is our expectation that the fact that it is known that $N_0$ is a life cycle alleviates the problem to such an extent that it is not a problem in practice. It is an interesting topic for further research to find an efficient algorithm to verify this requirement. It might be possible to replace the requirement with a simple *structural* requirement that can be verified efficiently and that does not compromise the generality of the rule too much.

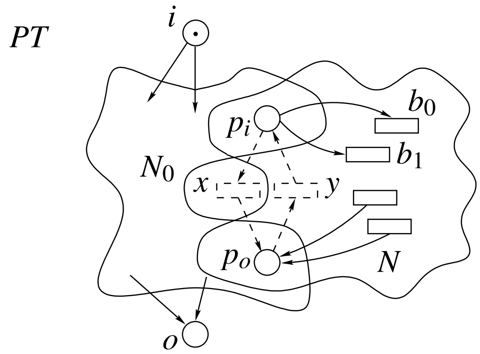



Figure 7.16: A protocol-inheritance rule.

**Theorem 7.17. (Protocol-inheritance rule** *PT***)** *Let $N_0 = (P_0, T_0, F_0, \ell_0)$ be an object life cycle in $\mathcal{O}$. Let $N = (P, T, F, \ell)$ be a* connected*, $M$-labeled,* free-choice *P/T net. Assume that $x \in U$ is a fresh identifier not appearing in $P_0 \cup T_0 \cup P \cup T$. If $N$ contains places $p_i, p_o \in P$ and a transition $y \in T$ such that*

*i)* $P_0 \cap P = \{p_i, p_o\}$, $T_0 \cap T = \emptyset$,

*ii)* $\boldsymbol{i}_N y = \{p_o\}$, $\boldsymbol{o}_N y = \{p_i\}$, $p_i \neq p_o \Rightarrow \boldsymbol{i}_N p_i = \boldsymbol{o}_N p_o = \{y\}$,

*iii)* $(\forall t : t \in \boldsymbol{o}_N p_i : \ell(t) \in E \backslash \alpha(N_0))$, *iv)* $(N, [p_i])$ *is live and bounded,*

*v)* $N_1 = N_0 \cup (P, T \backslash \{y\}, F \backslash \{(y, p_i), (p_o, y)\}, \ell \backslash \{(y, \ell(y))\})$ *is well defined, and*

*vi)* $N_0^x = (P_0, T_0 \cup \{x\}, F_0 \cup \{(p_i, x), (x, p_o)\}, \ell_0 \cup \{(x, \tau)\})$ *is an object life cycle,*

*then* $N_1$ *is an object life cycle in* $\mathcal{O}$ *such that* $N_1 \leq_{pt} N_0$.

**Proof.** The proof consists of two parts. In the first part, a branching bisimulation is given showing that $(N_0, [i])$ and $(N_1, [i])$ satisfy the requirement of Definition 6.13 *i)* (Protocol inheritance). In the second part, it is proven that $N_1$ is an object life cycle.

It must be shown that there exists an $H \subseteq E$ such that $\partial_H(N_1, [i]) \sim_b (N_0, [i])$. Property 6.20 suggests that $H$ can be chosen equal to $\alpha(N_1, [i]) \backslash \alpha(N_0)$. Let $\mathcal{R}$ be the relation $\{(\partial_H(N_1, u), (N_0, u)) \mid u \in \mathcal{B}(P_0)\}$. It is straightforward to prove that this relation is a branching bisimulation between $\partial_H(N_1, [i])$ and $(N_0, [i])$, as defined in Definition 2.8. The essence of the proof is that places of $N$ become never marked in $N_1$ after encapsulation of the guards and, thus, transitions of $N$ become never enabled. Clearly, $\partial_H(N_1, [i])\mathcal{R}(N_0, [i])$. Assume that $u$ is a marking in $\mathcal{B}(P_0)$, which implies that $\partial_H(N_1, u)\mathcal{R}(N_0, u)$. Three requirements have to be verified.

*i)* First, assume that $t \in T_0 \cup T$ is a transition of $N_1$ such that $\partial_H(N_1, u)[t\rangle$ and such that $\partial_H(N_1, u)\ [\alpha\rangle$ $\partial_H(N_1, u')$ where $\alpha = (\ell_0 \cup \ell)(t)$ and where marking $u' \in \mathcal{B}(P_0 \cup P)$ is equal to $u - \boldsymbol{i}_{N_1}t \uplus \boldsymbol{o}_{N_1}t$. Requirement *i)* of Definition 2.8 implies that it must be shown that there exist two markings $v', v'' \in \mathcal{B}(P_0)$ such that $(N_0, u) \Longrightarrow (N_0, v'')\ [(\alpha)\rangle\ (N_0, v')$, $\partial_H(N_1, u)\mathcal{R}(N_0, v'')$, and $\partial_H(N_1, u')\mathcal{R}(N_0, v')$. It follows from the fact that $(N, [p_i])$ is bounded and Property 5.17 (Characterization of boundedness) that every transition in $N$ has a nonempty preset. Thus, it follows from the definition of $H$, Definition 6.10 (Encapsulation), the fact that $u$ is a marking in $\mathcal{B}(P_0)$, and Requirement *iii)* of Theorem 7.17, that $t$ is a transition in $T_0$ and that $u'$ is a marking in $\mathcal{B}(P_0)$. Consequently, $(N_0, u)\ [\ell_0(t)\rangle\ (N_0, u')$ with $\ell_0(t) = \alpha$. Hence, assuming that $v'' = u$ and $v' = u'$, the proof requirement is satisfied.

*ii)* Second, assume that $t \in T_0$ is a transition of $N_0$ such that $(N_0, u)[t\rangle$ and such that $(N_0, u)\ [\ell_0(t)\rangle$ $(N_0, u')$ where $u' \in \mathcal{B}(P_0)$ is equal to $u - \boldsymbol{i}_{N_0}t \uplus \boldsymbol{o}_{N_0}t$. Clearly, this means that $\partial_H(N_1, u)\ [\ell_0(t)\rangle$ $\partial_H(N_1, u')$. Thus, also Requirement *ii)* of Definition 2.8 is satisfied.

*iii)* Third, Requirement *iii)* of Definition 2.8 states that the following two implications must be shown: $\downarrow \partial_H(N_1, u) \Rightarrow \Downarrow (N_0, u)$ and $\downarrow (N_0, u) \Rightarrow \Downarrow \partial_H(N_1, u)$. Assume $\downarrow \partial_H(N_1, u)$. It follows from Definition 6.1 (Semantics of $M$-labeled P/T nets) that $u = [o]$. Hence, also $\downarrow (N_0, u)$. Definition 2.7 ($\Downarrow$) yields immediately that $\Downarrow (N_0, u)$. The second implication is proven in the same way, which completes the first part of the proof.

The second part of the proof shows that $N_1$ is an object life cycle. It is straightforward to see that $N_1$ satisfies Requirement *i)* of Definition 6.2 (Object life cycle). If $p_i = p_o$, then Requirements *ii)* and *iii)* of Definition 6.2 follow from Requirement *vi)* of Theorem 7.17. If $p_i \neq p_o$, these two requirements follow from Requirements *ii)* and *vi)* of Theorem 7.17.

To prove that $N_1$ satisfies Requirements *iv)* and *v)* of Definition 6.2, we start by showing that any firing sequence of $(N_1, [i])$ preserves an invariant given below. Let $T_i$ be the set of transitions $\{t \in T \mid \boldsymbol{i}_N{=}\{p_i\}\}$ and $T_o$ the set of transitions $\{t \in T \mid \boldsymbol{o}_N{=}\{p_o\}\}$. Note that it easily follows from the fact that $(N, [p_i])$ is live and bounded that there are no other transitions in $T$ than those in $T_i$ with $p_i$ in their preset; similarly, no other transitions in $T$ than those in $T_o$ have $p_o$ in their postset. For any firing sequence $\sigma \in (T_0 \cup T)^*$ of $(N_1, [i])$, $\sigma(T_i)$ and $\sigma(T_o)$ denote the number of occurrences of transitions in $T_i$ in $\sigma$ and the number of occurrences of transitions in $T_o$ in $\sigma$, respectively. For any $\sigma \in (T_0 \cup T)^*$ and marking $s \in \mathcal{B}(P_0 \cup P)$ such that $(N_1, [i])\ [\sigma\rangle\ (N_1, s)$, it can be shown that $\sigma$ satisfies the invariant that $[p_o^{\sigma(T_i)-\sigma(T_o)}]$ is a home marking of $(N, s \upharpoonright P \backslash \{p_i, p_o\})$, where $[{p_o}^0] = \mathbf{0}$. (The reader should convince him- or herself that $\sigma(T_i) - \sigma(T_o)$ cannot be negative.) The proof is by induction on the length of $\sigma$.

Assume $\sigma = \varepsilon$, which means that $(N_1, [i])\,[\sigma\rangle\,(N_1, [i])$. It follows that $\sigma(T_i) = \sigma(T_o) = 0$ and $[i] \upharpoonright P\backslash\{p_i, p_o\} = \mathbf{0}$. Lemma 7.6 implies the desired result.

Assume that the desired property holds for a $\sigma \in (T_0 \cup T)^*$ and marking $s \in \mathcal{B}(P_0 \cup P)$ such that $(N_1, [i])\,[\sigma\rangle\,(N_1, s)$. Let $t \in T_0 \cup T$, $\sigma' = \sigma t$, and $s' = s - \boldsymbol{i}_{N_1} t \uplus \boldsymbol{o}_{N_1} t$. It must be shown that $[p_o^{\sigma'(T_i)-\sigma'(T_o)}]$ is a home marking of $(N, s' \upharpoonright P\backslash\{p_i, p_o\})$. Three cases can be distinguished.

First, assume that $t \notin T_i \cup T_o$ or $t \in T_i \cap T_o$. It follows that $\sigma'(T_i) - \sigma'(T_o) = \sigma(T_i) - \sigma(T_o)$. If $t \notin T\backslash(T_i \cup T_o)$, then $s' \upharpoonright P\backslash\{p_i, p_o\} = s \upharpoonright P\backslash\{p_i, p_o\}$ and the desired result follows immediately from the assumption that $[p_o^{\sigma(T_i)-\sigma(T_o)}]$ is a home marking of $(N, s \upharpoonright P\backslash\{p_i, p_o\})$. If $t \in T\backslash(T_i \cup T_o)$, then $(N, s \upharpoonright P\backslash\{p_i, p_o\})\,[t\rangle\,(N, s' \upharpoonright P\backslash\{p_i, p_o\})$ and the desired result follows from the assumption and Definition 5.25 (Home marking).

Second, assume that $t \in T_i\backslash T_o$. It follows that $\sigma'(T_i) - \sigma'(T_o) = \sigma(T_i) - \sigma(T_o) + 1$, $s' = s - [p_i] \uplus \boldsymbol{o}_N t$, and $s' \upharpoonright P\backslash\{p_i, p_o\} = s \upharpoonright P\backslash\{p_i, p_o\} \uplus \boldsymbol{o}_N t$. It can be shown that $[p_o^{\sigma(T_i)-\sigma(T_o)+1}]$ is a home marking of $(N, s \upharpoonright P\backslash\{p_i, p_o\} \uplus [p_o])$. Observe that $(N, [p_o])$ is live and bounded. If $\sigma(T_i) = \sigma(T_o)$, then, by Lemma 7.6 and the assumption that $[p_o^{\sigma(T_i)-\sigma(T_o)}]$ is a home marking of $(N, s \upharpoonright P\backslash\{p_i, p_o\})$, $s \upharpoonright P\backslash\{p_i, p_o\} = \mathbf{0}$. Lemma 7.7 implies that $[p_o]$ is a home marking of $(N, [p_o])$, which proves that $[p_o^{\sigma(T_i)-\sigma(T_o)+1}]$ is a home marking of $(N, s \upharpoonright P\backslash\{p_i, p_o\} \uplus [p_o])$ in this case. If $\sigma(T_i) > \sigma(T_o)$, then $[p_o^{\sigma(T_i)-\sigma(T_o)}] \neq \mathbf{0}$ and, by Lemma 7.6, $s \upharpoonright P\backslash\{p_i, p_o\} \neq \mathbf{0}$. Thus, it follows from the fact that $(N, [p_o])$ is live and Property 5.46 (Monotonicity of liveness) that also $(N, [p_o^{\sigma(T_i)-\sigma(T_o)}])$ is live. Definitions 5.21 (Liveness) and 5.25 (Home marking) yield that $(N, s \upharpoonright P\backslash\{p_i, p_o\})$ is live. The fact that $(N, [p_o])$ is bounded and Property 5.51 yield that $(N, s \upharpoonright P\backslash\{p_i, p_o\})$ is also bounded. Hence, the fact that $[p_o^{\sigma(T_i)-\sigma(T_o)}]$ is a home marking of $(N, s \upharpoonright P\backslash\{p_i, p_o\})$ and Property 5.54 (Monotonicity of home markings) yield also in this case that $[p_o^{\sigma(T_i)-\sigma(T_o)+1}]$ is a home marking of $(N, s \upharpoonright P\backslash\{p_i, p_o\} \uplus [p_o])$. Clearly, $(N, s \upharpoonright P\backslash\{p_i, p_o\} \uplus [p_o])\,[yt\rangle\,(N, s \upharpoonright P\backslash\{p_i, p_o\} \uplus \boldsymbol{o}_N t)$. Hence, Definition 5.25 (Home marking) yields that $[p_o^{\sigma(T_i)-\sigma(T_o)+1}]$ is a home marking of $(N, s \upharpoonright P\backslash\{p_i, p_o\} \uplus \boldsymbol{o}_N t)$, which is the desired result.

Third, assume that $t \in T_o\backslash T_i$. It follows that $\sigma'(T_i) - \sigma'(T_o) = \sigma(T_i) - \sigma(T_o) - 1$, $s' = s - \boldsymbol{i}_N t \uplus [p_o]$, and $s' \upharpoonright P\backslash\{p_i, p_o\} = s \upharpoonright P\backslash\{p_i, p_o\} - \boldsymbol{i}_N t$. It follows from the assumption that $[p_o^{\sigma(T_i)-\sigma(T_o)}]$ is a home marking of $(N, s \upharpoonright P\backslash\{p_i, p_o\})$ and the fact that $(N, s \upharpoonright P\backslash\{p_i, p_o\})\,[t\rangle\,(N, s \upharpoonright P\backslash\{p_i, p_o\} - \boldsymbol{i}_N t \uplus [p_o])$ that $[p_o^{\sigma(T_i)-\sigma(T_o)}]$ is also a home marking of $(N, s \upharpoonright P\backslash\{p_i, p_o\} - \boldsymbol{i}_N t \uplus [p_o])$. Recall that $(N, [p_o])$ is live and bounded. It follows from Lemma 7.10 that $[p_o^{\sigma(T_i)-\sigma(T_o)-1}]$ is a home marking of $(N, s \upharpoonright P\backslash\{p_i, p_o\} - \boldsymbol{i}_N t)$, which is the desired result.

Summarizing, it has been shown that any $\sigma \in (T_0 \cup T)^*$ and marking $s \in \mathcal{B}(P_0 \cup P)$ such that $(N_1, [i])[\sigma\rangle$ $(N_1, s)$ satisfy the invariant that $[p_o^{\sigma(T_i)-\sigma(T_o)}]$ is a home marking of $(N, s \upharpoonright P\backslash\{p_i, p_o\})$. Assume that $\sigma \in (T_0 \cup T)^*$ and marking $s \in \mathcal{B}(P_0 \cup P)$ are such that $(N_1, [i])\,[\sigma\rangle\,(N_1, s)$. Let $\sigma' \in T^*$ be a firing sequence of $N$ such that $(N, s \upharpoonright P\backslash\{p_i, p_o\})\,[\sigma'\rangle\,(N, [p_o^{\sigma(T_i)-\sigma(T_o)}])$. Lemma 7.11 implies that $\sigma'$ can be chosen in such a way that it contains no transitions from $T_i \cup \{y\}$. Hence, $(N_1, [i])\,[\sigma\sigma'\rangle\,(N_1, s \upharpoonright P_0 \uplus [p_o^{\sigma(T_i)-\sigma(T_o)}])$. Thus, $\sigma$ can be always be extended in such a way that all tokens in places in $P\backslash\{p_i, p_o\}$ are moved to place $p_o$. In other words, assuming that $\tilde{\sigma} \in (T_0 \cup \{x\})^*$ is the firing sequence obtained by replacing every occurrence of a transition in $T_i$ in $\sigma$ with transition $x$ and removing all other occurrences of transitions in $T$, it follows from the above invariance property that this firing sequence is enabled in $(N_0^x, [i])$ and $(N_0^x, [i])\,[\tilde{\sigma}\rangle\,(N_0^x, s \upharpoonright P_0 \uplus [p_o^{\sigma(T_i)-\sigma(T_o)}])$.

To prove that $N_1$ satisfies Requirement *iv)* of Definition 6.2, it must be shown that, for every reachable marking $s \in [N_1, [i]\rangle$, $o \in s$ implies $s = [o]$. Let $s \in [N_1, [i]\rangle$ be a reachable marking of $(N_1, [i])$ such that $o \in s$ and let $\sigma \in (T_0 \cup T)^*$ be a firing sequence such that $(N_1, [i])\,[\sigma\rangle\,(N_1, s)$. It follows from the above observation that $(N_0^x, [i])\,[\tilde{\sigma}\rangle\,(N_0^x, s \upharpoonright P_0 \uplus [p_o^{\sigma(T_i)-\sigma(T_o)}])$. Clearly, since $o \in s$ and $o \in P_0$, $o \in s \upharpoonright P_0 \uplus [p_o^{\sigma(T_i)-\sigma(T_o)}]$. It follows from the fact that $N_0^x$ is an object life cycle and Requirement *iv)* of Definition 6.2 that $s \upharpoonright P_0 \uplus [p_o^{\sigma(T_i)-\sigma(T_o)}] = [o]$. Assume $s > [o]$. It follows that $s \upharpoonright P_0 > [o]$ and/or $s \upharpoonright P\backslash\{p_i, p_o\} \neq \mathbf{0}$. Clearly, the former contradicts the facts that $o \in P_0$ and $s \upharpoonright P_0 \uplus [p_o^{\sigma(T_i)-\sigma(T_o)}] = [o]$.

According to the above invariance property and Lemma 7.6, the latter implies that $\sigma(T_i) - \sigma(T_o) > 0$. Since $o \in s$ and $o \in P_0$, it follows that $o \in s \restriction P_0$. The combination of $\sigma(T_i) - \sigma(T_o) > 0$ and $o \in s \restriction P_0$ contradicts the fact that $s \restriction P_0 \uplus [p_o^{\sigma(T_i)-\sigma(T_o)}] = [o]$, which means that also $s \restriction P \backslash \{p_i, p_o\} \neq \mathbf{0}$ leads to a contradiction. Hence, $s = [o]$, which completes this part of the proof.

To prove that $N_1$ satisfies Requirement *v)* (Termination option) of Definition 6.2 (Object life cycle), it must be shown that, for every reachable marking $s \in [N_1, [i]\rangle$, $[o] \in [N_1, s\rangle$. Let $s \in [N_1, [i]\rangle$ be a reachable marking of $(N_1, [i])$ and let $\sigma \in (T_0 \cup T)^*$ be a firing sequence such that $(N_1, [i])\ [\sigma\rangle\ (N_1, s)$. Recall from above that there exists a $\sigma' \in T^*$ such that $(N_1, s)[\sigma'\rangle(N_1, s \restriction P_0 \uplus [p_o^{\sigma(T_i)-\sigma(T_o)}])$ and that $(N_0^x, [i])[\tilde{\sigma}\rangle(N_0^x, s \restriction P_0 \uplus [p_o^{\sigma(T_i)-\sigma(T_o)}])$. It follows from the fact that $N_0^x$ is an object life cycle and Requirement *v)* (Termination option) of Definition 6.2 that there exists a $\sigma_0 \in (T_0 \cup \{x\})^*$ such that $(N_0^x, s \restriction P_0 \uplus [p_o^{\sigma(T_i)-\sigma(T_o)}])\ [\sigma_0\rangle\ (N_0^x, [o])$. It is possible to transform $\sigma_0$ into a firing sequence $\bar{\sigma_0} \in (T_0 \cup T)^*$ such that $(N_1, s \restriction P_0 \uplus [p_o^{\sigma(T_i)-\sigma(T_o)}])\ [\bar{\sigma_0}\rangle\ (N_1, [o])$. This transformation is defined as follows. Let $\sigma_1 \in (T \backslash \{y\})^*$ be a firing sequence such that $(N, [p_i])\ [\sigma_1\rangle\ (N, [p_o])$. It follows from the fact that $(N, [p_i])$ is live and bounded, Lemma 7.7, and Lemma 7.11 that such a firing sequence exists. Let $\bar{\sigma_0}$ be the firing sequence obtained by replacing every occurrence of $x$ by the sequence $\sigma_1$. Clearly, this firing sequence satisfies the requirement that $(N_1, s \restriction P_0 \uplus [p_o^{\sigma(T_i)-\sigma(T_o)}])\ [\bar{\sigma_0}\rangle\ (N_1, [o])$. Thus, $(N_1, [i])\ [\sigma\sigma'\bar{\sigma_0}\rangle\ (N_1, [o])$, which completes the proof that $(N_1, [i])$ satisfies Requirement *v)* (Termination option) of Definition 6.2.

It remains to prove Requirement *vi)* of Definition 6.2 saying that $(N_1, [i])$ contains no dead transitions. The proof is very similar to – but simpler than – the corresponding part of the proof of Theorem 7.13. The only differences are that $N_0^x$ plays the role of $N_0$ and that the transformation of firing sequences is the one introduced in the previous part of the current proof. □

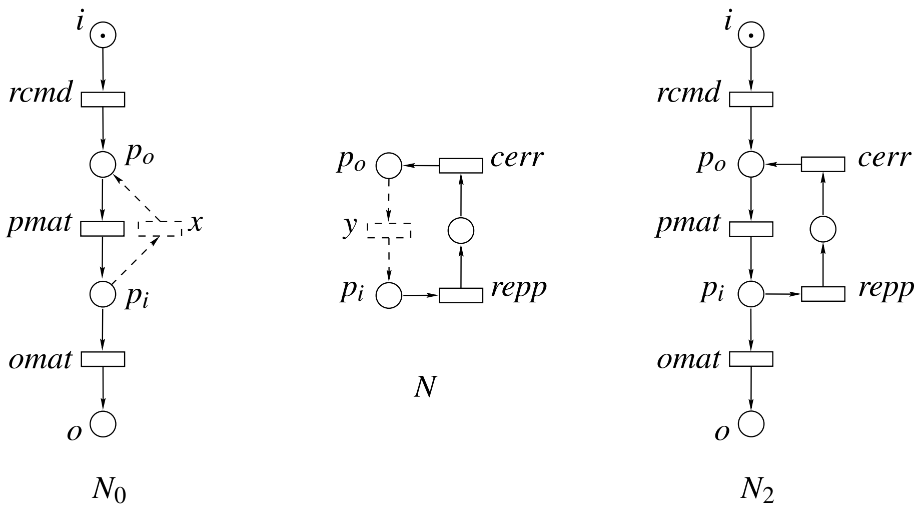



Figure 7.18: A transformation based on inheritance rule *PT*.

**Example 7.19.** Consider Example 6.15. The protocol-inheritance relationships between life cycles $N_1$ and $N_0$ and between $N_2$ and $N_0$ can be proven by means of inheritance rule *PT*. In both cases, method *repp* acts as the guard. Figure 7.18 illustrates the transformation from $N_0$ to $N_2$ in detail. It is not difficult to verify that $N_0$ extended with the fresh transition $x$ is an object life cycle. Furthermore, $(N, [p_i])$ is clearly live and bounded. Thus, according to Theorem 7.17, the union of $N_0$ and $N$ minus the dashed transition and arcs, which equals $N_2$, is a subclass of $N_0$.

## 7.5 Inheritance rule $PJ3$

The last inheritance rule of this section corresponds to the addition of parallel behavior to a life cycle. To formulate this rule, the following auxiliary definition is needed. A place of a marked P/T net is said to be *redundant* or *implicit* if and only if it does not depend on the number of tokens in the place whether any of its output transitions is enabled by some reachable marking.

**Definition 7.20. (Implicit place)** Let $(N, s)$ with $N = (P, T, F, \ell)$ be a marked, $M$-labeled P/T net. A place $p \in P$ is called *implicit* in $(N, s)$ if and only if, for all reachable markings $s' \in [N, s\rangle$ and transitions $t \in \boldsymbol{o}p$, $s' \geq \boldsymbol{i}t \backslash \{p\} \Rightarrow s' \geq \boldsymbol{i}t$.

Implicit places and their properties have been studied in [18, 24]. In the context of this paper, only the following property of implicit places is important. Implicit places different from the output place may be removed from an object life cycle yielding a P/T net which is branching bisimilar to the original life cycle. It is even possible to prove a stronger result, namely that the resulting P/T net is again an object life cycle. However, since this stronger result is not needed, only the first mentioned result is proven.

**Lemma 7.21.** *Let $N = (P, T, F, \ell)$ be an object life cycle containing a place $p \in P\backslash\{o\}$ that is implicit in $(N, [i])$. If $N_0$ is the subnet of $N$ generated by the set of nodes $P\backslash\{p\} \cup T$, as defined in Definition 5.27, then $(N, [i]) \sim_b (N_0, [i])$.*

**Proof.** Let $\mathcal{R}$ be the relation $\{((N, u), (N_0, v)) \mid u \in [N, [i]\rangle \wedge v \in [N_0, [i]\rangle \wedge u \restriction (P\backslash\{p\}) = v\}$. It is not difficult to verify that, since $p$ is implicit in $(N, [i])$, $p \neq i$. As a result, $(N, [i])\mathcal{R}(N_0, [i])$. It remains to be shown that $\mathcal{R}$ is a branching bisimulation. Assume that $u \in [N, [i]\rangle$ and $v \in [N_0, [i]\rangle$ are markings such that $(N, u)\mathcal{R}(N_0, v)$.

*i*) Assume that $t \in T$ is such that $(N, u)[t\rangle$ and $(N, u)\, [\ell(t)\rangle\, (N, u')$ with $u' = u - \boldsymbol{i}_N t \uplus \boldsymbol{o}_N t$. It follows immediately from the fact that $u \restriction (P\backslash\{p\}) = v$, the definition of $N_0$, and Definition 5.7 (Firing rule) that $(N_0, v)\, [\ell(t)\rangle\, (N_0, u' \restriction (P\backslash\{p\}))$. Hence, it easily follows that Requirement *i*) of Definition 2.8 (Branching bisimilarity) is satisfied.

*ii*) Assume that $t \in T$ is such that $(N_0, v)[t\rangle$ and $(N_0, v)[\ell(t)\rangle(N_0, v')$ with $v' = v - \boldsymbol{i}_{N_0} t \uplus \boldsymbol{o}_{N_0} t$. It follows from the facts that $u \restriction (P\backslash\{p\}) = v$ and that $p$ is implicit in $(N, [i])$, Definition 7.20 (Implicit place), the definition of $N_0$, and Definition 5.7 (Firing rule) that $(N, u)\, [\ell(t)\rangle\, (N, v' \uplus ((u - \boldsymbol{i}_N t \uplus \boldsymbol{o}_N t) \restriction \{p\}))$. Thus, it is not difficult to verify that also Requirement *ii*) of Definition 2.8 is satisfied.

*iii*) First, assume that $\downarrow(N, u)$. It follows from Definition 6.1 (Semantics of $M$-labeled P/T nets) that $u = [o]$. Since $u \restriction (P\backslash\{p\}) = v$ and $p \neq o$, also $v = [o]$. Thus, $\downarrow(N_0, v)$, which, by Definition 2.7 ($\Downarrow$), implies that $\Downarrow(N_0, v)$. Second, assume that $\downarrow(N_0, v)$. It follows that $v = [o]$. Since $u \restriction (P\backslash\{p\}) = v$ and $p \neq o$, $u = [o] \uplus (u \restriction \{p\})$. Since $N$ is an object life cycle, it follows from Requirement *iv*) (Proper termination) of Definition 6.2 (Object life cycle) and the fact that $u \in [N, [i]\rangle$ that $u = [o]$. As a result, $\Downarrow(N, u)$, which implies that also Requirement *iii*) of Definition 2.8 is satisfied.

□

Lemma 7.21 does not carry over to arbitrary P/T nets, as the reader familiar with the literature on implicit places might expect. The reason is the specific semantics for P/T nets defined in Definition 6.1. The termination predicate in this semantics says that a P/T net terminates if and only if it reaches a marking with a single token in output place $o$. To illustrate the problem, consider a P/T net which reaches a marking with a single token in $o$ plus some additional tokens in an implicit place. Clearly, according to the semantics, this net has not terminated successfully. However, the P/T net that results from removing the implicit place does terminate in the corresponding marking, which means that the two nets cannot be branching bisimilar.

The main result of this subsection is the fourth inheritance rule. It is inspired by and carries the same name as the algebraic axiom of inheritance $PJ3$, given in Property 4.27. Inheritance rule $PJ3$ is formalized in Theorem 7.23 given below. It shows under what restrictions it is allowed to extend an object life cycle with a parallel branch of behavior. The result of rule *PJ*3 is a subclass of the original life cycle under projection inheritance. It is illustrated in Figure 7.22. As before, $N_0$ is the original life cycle. Again, the modification of $N_0$ is based on a free-choice P/T net $N$ containing a place $p$ such that $(N, [p])$ is live and bounded. The two net structures $N_0$ and $N$ share two transitions $t_i$ and $t_o$. In $N$, place $p$ is the only input place of $t_i$ and the only output place of $t_o$. Furthermore, $p$ has no other input or output transitions. The net structure $N_1$ resulting from inheritance rule *PJ*3 is defined as the union of $N_0$ and $N$ after the removal of place $p$. These assumptions mean that transitions $t_i$ and $t_o$ function as the input and output transition of the extra parallel branch modeled by $N$. The basic idea is that the P/T net $(N_1, [i])$ satisfies the property that every firing of transition $t_i$ is eventually followed by a firing of transition $t_o$. The requirement that $(N, [p])$ is free-choice, live, and bounded guarantees that each time transition $t_i$ fires the resulting tokens in places of $N$ can be moved to the input places of transition $t_o$ in $N$ by only firing transitions of $N$ other than $t_i$ and $t_o$. In addition, to guarantee that $(N_1, [i])$ satisfies the desired property, also $(N_0, [i])$ must be such that every firing of $t_i$ is followed by exactly one firing of $t_o$. To achieve this goal, assume that $N_0$ is extended with a place $q$ with $t_i$ as its only input transition and $t_o$ as its only output transition. Requiring that place $q$ is implicit in this extension guarantees that a firing of transition $t_o$ is always preceded by a firing of $t_i$. It is not difficult to see that the number of tokens in $q$ corresponds to the number of firings of transition $t_i$ which have not yet been followed by a firing of $t_o$. To guarantee that $(N_0, [i])$ cannot terminate without firing $t_o$ as many times as $t_i$, the extension of $N_0$ with place $q$ must be such that it cannot put a token in place $o$ while leaving tokens in $q$. Clearly, this is achieved when the extension of $N_0$ with $q$ yields another life cycle. The combination of the requirements on $N_0$ and $N$ implies that $N_1$ is an object life cycle satisfying the property that every firing of $t_i$ is eventually followed by a firing of $t_o$. The attentive reader might notice the duality between inheritance rules *PT* of the previous subsection and *PJ*3.

The generality of inheritance rule *PJ*3 is similar to the generality of the other inheritance rules of this section. However, the requirements of *PJ*3 cannot be verified with the same efficiency as the requirements of the other rules. The two requirements concerning the extension of $N_0$ with place $q$ are, in theory, hard to verify. However, the requirement that $q$ is implicit in the extension of $N_0$ can be slightly strengthened by requiring that $q$ is *structurally* implicit [18, 24]. This hardly compromises the generality of the rule from a practical point of view and it is possible to verify whether a place is structurally implicit in polynomial time [24]. The requirement that the extension of $N_0$ with $q$ yields another life cycle is theoretically also hard to verify. However, the input that $N_0$ is a life cycle and that $q$ is implicit in the extension might alleviate this problem. The applicability of the rule has to be tested in practice. It might also be an interesting question for future research to find the exact complexity of this problem.

As a final remark, note that the proof of Theorem 7.23 is similar to the proof of Theorem 7.13. Therefore, it is not given in as much detail and with the same amount of explanation as the proof of Theorem 7.13. However, all essential aspects of the proof are given in detail.

**Theorem 7.23. (Projection-inheritance rule** *PJ*3**)** *Let $N_0 = (P_0, T_0, F_0, \ell_0)$ be an object life cycle in $\mathcal{O}$. Let $N = (P, T, F, \ell)$ be a* connected, *$M$-labeled,* free-choice *P/T net. Assume that $q \in U$ is a fresh identifier not appearing in $P_0 \cup T_0 \cup P \cup T$. If $N$ contains a place $p \in P$ and transitions $t_i, t_o \in T$ such that*

*i)* $P_0 \cap P = \emptyset$, $T_0 \cap T = \{t_i, t_o\}$,

*ii)* $\boldsymbol{i}_N p = \{t_o\}, \boldsymbol{o}_N p = \{t_i\}, \boldsymbol{i}_N t_i = \{p\}, \boldsymbol{o}_N t_o = \{p\}$,

*iii)* $(\forall t : t \in T \backslash T_0 : \ell(t) \notin \alpha(N_0))$,

*iv)* $(N, [p])$ *is live and bounded,*

*v)* $N_1 = N_0 \cup (P \backslash \{p\}, T, F \backslash \{(p, t_i), (t_o, p)\}, \ell)$ *is well defined,*

*vi)* $q$ *is implicit in* $(N_0^q, [i])$ *with* $N_0^q = (P_0 \cup \{q\}, T_0, F_0 \cup \{(t_i, q), (q, t_o)\}, \ell_0)$, *and*

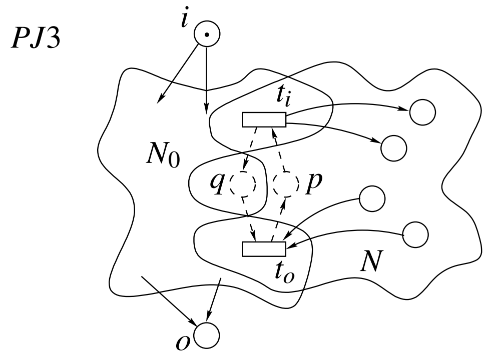



Figure 7.22: A projection-inheritance rule.

*vii*) $N_0^q$ *is an object life cycle,*

*then* $N_1$ *is an object life cycle in* $\mathcal{O}$ *such that* $N_1 \leq_{pj} N_0$.

**Proof.** The proof goes along the same lines as the proof of Theorem 7.13. Let $I = \alpha(N_1, [i])\backslash\alpha(N_0)$. It must be shown that $\tau_I(N_1, [i])$ is branching bisimilar to $(N_0, [i])$. According to Lemma 7.21, it suffices to show that $\tau_I(N_1, [i])$ is branching bisimilar to $(N_0^q, [i])$. Let $\mathcal{R} = \{(\tau_I(N_1, u), (N_0^q, v)) \mid u \in [N_1, [i]\rangle \wedge v \in [N_0^q, [i]\rangle \wedge u \lceil P_0 = v \lceil P_0 \wedge [p^{v(q)}]$ is a home marking of $(N, u \lceil P)\}$. Note that $p \neq i$, because $P_0 \cap P = \emptyset$. Lemma 7.6 yields that $\tau_I(N_1, [i])\mathcal{R}(N_0^q, [i])$. To prove that $\mathcal{R}$ is a branching bisimulation, assume that $u \in [N_1, [i]\rangle$ and $v \in [N_0^q, [i]\rangle$ are such that $\tau_I(N_1, u)\mathcal{R}(N_0^q, v)$. That is, $u \lceil P_0 = v \lceil P_0$ and $[p^{v(q)}]$ is a home marking of $(N, u \lceil P)$.

*i*) Assume that $t \in T_0 \cup T$ is such that $\tau_I(N_1, u)[t\rangle$ and $\tau_I(N_1, u)\ [\alpha\rangle\ \tau_I(N_1, u')$ with $\alpha = (\ell_0 \cup \ell)(t)$ and $u' = u - \boldsymbol{i}_{N_1} t \uplus \boldsymbol{o}_{N_1} t$. It must be shown that there exist $v', v'' \in \mathcal{B}(P_0 \cup \{q\})$ such that *a*) $(N_0^q, v) \Longrightarrow (N_0^q, v'')\ [(\alpha)\rangle\ (N_0^q, v')$, *b*) $\tau_I(N_1, u)\mathcal{R}(N_0^q, v'')$, and *c*) $\tau_I(N_1, u')\mathcal{R}(N_0^q, v')$.
First, assume that $t \in T\backslash T_0$. The argument in this case is identical to the argument used in the proof of the analogue case of Theorem 7.13.
Second, assume that $t \in T_0$. Let $v'' = v$ and $v' = v - \boldsymbol{i}_{N_0^q} t \uplus \boldsymbol{o}_{N_0^q} t$. To prove Requirement *a*), observe that $\boldsymbol{i}_{N_0^q} t = \boldsymbol{i}_{N_1} t \cap P_0$ unless $t = t_o$. In the latter case, $\boldsymbol{i}_{N_0^q} t = (\boldsymbol{i}_{N_1} t \cap P_0) \cup \{q\}$. Requirement *a*) easily follows from the assumptions and the fact that $q$ is implicit in $(N_0^q, [i])$. Requirement *b*) follows immediately from the assumptions. Finally, Requirement *c*) is proven as follows. It follows from the assumptions and Requirement *a*) that $u' \in [N_1, [i]\rangle$ and $v' \in [N_0^q, [i]\rangle$. The fact that $u \lceil P_0 = v \lceil P_0$ yields that $u' \lceil P_0 = v' \lceil P_0$. It remains to be shown that $[p^{v'(q)}]$ is a home marking of $(N, u' \lceil P)$. Recall that $[p^{v(q)}]$ is a home marking of $(N, u \lceil P)$. Three cases can be distinguished. First, assume that $t \notin \{t_i, t_o\}$. It follows easily from the assumptions that $v'(q) = v(q)$ and $u' \lceil P = u \lceil P$, which immediately proves the requirement. Second, assume that $t = t_i$. It follows from the requirements of Theorem 7.23 that $v'(q) = v(q) + 1$ and $u' \lceil P = u \lceil P \uplus \boldsymbol{o}_N t$. It can be shown that $[p^{v(q)+1}]$ is a home marking of $(N, u \lceil P \uplus [p])$. It follows from the assumption that $[p^{v(q)}]$ is a home marking of $(N, u \lceil P)$ and Lemma 7.6 that $[p^{v(q)}] = \mathbf{0}$ if and only if $u \lceil P = \mathbf{0}$. Hence, if $[p^{v(q)}] = u \lceil P = \mathbf{0}$, then the desired result that $[p^{v(q)+1}]$ is a home marking of $(N, u \lceil P \uplus [p])$ follows from Lemma 7.7. If $[p^{v(q)}] \neq \mathbf{0}$ and $u \lceil P \neq \mathbf{0}$, it follows from the fact that $(N, [p])$ is live and Property 5.46 (Monotonicity of liveness) that $(N, [p^{v(q)}])$ is live. Definitions 5.21 (Liveness) and 5.25 (Home marking) yield that $(N, u \lceil P)$ is live. The fact that $(N, [p])$ is bounded and Property 5.51 yield that $(N, u \lceil P)$ is also bounded. Hence, in this case, the desired result that $[p^{v(q)+1}]$ is a home marking of $(N, u \lceil P \uplus [p])$ follows from Property 5.54 (Monotonicity of home markings). Since $\boldsymbol{i}_N t_i = \{p\}$, it follows from the assumption that $t = t_i$ and Definitions 5.7 (Firing rule) and 5.25 (Home marking) that $[p^{v(q)+1}]$ is

a home marking of $(N, u \restriction P \uplus \boldsymbol{o}_N t)$, which proves Requirement *c*) in this case. Third, assume that $t = t_o$. It follows that $v'(q) = v(q) - 1$ and $u' \restriction P = u \restriction P - \boldsymbol{i}_N t$. Definition 5.7 (Firing rule) yields that $(N, u \restriction P)\ [\ell(t)\rangle\ (N, u \restriction P - \boldsymbol{i}_N t \uplus [p])$. The fact that $[p^{v(q)}]$ is a home marking of $(N, u \restriction P)$ and Definition 5.25 (Home marking) imply that $[p^{v(q)}]$ is a home marking of $(N, u \restriction P - \boldsymbol{i}_N t \uplus [p])$. It follows from Lemma 7.10 that $[p^{v(q)-1}]$ is a home marking of $(N, u \restriction P - \boldsymbol{i}_N t)$, which proves the requirement also in this case.

*ii*) Assume that $t \in T_0$ is such that $(N_0^q, v)[t\rangle$ and $(N_0^q, v)\ [\alpha\rangle\ (N_0^q, v')$ with $\alpha = \ell_0(t)$ and $v' = v - \boldsymbol{i}_{N_0^q} t \uplus \boldsymbol{o}_{N_0^q} t$. It must be shown that there exist $u', u'' \in \mathcal{B}(P_0 \cup P \backslash \{p\})$ such that *a*) $\tau_I(N_1, u) \Longrightarrow \tau_I(N_1, u'')\ [(\alpha)\rangle\ \tau_I(N_1, u')$, *b*) $\tau_I(N_1, u'')\mathcal{R}(N_0^q, v)$, and *c*) $\tau_I(N_1, u')\mathcal{R}(N_0^q, v')$. Two cases need to be distinguished.

First, assume $t \neq t_o$. Let $u'' = u$ and $u' = u - \boldsymbol{i}_{N_1} t \uplus \boldsymbol{o}_{N_1} t$. It follows from the assumptions that $\boldsymbol{i}_{N_1} t = \boldsymbol{i}_{N_0^q} t$, which proves Requirement *a*). Requirement *b*) follows immediately from the assumptions. To prove Requirement *c*), observe that $v' \in [N_0^q, [i]\rangle$ and $u' \in [N_1, [i]\rangle$. Two cases are distinguished. If $t \neq t_i$, then $\boldsymbol{o}_{N_1} t = \boldsymbol{o}_{N_0^q} t$. Thus, $u' \restriction P_0 = v' \restriction P_0$, $v'(q) = v(q)$, and $u' \restriction P = u \restriction P$. Requirement *c*) follows immediately. If $t = t_i$, then $\boldsymbol{o}_{N_1} t = \boldsymbol{o}_{N_0^q} t \backslash \{q\} \cup \boldsymbol{o}_N t$. It follows that $u' \restriction P_0 = v' \restriction P_0$, $v'(q) = v(q) + 1$, $u' \restriction P = u \restriction P \uplus \boldsymbol{o}_N t$. It remains to be shown that $[p^{v'(q)}]$ is a home marking of $(N, u' \restriction P)$. Using the same argument as in the previous part of the proof, it follows from the assumption that $[p^{v(q)}]$ is a home marking of $(N, u \restriction P)$ that $[p^{v(q)+1}]$ is a home marking of $(N, u \restriction P \uplus [p])$. Since $(N, u \restriction P \uplus [p])\ [t_i\rangle\ (N, u \restriction P \uplus \boldsymbol{o}_N t)$, Definition 5.25 (Home marking) yields that $[p^{v(q)+1}]$ is a home marking of $(N, u \restriction P \uplus \boldsymbol{o}_N t)$, which proves the requirement.

Second, assume $t = t_o$. Let $u'' = v \restriction P_0 \uplus (\uplus r : r \in \boldsymbol{i}_N t : [r^{v(q)}])$ and $u' = v' \restriction P_0 \uplus (\uplus r : r \in \boldsymbol{i}_N t : [r^{v'(q)}])$. Since $t = t_o$, it follows that $\boldsymbol{i}_{N_1} t = \boldsymbol{i}_{N_0^q} t \backslash \{q\} \cup \boldsymbol{i}_N t$ and $\boldsymbol{o}_{N_1} t = \boldsymbol{o}_{N_0^q} t$. The proof of Requirement *a*) is as follows. The fact that $[p^{v(q)}]$ is a home marking of $(N, u \restriction P)$ and Lemma 7.11 imply that there exists a firing sequence $\sigma \in (T \backslash \{t_i\})^*$ such that $(N, u \restriction P)\ [\sigma\rangle\ (N, [p^{v(q)}])$. Note that it follows from the definition of $N_1$ that $u(p) = 0$. Consequently, $\sigma$ can be chosen such that $\sigma = \sigma_0 \sigma_1$ where $t_o \notin \sigma_0$ and where $\sigma_1$ is of length $v(q)$ with, for all $i$, $0 \leq i < v(q)$, $\sigma_1(i) = t_o$. As a result, $(N, u \restriction P)\ [\sigma_0\rangle\ (N, (\uplus r : r \in \boldsymbol{i}_N t : [r^{v(q)}]))$. It follows from the fact that $u \restriction P_0 = v \restriction P_0$ that $\tau_I(N_1, u)\ [\sigma_0\rangle\ \tau_I(N_1, u'')$. Requirement *iii*) of Theorem 7.23 yields that $\tau_I(N_1, u) \Longrightarrow \tau_I(N_1, u'')$. It follows from the observation that $\boldsymbol{i}_{N_1} t = \boldsymbol{i}_{N_0^q} t \backslash \{q\} \cup \boldsymbol{i}_N t$, $\boldsymbol{o}_{N_1} t = \boldsymbol{o}_{N_0^q} t$, and $v'(q) = v(q) - 1$ that $\tau_I(N_1, u'')\ [\ell_0(t)\rangle\ \tau_I(N_1, u')$, which proves Requirement *a*). To prove Requirement *b*), observe that $u'' \restriction P_0 = v \restriction P_0$. It remains to be shown that $[p^{v(q)}]$ is a home marking of $(N, u'' \restriction P)$. In the proof of Requirement *a*), it has been shown that $u'' \restriction P \in [N, u \restriction P\rangle$. Thus, the fact that $[p^{v(q)}]$ is a home marking of $(N, u \restriction P)$ implies that $[p^{v(q)}]$ is also a home marking of $(N, u'' \restriction P)$. To prove Requirement *c*), note that $u' \restriction P_0 = v' \restriction P_0$. Therefore, it suffices to show that $[p^{v'(q)}]$ is a home marking of $(N, u' \restriction P)$. Observe that $v'(q) = v(q) - 1$ and $u' \restriction P = u'' - \boldsymbol{i}_N t$. Since $(N, u'' \restriction P)\ [t\rangle\ (N, u'' \restriction P - \boldsymbol{i}_N t \uplus [p])$, it follows from the fact that $[p^{v(q)}]$ is a home marking of $(N, u'' \restriction P)$ that also $[p^{v(q)}]$ is a home marking of $(N, u'' \restriction P - \boldsymbol{i}_N t \uplus [p])$. Thus, Lemma 7.10 implies that $[p^{v(q)-1}]$ is a home marking of $(N, u'' \restriction P - \boldsymbol{i}_N t)$, which proves Requirement *c*).

*iii*) The argument to prove Requirement *iii*) of Definition 2.8 (Branching bisimilarity) is identical to the argument used in the corresponding case of the proof of Theorem 7.13 under the assumption that $p \neq o$. Note that this assumption is valid because Requirement *i*) of Theorem 7.23 states that $P_0 \cap P = \emptyset$.

It remains to be shown that $N_1$ is an object life cycle. It is clear that $N_1$ satisfies Requirements *i*) through *iii*) of Definition 6.2 (Object life cycle). The proof that $N_1$ satisfies Requirement *iv*) (Proper termination) of Definition 6.2 is almost identical to the corresponding part of the proof of Theorem 7.13, again, under the assumption that $p \neq o$.

To prove that $N_1$ satisfies Requirement *v*) (Termination option) of Definition 6.2, assume that $u \in [N_1, [i]\rangle$. Since $\mathcal{R}$ is a branching bisimulation, Lemma 6.17 yields that there exists a $v \in [N_0^q, [i]\rangle$ such that $u \upharpoonright P_0 = v \upharpoonright P_0$ and $[p^{v(q)}]$ is a home marking of $(N, u \upharpoonright P)$. As shown in part *ii*) of the proof that $\mathcal{R}$ is a branching bisimulation, it is possible to construct a firing sequence $\sigma \in (T\backslash\{t_i, t_o\})^*$ such that $(N_1, u)\ [\sigma\rangle\ (N_1, v \upharpoonright P_0 \uplus (\uplus r : r \in \boldsymbol{i}_N t_o : [r^{v(q)}]))$. It follows from the fact that $N_0^q$ is a life cycle that there is a firing sequence $\sigma_0 \in {T_0}^*$ such that $(N_0^q, v)\ [\sigma_0\rangle\ (N_0^q, [o])$. It is possible to transform $\sigma_0$ into a sequence $\bar{\sigma_0} \in (T_0 \cup T)^*$ such that $(N_1, v \upharpoonright P_0 \uplus (\uplus r : r \in \boldsymbol{i}_N t_o : [r^{v(q)}]))\ [\bar{\sigma_0}\rangle\ (N_1, [o])$, proving that $N_1$ satisfies Requirement *v*) of Definition 6.2. The transformation is inductively defined as follows. If $t_i \notin \sigma_0$, then $\bar{\sigma_0} = \sigma_0$. It is straightforward to see that this firing sequence satisfies the requirement. If $t_i \in \sigma_0$, then assume that $\sigma_0 = \sigma_1 t_i \sigma_2$ with $t_i \notin \sigma_1$. Let $v', v'' \in \mathcal{B}(P_0 \cup \{q\})$ be such that $(N_0^q, v)[\sigma_1\rangle(N_0^q, v')[t_i\rangle(N_0^q, v'')[\sigma_2\rangle(N_0^q, [o])$. It follows that $(N_1, v \upharpoonright P_0 \uplus (\uplus r : r \in \boldsymbol{i}_N t_o : [r^{v(q)}]))\ [\sigma_1\rangle\ (N_1, v' \upharpoonright P_0 \uplus (\uplus r : r \in \boldsymbol{i}_N t_o : [r^{v'(q)}]))\ [t_i\rangle\ (N_1, v'' \upharpoonright P_0 \uplus (\uplus r : r \in \boldsymbol{i}_N t_o : [r^{v'(q)}]) \uplus \boldsymbol{o}_N t_i)$. Lemma 7.7 states that $[p]$ is a home marking of $(N, [p])$. Since $(N, [p])\ [t_i\rangle\ (N, \boldsymbol{o}_N t_i)$, it follows that $[p]$ is also a home marking of $(N, \boldsymbol{o}_N t_i)$. Lemma 7.11 and the fact that $\boldsymbol{i}_N p = \{t_o\}$ imply that there exists a firing sequence $\sigma_3 \in (T\backslash\{t_i, t_o\})^*$ such that $(N, \boldsymbol{o}_N t_i)\ [\sigma_3\rangle\ (N, \boldsymbol{i}_N t_o)$. It follows that $(N_1, v'' \upharpoonright P_0 \uplus (\uplus r : r \in \boldsymbol{i}_N t_o : [r^{v'(q)}]) \uplus \boldsymbol{o}_N t_i)\ [\sigma_3\rangle\ (N_1, v'' \upharpoonright P_0 \uplus (\uplus r : r \in \boldsymbol{i}_N t_o : [r^{v'(q)+1}]))$. Observe that $v''(q) = v'(q) + 1$. Hence, $(N_1, v \upharpoonright P_0 \uplus (\uplus r : r \in \boldsymbol{i}_N t_o : [r^{v(q)}]))\ [\sigma_1 t_i \sigma_3\rangle\ (N_1, v'' \upharpoonright P_0 \uplus (\uplus r : r \in \boldsymbol{i}_N t_o : [r^{v''(q)}]))$. Choosing $\bar{\sigma_0} = \sigma_1 t_i \sigma_3 \bar{\sigma_2}$ yields that $(N_1, v \upharpoonright P_0 \uplus (\uplus r : r \in \boldsymbol{i}_N t_o : [r^{v(q)}]))\ [\bar{\sigma_0}\rangle\ (N_1, [o])$, completing the proof.

The proof that $N_1$ also satisfies Requirement *vi*) of Definition 6.2 is very similar to the corresponding part of the proof of Theorem 7.13. The only differences are that $t_i$ plays the role of $t_p$ and that the transformation of firing sequences is the one introduced in the previous part of the proof. □

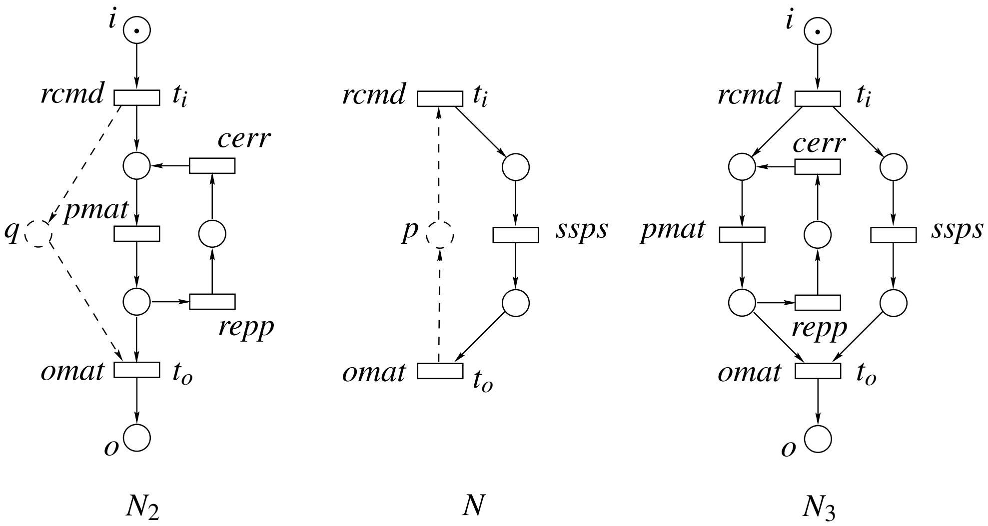



Figure 7.24: A transformation based on inheritance rule *PJ*3.

**Example 7.25.** Let us return one more time to Example 6.15. The subclass relationship between $N_3$ and $N_2$ can be proven by means of inheritance rule *PJ*3 of Theorem 7.23. The transformation is depicted in Figure 7.24. Let $N_2^q$ denote the P/T net $N_2$ extended with place $q$. According to Definition 7.20, place $q$ is implicit in $(N_2^q, [i])$. In addition, $N_2^q$ is an object life cycle. This means that the extension of $N_2$ with place $q$ satisfies the requirements of Theorem 7.23. As explained, these requirements guarantee that, in $(N_2, [i])$, each firing of transition $t_i$ is ultimately followed by exactly one firing of transition $t_o$. The transformation of $N_2$ is based on the connected, free-choice P/T net $N$. Since $(N, [p])$ is live and bounded and since *ssps* is a method identifier not occurring in $N_2$, also $N$ satisfies the requirements of Theorem 7.23. The union of $N_2$ and $N$ minus place $p$ and its incoming and outgoing arcs yields P/T net $N_3$. Thus, it may be concluded from Theorem 7.23 that object life cycle $N_3$ is a subclass under projection inheritance of life cycle $N_2$.

## 7.6 Concluding remarks

So far in this section, four inheritance rules have been presented to construct subclasses of object life cycles under different forms of inheritance. The rules correspond to design constructs that are often used in practice, namely iteration, sequential composition, choice, and parallel composition. As explained in the introduction to this section, the rules are compromises between generality and efficiency. Inheritance rules *PP* and *PJ*, corresponding to iteration and sequential composition respectively, are reasonably general and very efficient. Rules *PT* and *PJ*3, corresponding to the choice construct and parallel composition, are also reasonably general, but, at least in theory, they are not very efficient. However, it is our belief that, in practice, the theoretical complexity does not cause problems; of course, this claim has to be tested in non-trivial applications. Nevertheless, it is interesting to briefly consider generalizations and restrictions of the rules.

In each of the four inheritance rules, the transformation of the original life cycle is based on a live and bounded free-choice P/T net. An interesting question is whether it is possible to drop the free-choice requirement. The free-choice requirement itself is not essential. It is important that the class of nets on which the extension in the inheritance rules is based satisfies the various results about home markings derived in Sections 5.4 and 7.2 for live and bounded free-choice nets. The general class of live and bounded P/T nets is not suitable for this purpose, as can be shown by means of the P/T net in Figure 7.26.

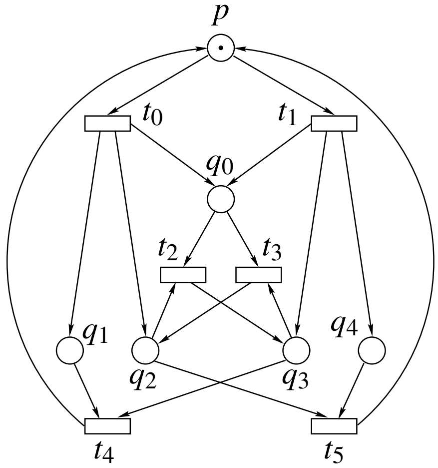



Figure 7.26: Non-monotonicity of home markings for live and bounded P/T nets.

The figure shows a P/T net $N$, which is not free-choice, with a place $p$ containing a single token. It is not difficult to verify that $(N, [p])$ is live and bounded and that $[p]$ is a home marking of $(N, [p])$. However, consider the marked P/T net $(N, [p^2])$. Firing sequence $t_0 t_0 t_2 t_3$ leads to marking $[q_1{}^2, q_2{}^2]$. In this marking, no transitions are enabled. Clearly, this implies that $[p^2]$ is not a home marking of $(N, [p^2])$. Thus, the class of live and bounded P/T nets does not satisfy the property that home markings are monotone. Note that $(N, [p^2])$ is also no longer live and bounded. Since the reachable marking $[q_1{}^2, q_2{}^2]$ is a deadlock, $(N, [p^2])$ is obviously not live. Furthermore, firing the sequence of transitions $t_0 t_1 t_4 t_5$, starting from the initial marking, yields marking $[p^2, q_0{}^2]$, which means that the number of tokens in place $q_0$ can increase indefinitely. Thus, $(N, [p^2])$ is also not bounded.

A consequence of the observation that the general class of live and bounded P/T nets does not satisfy monotonicity of home markings is that it is not possible to omit the free-choice requirement in rules *PP*, *PJ*, *PT*, and *PJ*3. It remains an interesting question to find classes of P/T nets that could replace the class of live and bounded free-choice P/T nets in the inheritance rules. The class of live and bounded well-handled P/T nets might be a candidate. Well-handled P/T nets and some of their properties are studied in [1, 3].

An option to generalize the four inheritance rules of this section is to base the extension in the rules on the class of live and bounded P/T nets, dropping the free-choice requirement, while at the same time adding other restrictions to the rules to guarantee their correctness. Note that any such restrictions must exclude the P/T net of Figure 7.26 as an allowed extension. Requiring that the result of the transformation yields again a life cycle might be sufficient. However, proving such a generalization requires a lot of work. It is not clear whether it is worth the effort. The verification whether a particular transformation satisfies the requirements of such a generalized rule is very inefficient. As a result, it might be very hard to apply such a rule in practice. A second reason for not putting too much effort in generalizing the current set of inheritance rules is that these rules have not yet been thoroughly tested in practice.

It may be clear from the above discussion that, in our opinion, the free-choice requirement in the inheritance rules of this section is a good compromise between generality and efficiency. However, there appears to be another reasonable compromise, based on the notion of so-called *safe* P/T nets. A marked (labeled) P/T net is said to be safe if and only if all places of the P/T net contain at most one token in any marking reachable from the initial marking. Note that safeness is a requirement that implies boundedness. In a setting where object life cycles are required to be safe, the free-choice requirement in the four inheritance rules of this section can be replaced by the requirement that the P/T net upon which the modification of the original life cycle is based is safe, given the specific initial marking identified in each of the rules.

As an example, let us consider inheritance rule *PP* of Theorem 7.3 and illustrated in Figure 7.2. It has been explained in Section 7.2 that the key to the correctness of this inheritance rule is that any tokens consumed from the special place $p$ by a transition of $N$ can, under all circumstances, be returned to place $p$. In particular, the lemmas in that section show that adding tokens to or removing tokens from place $p$ does not disturb this process. The safeness requirement concerning object life cycles implies that place $p$ can always contain at most one token. If this token is consumed from $p$ by a transition of $N$, the liveness requirement regarding $N$ is sufficient to guarantee that this token can be returned, whereas the safeness requirement of both the original life cycle and the extension $N$ guarantees that, in the process, no tokens can be added to place $p$. Thus, if object life cycles are required to be safe, the restriction that $(N, [p])$ is live and safe is sufficient to guarantee that a token consumed by $N$ from $p$ can always be returned. For the other inheritance rules of this section, similar arguments hold.

In safe P/T nets, places model *conditions*, which can be either true or false. In practice, safeness appears to be a reasonable assumption for object life cycles. All the examples used so far, as well as all the object life cycles occurring in the case study of the next section are safe. The complexity of deciding both safeness of general P/T nets and liveness of safe P/T nets is PSPACE-complete [32]. As a result, the complexity of verifying the requirements of any of the four inheritance rules adapted to safe P/T nets is (worst-case) PSPACE-complete. This means that, theoretically, the requirements of rules *PP* and *PJ* are easier to verify than the requirements of their safe variants, whereas the requirements of *PT* and *PJ*3 are more complex to verify than the requirements of their safe counterparts.

As already mentioned in Section 6.1, the class of so-called sound workflow nets of [1, 2] is very similar to the class of object life cycles introduced in Definition 6.2. Workflow nets are used to model *business* processes. Business processes tend to change quite often, which causes many problems for maintaining these processes. In [4], the inheritance framework developed in this section adapted to a setting of safe P/T nets as discussed above is applied to tackle several of the problems related to the frequent changes occurring in business processes. The results of [4] support our belief that safe object life cycles and the corresponding variants of the inheritance rules yield an interesting alternative to the framework developed in this section.

Comparing the inheritance rules in this section to the axioms of inheritance in Section 4.2, it is noteworthy that no inheritance rules have been given corresponding to the axioms of life-cycle inheritance defined in Property 4.30. The reason is that such inheritance rules are simply combinations of the four rules of this section. All kinds of combinations of inheritance rules are allowed, as long as the original life cycle and its extensions satisfy certain restrictions with respect to their transition labels. Basically, it must be guaranteed that methods which function as guards in an extension based on inheritance rule *PT* do not appear in extensions based on any of the other inheritance rules. This restriction follows from the original definition of the four inheritance relations. The guards in inheritance rule *PT* are encapsulated, whereas the new methods in any of the extensions in the other rules are hidden. As explained, in the definition of life-cycle inheritance, it is not allowed to treat different calls of the same methods in a different way. Examples of combinations of inheritance rules are the following. Clearly, based on inheritance rule *PJ*, it is allowed to replace two arcs in a P/T net simultaneously as long as in both cases the requirements of rule *PJ* are met. In this case, the abovementioned restriction is trivially satisfied. It is also allowed to replace an arc while at the same time adding an iteration. Another example is the simultaneous extension of a life cycle with an alternative and a parallel branch of behavior, which is allowed when the guards of the alternative do not appear in the extra parallel branch.

Finally, as already mentioned in Section 6.1, some of the rules developed in this section can also be used to construct object life cycles in general, without the aim to construct subclasses of existing life cycles. In particular, rules *PP* and *PJ* are suited for this purpose. The requirements of these rules can be efficiently verified and the result is again a life cycle. However, also rules *PT* and *PJ*3 might be useful in some cases. Since transition labeling does not play a role in the definition of object life cycles, the rules can even be generalized when the aim is solely to construct object life cycles. The requirements concerning the labels of transitions added to the original object life cycle may be dropped. An interesting consequence of the observation that the four inheritance rules preserve life-cycle properties is that, based on Theorem 6.7 (Characterization of object life cycles), they can be easily transformed into liveness-and-boundedness-preserving transformation rules on P/T nets. Recall that also [2] and [72] describe a set of liveness-and-boundedness-preserving transformation rules that, similar to the rules of this subsection, correspond to design constructs such as sequential composition, parallel composition, choice, and iteration. However, the rules presented in this section are far more general than the rules of [2] and [72]. The fact that the inheritance rules of this section can be transformed into liveness-and-boundedness-preserving transformation rules implies that they might have many more applications than only the one that is presented in this paper.

# 8 Case Study

To validate the theory developed so far, the framework of the previous two sections is applied to a small case study, namely the development of a groupware editor. The Petri-net-framework is chosen instead of the process-algebraic-framework of Section 4, because the former is closer to practical applicability than the latter. The aim of this case study is not to give full specifications of all the classes in the design of a groupware editor, but to focus on the object life cycles. This means that data types, class attributes, and method implementations are omitted. If necessary, an informal explanation is given.

**Informal system requirements** The informal requirements for the groupware editor are as follows. The editor is meant to edit some kind of diagrams. Multiple users, possibly situated at different workstations, may be editing a single diagram in a joint editing session. Users may choose to either view or edit a diagram. They may join or leave a session at will. It must be clear to all users who is currently editing some given diagram. The diagrams under consideration may be complex, possibly consisting of multiple components. Components may introduce hierarchy in a diagram. It is possible to open or close a component revealing or hiding its details. Not just any user may view or edit any component in a diagram. Users must have permission to do so. Permissions are not fixed; to get permission, a user can simply select a component and then try the desired command. If there are no conflicting permissions, the command succeeds; otherwise, it fails and the user is notified. Users may explicitly surrender permissions. Most permissions are automatically reset if the user leaves the editing session; a few permissions may persist between sessions.

In the next subsections, the concept of life-cycle inheritance is used to design a groupware editor satisfying the abovementioned informal requirements. The development is split into three steps. First, a groupware *viewer* is designed. The design is fairly simple and the resulting system allows multiple users to view existing diagrams. Second, the viewer is transformed into a multi-user editor by adding editing functions to the life cycles of classes in the viewer design. Third, the multi-user editor is specialized to a groupware editor by adding permissions. The case study shows how life-cycle inheritance can be used to structure an object-oriented design process. It also shows how object life cycles can be reused in a design.

## 8.1 Multi-user viewer

Four classes can be distinguished in the design of the viewer: `user`, `diagram`, `user_ session`, and `edit_session`. The first two classes have straightforward interpretations. The third and fourth class are intended to structure viewing sessions. During such a session, an `edit_session` object keeps track of all users viewing a particular diagram. An object of class `user_session` maintains all data involved in a particular session of a particular user. Figure 8.1 shows the life cycles for the objects of the above four classes. Some of the places in life cycles `user` and `user_session` are marked with identifiers for the purpose of future reference.

Users can be created (*cru*) or deleted (*delu*). This means that they are added to or removed from the list of users of the viewer. Once a user exists, he or she may join and leave edit sessions. A user that participates in at least one edit session may issue *view* commands in order to view diagrams. At this point, the user has no other commands available.

Class `diagram` has a very straightforward life cycle. Diagrams can be created (*crd*), viewed (*view*), and deleted (*deld*).

A user interacts with a diagram through a `user_session` object. Each time the user joins an edit session, a new object of class `user_session` is created. As before, the *view* command is the only possible command a user can execute. If the user leaves the edit session, the `user_session` is terminated.

Objects of class `edit_session` keep track of all users involved in a single edit session. The first user that "joins" an edit session for some diagram, actually creates a new `edit_session` object. If users join an already existing session, no new object is created; the information is simply stored in the existing object. To fulfill the requirement that all users must know who is participating in the session, it is assumed that the implementation of *join* is such that the new user gets a list of users already present. Furthermore, the information about a new user is broadcast (*brdcst*) to all other users participating in the session. When the user leaves, this information is broadcast to all remaining users. The last user leaving the session terminates the `edit_session` object.

In a complete object-oriented specification, it must be clear which methods interact with each other. In this case study, the interaction between methods is not formally specified. Instead, the following assumptions are made. First, `user` methods invoke methods with the same name in `user_session`, which, in turn, invoke methods of the same name in `edit_session` and `diagram`. Second, methods which do not have

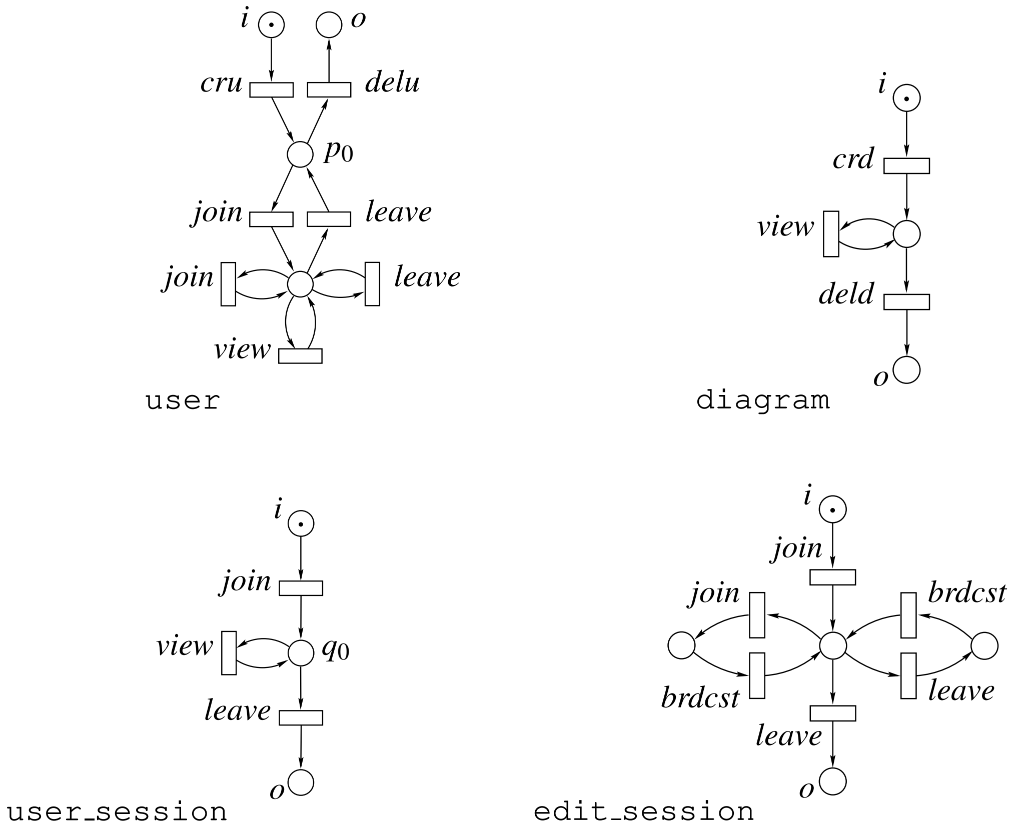



Figure 8.1: A multi-user viewer.

counterparts in one of the other classes are assumed to interact with (objects in) the environment. The examples of these methods in Figure 8.1 are *cru*, *delu*, *crd*, *deld*, and *brdcst*.

## 8.2 Multi-user editor

In the second step of the design process, the multi-user viewer of the previous subsection is transformed into a multi-user editor. Permissions are not yet incorporated. They are added in the third and final step of the design. Consequently, in the version of the editor described in the current subsection, it is possible that a component is deleted by one user, while it is being changed or viewed by another one.

Three classes in the design of the viewer, shown in Figure 8.1, are extended with editing facilities, namely `user`, `diagrams`, and `user_session`. Class `edit_session` does not need to change. Thus, the design of the editor consists of four classes, namely `user_e`, `diagram_e`, `user_session_e`, and `edit_session`. The new classes are shown in Figure 8.2. In the remainder of this subsection, it is shown that the three new classes are subclasses of the corresponding classes of the viewer of Figure 8.1 under life-cycle inheritance.

First, consider class `diagram_e`. In the editor, it is possible to modify diagrams by means of method *mod*. It easily follows from inheritance rule *PP* of Theorem 7.3 that `diagram_e` is a subclass of class `diagram` under protocol/projection inheritance. Thus, it also is a subclass of `diagram` under life-cycle inheritance.

Second, class `user` of Figure 8.1 has been extended to class `user_e` in the design of the editor. In the editor, users of the system are responsible for creating and deleting diagrams (methods *crd* and *deld*).

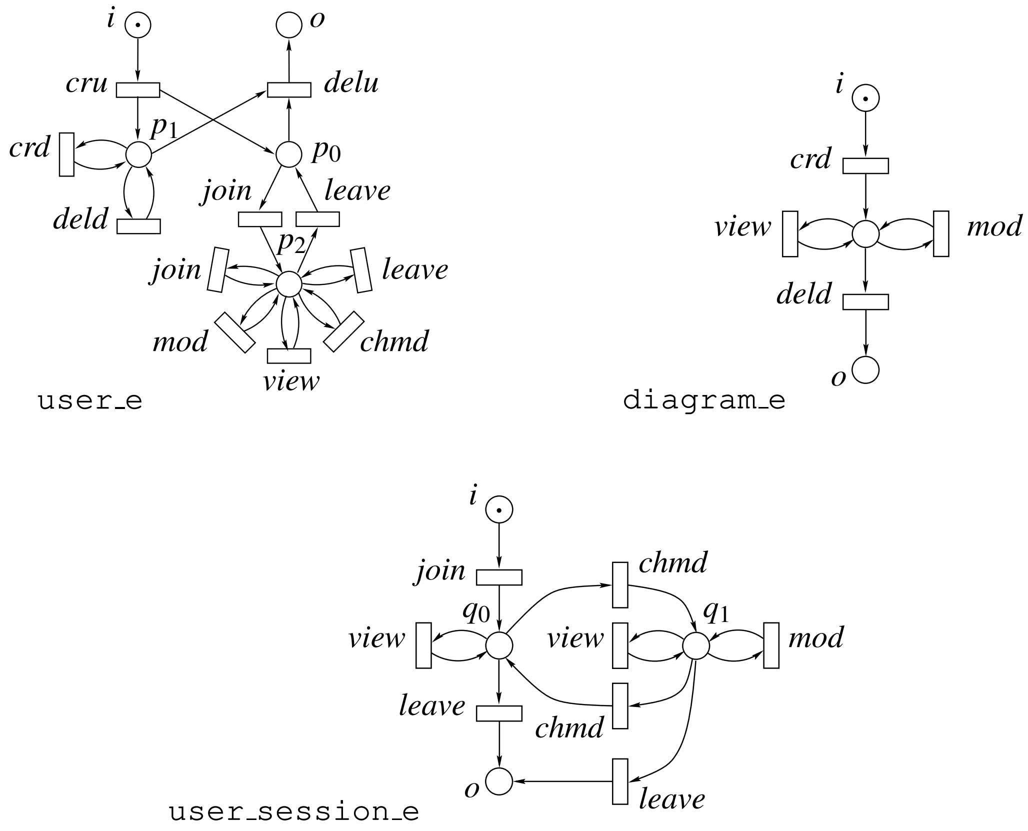



Figure 8.2: A multi-user editor.

Creation and deletion of diagrams can be done independently of editing (other) diagrams. Thus, a branch of parallel behavior, consisting of place $p_1$ and the two aforementioned methods, is added to the life cycle of class `user`. Inheritance rule *PJ*3 of Theorem 7.23 can be used to prove that this extension yields a subclass under projection inheritance. The two transitions labeled *cru* and *delu* serve as the input and output transition of the additional parallel branch, respectively. Due to the presence of place $p_0$ in life cycle `user`, any extra place between *cru* and *delu* is clearly implicit in this life cycle. For the same reason, the addition of such an extra place preserves the life cycle properties.

Another extra feature of the editor compared to the viewer of the previous subsection is that users are allowed to modify diagrams. For this purpose, class `user_e` contains method *mod*. Users have to choose whether they want to modify a diagram or whether they are satisfied with just the option to view it. Method *chmd* (change mode) can be used to toggle between viewing and editing mode. The implementation of methods *mod* and *chmd* can be such that only a subset of users is allowed to enter the editing mode. It simply follows from applying inheritance rule *PP* that the addition of methods *mod* and *chmd* to class `user` leads to a subclass.

It is not difficult to verify that the subsequent application of inheritance rules *PJ*3 and *PP* yields that `user_e` is a subclass of `user` under life-cycle inheritance.

Third, it must be shown that `user_session_e` is a subclass of `user_session`. For each edit session a user joins, a `user_session_e` object is created. This object keeps track of the mode in which the user is for this particular diagram. Initially, the user is in viewing mode. By invoking method *chmd*, the user can change to editing mode. This means that the object life cycle of class `user_session` in Figure 8.1 is extended with a choice. Inheritance rule *PT* of Theorem 7.17 can be applied to show that user_session_e is a subclass of user_session. Method *chmd* acts as the guard. Method *leave* in user_session guarantees that any extra transition between places $q_0$ and $o$ in this life cycle preserves the life cycle properties.

Summarizing, three of the four inheritance rules of the previous section have been used to extend the viewer of Figure 8.1 to an editor. The most important conclusion of this subsection is that the specifications of the object life cycles of the viewer have been *reused* during the design process.

## 8.3 Groupware editor

In the last step of the design of the groupware editor, a mechanism is added to handle permissions. Permissions are needed to prevent all kinds of anomalies. For example, they guarantee that it is impossible that one user deletes a component of a diagram when, at the same time, another user is viewing this component. Four new classes are designed, namely user_p, diagram_p, user_session_p, and edit_session_p. Figure 8.3 shows two of these four classes.

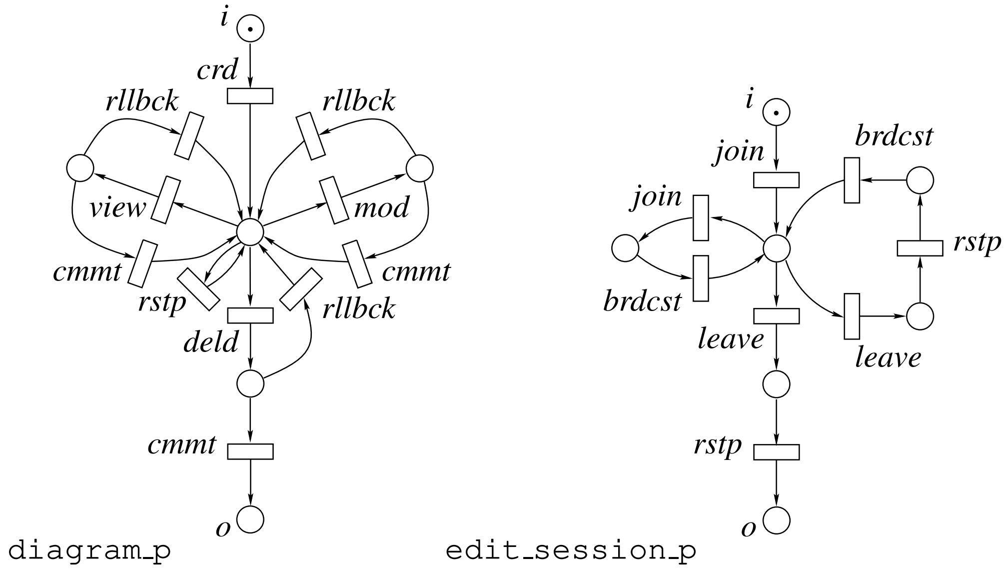



Figure 8.3: A groupware editor.

The current set of permissions for editing a diagram is maintained in the corresponding diagram_p object. This is the only feasible option since some permissions may persist between edit sessions, as stated in the system requirements explained in the introduction to this section. Also explained in the requirements is that a user can get permission for some editing command by simply selecting a component and executing the command. Therefore, after every editing action *view*, *mod*, or *deld*, diagram_p executes either a rollback (*rllbck*) or a commit (*cmmt*), depending on whether or not the user has permission for the specific editing action. Another change in diagram_p when compared to class diagram_e is that the method *rstp* has been added which can be used to reset permissions. It must be taken care of that the three other classes invoke *rstp* whenever appropriate.

Classes user_p and user_session_p are simple extensions of the multi-user-editor classes user_e and user_session_e of Figure 8.2. Therefore, these two classes are not shown in Figure 8.3. Since users may surrender permissions at any time, class user_p is obtained from user_e by adding a loop consisting of a single transition labeled *rstp* to place $p_2$. Since users interact with diagrams through user sessions, the life cycle of user_session_p is constructed from user_session_e by adding the same loop as above to places $q_0$ and $q_1$.

The fourth class in the design of the groupware editor is class `edit_session_p` which is constructed from class `edit_session` of Figure 8.1. As mentioned in the system requirements, permissions are reset when a user leaves a session. This means that `edit_session` is extended with calls of method *rstp* after each invocation of the *leave* method.

It remains to be shown that the new classes are subclasses of the corresponding classes in the earlier designs. The addition of *rstp* to classes `user_e` and `user_session_e` is captured by inheritance rule *PP* of Theorem 7.3. Hence, `user_p` and `user_session_p` are subclasses of `user_e` and `user_session_e`, respectively.

The addition of *rstp* to `diagram_e` is also captured by rule *PP*. To show that the addition of method *cmmt* to `diagram_e` is correct, a generalization of inheritance rule *PJ* as discussed in Section 7.6 is needed. Applying rule *PJ* three times simultaneously shows that the extension with method *cmmt* yields a subclass. To show that the addition of method *rllbck* is allowed, consider the intermediate result obtained after the previous two additions. The desired result follows easily from inheritance rule *PT* of Theorem 7.17, again applied three times simultaneously. Hence, class `diagram_p` is a subclass of `diagram_e`.

Finally, to show that `edit_session_p` is a subclass of `edit_session`, it suffices to apply rule *PJ* twice simultaneously.

Summarizing the results of this subsection, it has been shown that the four classes in the design of the groupware editor are subclasses under life-cycle inheritance of the corresponding classes in the multi-user editor discussed in the previous subsection. The classes of the multi-user editor, in turn, are subclasses of the corresponding classes of the viewer presented in Section 8.1. Since life-cycle inheritance is transitive, it follows that the four classes of the groupware editor are also subclasses of the corresponding viewer classes. This means that the groupware editor preserves the behavior of the viewer. Although the case study is not very complex, it shows that the concept of life-cycle inheritance can be used to structure the design process. In addition, the inheritance rules of Section 7 stimulate the reuse of object life cycles. Observe that all four inheritance rules are needed in the case study. More realistic case studies are needed to test whether the four rules are sufficient in practical design situations. For this purpose, it is also necessary to incorporate the results of this paper in a complete object-oriented formalism.

# 9 Conclusions

**Concluding remarks** The concept of inheritance is one of the key concepts in object-oriented design. However, in most object-oriented methods which are in common use, the most important being UML [21, 58], inheritance is only well defined for the set of methods of a class and its attributes. It is implicitly assumed that the behavior of an object of a subclass extends the behavior of an object of its superclass. To overcome this omission in the definition of inheritance, this paper studies inheritance of dynamic behavior.

Section 4 studies inheritance of behavior in a simple process-algebraic setting. Process algebra is particularly well suited for this purpose, because it does not have an explicit representation of process states. In addition, the notions of encapsulation and abstraction have been studied extensively in a process-algebraic setting. Encapsulation corresponds to *blocking* actions, whereas abstraction corresponds to *hiding* actions. Intuitively, blocking and hiding method calls play an important role in inheritance of behavior. Section 4.1 introduces four inheritance relations, based on either encapsulation or abstraction or both. Each of the inheritance relations captures different kinds of extensions to an object life cycle. The axioms of inheritance of Section 4.2 illustrate the most characteristic extensions allowed under each form of inheritance. Protocol inheritance captures the addition of an alternative to an object life cycle. Projection inheritance allows the addition of parallel and sequential behavior. Under protocol/projection inheritance, it is allowed to postpone behavior. In a setting which allows the specification of iterative behavior, protocol/projection inheritance captures the extension of an object life cycle with an iteration. Life-cycle inheritance, which is the most general form of inheritance discussed in this paper, allows all kinds of combinations of the abovementioned extensions.

In Section 6, the four inheritance relations of Section 4 are studied in the framework of Petri nets. Due to the graphical nature of Petri nets and their explicit representation of states, the Petri-net framework is closer to object-oriented methods used in practice than the algebraic framework. Inspired by the algebraic axioms of inheritance, Section 7 presents four inheritance rules, which can be used to transform a class into a subclass. In Section 8, these inheritance rules are used in the development of a groupware editor. The concept of life-cycle inheritance and the accompanying inheritance rules prove to be useful in structuring the design process. In addition, they stimulate the reuse of object life cycles. The most important conclusion is that the concepts developed in this paper can be a promising addition to existing object-oriented methods.

The development of the concept of inheritance of behavior in this paper is a perfect illustration of the complementary nature of process algebra and Petri nets. Process algebra proved to be a very useful framework for developing a clear understanding of the important concepts in the formalization of inheritance of behavior. The Petri-net framework provides a formalization that is much closer to existing object-oriented methods. It would have been difficult to achieve the two goals of a clear conceptual understanding and a practical framework in a single theory.

**Future work** The most important future challenge is to integrate our concepts of inheritance in an existing object-oriented design method. If UML is chosen for this purpose, one could choose to translate the inheritance notions to statechart diagrams or one could choose to replace statechart diagrams by Petri nets. It is also possible to integrate the inheritance concepts in an object-oriented formalism based on Petri nets, such as OPN [47]. An advantage of this approach is that it results in an integrated framework with a sound theoretical basis. A disadvantage is that object-oriented languages based on Petri nets are not yet in common use. An advantage of incorporating the results of this paper in a framework as UML is that it will be easier to get acceptance of the notion of inheritance of behavior in practice, particularly when it is translated to statechart diagrams.

An aspect that plays an important role in the literature on inheritance is the so-called substitutability principle of [73]. The substitutability principle says that an object of some subclass can always be used in a context where an object of its superclass is expected. When incorporating the inheritance concept developed in this paper in a complete object-oriented design method, it must be investigated to what extent, or under which assumptions, the four inheritance relations adhere to the substitutability principle. To answer this question, it is not sufficient to consider life-cycle specifications in isolation. It is also necessary to take into account the static structure of classes as well as the interaction between objects.

Another topic for further study is the set of inheritance rules to construct subclasses from object life cycles. It is interesting to study variants of the current inheritance rules, trying to improve their generality or their efficiency. It would also be useful to translate the inheritance rules to the statechart diagrams of UML. Furthermore, it is interesting to transform the rules into transformation rules that preserve liveness and boundedness of P/T nets. One application of such transformation rules is that they can be used to construct object life cycles.

Future case studies with life-cycle inheritance might reveal that there is a need for a small extension of the framework. When verifying a subclass relationship between two object life cycles in the current framework, it is not allowed to treat different calls of the same method in a different way. All calls of the same method are either blocked, or hidden, or left untouched. However, it is not difficult to define variants of the four inheritance relations of this paper that allow a different treatment of different calls of the same method. It simply requires the use of temporary method names to distinguish the different groups of method calls. For example, a variant of protocol inheritance could be defined as follows. If an object life cycle is a subclass under protocol inheritance of another object life cycle, then any renaming of the methods new in the subclass yields a subclass of the life cycle under the variant of protocol inheritance. By choosing an appropriate renaming, it is possible that a method already present in the superclass acts as a guard. Thus, by using temporary names for methods acting as guards in the subclass, it is possible to distinguish new invocations of such a method in the subclass from calls of the same method already present in the object life cycle of the superclass.

Another interesting topic is the application of the inheritance concepts of this paper in the area of component-based software engineering [70, 71]. A common problem in existing object-oriented designs is that the dynamic interaction between objects is often implicit in the design, which frequently causes problems for the maintainability and extendibility of the design. Component-based software engineering is characterized by a strong focus on component interfaces and component interaction. The inheritance concepts of this paper can be seen as a step from object-oriented design towards component-based design. Some early results of the application of the inheritance notions of this paper in the area of component-based software engineering can be found in [6].

Another promising application area for our concepts of inheritance of behavior is workflow management. Petri nets are well suited to define and analyze workflow processes (see, for example, [2, 3]). Object life cycles and P/T nets modeling workflow processes are essentially the same. Our inheritance concepts are particularly useful in the area of adaptive and ad hoc workflow management. This area of workflow management focuses on handling process changes. An important topic in this context is the adaptation of workflow processes in such a way that the new workflow process preserves the behavior of the original process. A foundational study of the application of inheritance of behavior to tackling problems related to process changes in workflow management can be found in [4].

Finally, an issue that transcends all the abovementioned topics for future work is the collection of experimental results for the four inheritance relations of this paper (as well as for any future variants of these relations). Ultimately, only practical results can validate our approach to inheritance of behavior. Such practical experience might also shed more light on the question of which type of inheritance is useful for what kinds of applications and application areas.

**Related work** The literature on object-oriented design and its theoretical foundations contains several studies related to the research described in this paper. In [75], abstraction in a process-algebraic setting is suggested as an inheritance relation for behavior. However, only a few examples are studied. Based on these examples, the author concludes that abstraction is useful but not always sufficient to capture desirable subclass relationships. Life-cycle inheritance as introduced in this paper, which combines abstraction with encapsulation, is sufficiently powerful to prove the desired subclass relationships in the examples of [75].

Other research on inheritance of behavior or related concepts such as behavioral subtyping describes both fundamental studies, such as [7, 22, 25, 29, 42, 28, 49, 50, 51, 56, 57, 62, 63, 67], and a few practical studies, such as [48, 52]. The fundamental studies all take a specific formalism for specifying the dynamic behavior of a class as a starting point; the practical studies start from concrete examples of desired subclass relationships. Besides the aforementioned references, books [43] and [44] can be useful for readers interested in the topic of inheritance of behavior.

The given references describe a wide variety of inheritance and subtype relations. In many cases, the relations show similarities to either protocol inheritance or projection inheritance, although the corresponding concepts of blocking and hiding method calls are never mentioned. In particular, several of the fundamental studies mentioned above are based on the idea – inspired by the notion of substitutability – that a subclass must be able to accept at least all the sequences of method calls that its superclass is willing to accept. This notion is closely related to our concept of protocol inheritance. In general, our work distinguishes itself from the given references by the strong focus on the ordering of method calls and the constructive approach by means of the axioms of inheritance and the inheritance rules. In addition, our notion of life-cycle inheritance is more general than most of the inheritance and subtype relations appearing in the abovementioned work.

The variety in inheritance relations reported in the literature is not very surprising if one considers, for example, the large number of semantics that exist for concurrent systems (see, for example, [34]). There simply is no single semantics of concurrent systems that is suitable for all purposes. Similarly, it cannot be expected that there is a single inheritance relation for behavior that is always useful. However, it is promising that similar concepts appear to play a role in many of the inheritance and subtype relations for behavior currently described in the literature.

The research presented in this paper is also related to work in the area of concurrency theory. The relation is mostly a technical one. Recall that our four inheritance relations are preorders on object life cycles, which are a specific kind of processes. In the literature on concurrency theory, many preorders on various kinds of process models appear for many different purposes (see, for example, [34] for an overview). Often, such preorders represent so-called implementation relations or refinement relations. Similar to our inheritance relations, these implementation and refinement relations are used to structure the design process. However, conceptually, the inheritance mechanism, the specification/implementation mechanism, and stepwise refinement are complementary ways for dealing with the ever-increasing complexity of design processes. Of course, this does not exclude the possibility that the ideas presented in this paper can be used to support the specification/implementation and stepwise-refinement approaches to design. An example of a preorder that is related to one of our inheritance relations comes from the area of conformance testing [23]. Conformance testing is the problem of verifying by means of testing whether some system implementation satisfies a given specification. In [23], a testing relation called "extends" is described, which is related to protocol inheritance.

**Bibliographical remarks** This paper supersedes one technical report, namely [15], and two other papers, namely [5] and [14]. Report [15] forms the basis for the material presented in Section 4 of the current paper. In [5], an early version is presented of the material in Sections 6 and 7 of this paper, whereas [14] describes the case study of Section 8. The merit of the current paper is that it combines the conceptual study of inheritance of behavior in the algebraic framework, the translation of the algebraic concepts into the more practical Petri-net framework, and the application of the concepts in the case study. In addition, most of the material has evolved over time. In particular, the material in Sections 6 and 7 has been extended and improved when compared to the presentation in [5]. The formulation of the inheritance rules in Section 7 has improved in such a way that they are a good compromise between generality and efficiency and their correctness is proven in detail. In [5], each of the rules still contains a requirement that the result of the transformation yields an object life cycle, which is a requirement that can be hard to verify; in addition, some of the proofs in [5] are omitted, whereas other proofs are much simpler due to the extra requirements in the formulation of the rules. As a final remark, this paper is a revised version of [13, Chapter 4] and [16].

**Contribution** The main contribution of this paper is the complete formalization of the concept of inheritance of behavior starting from the intuitive notions of hiding and blocking method calls. The approach focuses on the ordering of method calls and is constructive. In two different formalisms, four inheritance relations are defined and inheritance rules are presented that can be used to construct subclasses from some given class. To validate the results, the theory is applied to a small case study. The fact that the concepts presented in this paper transcend the two formalisms of process algebra and Petri nets supports the conclusion that the notions of hiding and blocking method calls play an important role in studying and formalizing inheritance of behavior.

**Acknowledgments** Many people have contributed to this paper in some way. We would like to thank Jos Baeten, Eike Best, Gregor Engels, Reiko Heckel, Kees van Hee, Isabelle Reymen, Bram Stappers, Eric Verbeek, and Marc Voorhoeve.

## References

1. W.M.P. van der Aalst. Structural Characterizations of Sound Workflow Nets. Computing Science Report 96/23, Eindhoven University of Technology, Department of Mathematics and Computing Science, Eindhoven, The Netherlands, December 1996.
2. W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997, 18th. International Conference, ICATPN'97, Proceedings*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426, Toulouse, France, June 1997. Springer, Berlin, Germany, 1997.
3. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.
4. W.M.P. van der Aalst and T. Basten. Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theoretical Computer Science*. To appear.
5. W.M.P. van der Aalst and T. Basten. Life-Cycle Inheritance: A Petri-Net-Based Approach. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997, 18th. International Conference, ICATPN'97, Proceedings*, volume 1248 of *Lecture Notes in Computer Science*, pages 62–81, Toulouse, France, June 1997. Springer, Berlin, Germany, 1997.
6. W.M.P. van der Aalst, K.M. van Hee, and R.A. van der Toorn. Component-Based Software Architectures: A Framework Based on Inheritance of Behavior. *Science of Computer Programming*. To appear. Also appeared as BETA Working Paper Series WP 45, Eindhoven University of Technology, Department of Technology Management, Eindhoven, The Netherlands, 2000.
7. P. America. Designing an Object-Oriented Language with Behavioural Subtyping. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Foundations of Object-Oriented Languages, REX School/Workshop, Proceedings*, volume 489 of *Lecture Notes in Computer Science*, pages 60–90, Noordwijkerhout, The Netherlands, May/June 1990. Springer, Berlin, Germany, 1990.
8. J.C.M. Baeten and T. Basten. Partial-Order Process Algebra (and its Relation to Petri Nets). In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*. Elsevier Science, Amsterdam, The Netherlands. To appear.
9. J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Conditional Axioms and $\alpha/\beta$-calculus in Process Algebra. In M. Wirsing, editor, *Formal Description of Programming Concepts - III, Proceedings of the IFIP TC2/WG2.2 Working Conference*, pages 53–75, Ebberup, Denmark, August 1986. North–Holland, Amsterdam, The Netherlands, 1987.
10. 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.
11. 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.
12. T. Basten. Branching Bisimilarity is an Equivalence indeed! *Information Processing Letters*, 58(3):141–147, May 1996.
13. T. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. PhD thesis, Eindhoven University of Technology, Department of Mathematics and Computing Science, Eindhoven, The Netherlands, December 1998.
14. T. Basten and W.M.P. van der Aalst. Inheritance of Dynamic Behavior: Development of a Groupware Editor. In G.A. Agha, F. De Cindio, and G. Rozenberg, editors, *Concurrent Object-Oriented Programming and Petri Nets, Lecture Notes in Computer Science, Advances in Petri Nets*. Springer, Berlin, Germany. To appear.
15. T. Basten and W.M.P. van der Aalst. A Process-Algebraic Approach to Life-Cycle Inheritance: Inheritance = Encapsulation + Abstraction. Computing Science Report 96/05, Eindhoven University of Technology, Department of Mathematics and Computing Science, Eindhoven, The Netherlands, March 1996.

16. T. Basten and W.M.P. van der Aalst. Inheritance of Behavior. Computing Science Report 99/17, Eindhoven University of Technology, Department of Mathematics and Computing Science, Eindhoven, The Netherlands, November 1999.

17. J.A. Bergstra and J.W. Klop. Process Algebra for Synchronous Communication. *Information and Control*, 60(1–3):109–137, 1984.

18. G. Berthelot. Transformations and Decompositions of Nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course*, volume 254 of *Lecture Notes in Computer Science*, pages 359–376, Bad Honnef, Germany, September 1986. Springer, Berlin, Germany, 1987.

19. E. Best and C. Fernández C. *Nonsequential Processes: A Petri Net View*, volume 13 of *EATCS monographs on Theoretical Computer Science*. Springer, Berlin, Germany, 1988.

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

21. G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, Reading, Massachusetts, USA, 1998.

22. H. Bowman, C. Briscoe-Smith, J. Derrick, and B. Strulo. On Behavioural Subtyping in LOTOS. In H. Bowman and J. Derrick, editors, *Formal Methods for Open Object-based Distributed Systems, 2nd. IFIP International Conference, Proceedings*, pages 335–351, Canterbury, UK, July 1997. Chapman & Hall, London, UK, 1997.

23. E. Brinksma, G. Scollo, and C. Steenbergen. Lotos Specifications, their Implementations, and their Tests. In *Protocol Specification, Testing and Verification, VI, Proceedings of the IFIP WG 6.1 6th. international workshop*, pages 349–360, Montreal, Quebec, Canada, June 1986. Elsevier, North-Holland, Amsterdam, The Netherlands, 1987.

24. J.M. Colom and M. Silva. Improving the Linearly Based Characterization of P/T Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 113–145. Springer, Berlin, Germany, 1990.

25. E. Cusack. Refinement, Conformance and Inheritance. *Formal Aspects of Computing*, 3(2):129–141, 1991.

26. J. Desel. Reduction and Design of Well-behaved Concurrent Systems. In J.C.M. Baeten and J.W. Klop, editors, *CONCUR '90, Theories of Concurrency: Unification and Extension, Proceedings*, volume 458 of *Lecture Notes in Computer Science*, pages 166–181, Amsterdam, The Netherlands, August 1990. Springer, Berlin, Germany, 1990.

27. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.

28. K.K. Dhara and G.T. Leavens. Forcing Behavioral Subtyping through Specification Inheritance. In *1996 IEEE 18th. International Conference on Software Engineering, ICSE-18, Proceedings*, pages 258–267, Berlin, Germany, March 1996. IEEE Computer Society Press, Los Alamitos, California, USA 1996. A revised version is available as Technical Report TR #95-20c, Iowa State University, Department of Computer Science, March 1997.

29. J. Ebert and G. Engels. Structural and Behavioural Views on OMT-Classes. In E. Bertino and S. Urban, editors, *Object-Oriented Methodologies and Systems, International Symposium, ISOOMS'94, Proceedings*, volume 858 of *Lecture Notes in Computer Science*, pages 142–157, Palermo, Italy, September 1994. Springer, Berlin, Germany, 1994.

30. J. Esparza. Synthesis Rules for Petri Nets, and How They Lead to New Results. In J.C.M. Baeten and J.W. Klop, editors, *CONCUR '90, Theories of Concurrency: Unification and Extension, Proceedings*, volume 458 of *Lecture Notes in Computer Science*, pages 182–198, Amsterdam, The Netherlands, August 1990. Springer, Berlin, Germany, 1990.

31. J. Esparza. Decibility and Complexity of Petri Net Problems - An Introduction. In Reisig and Rozenberg [65], pages 374–428.

32. J. Esparza and M. Nielsen. Decibility Issues for Petri Nets - A Survey. *Journal of Information Processing and Cybernetics*, 30(3):143–160, 1994.

33. W.J. Fokkink. *Introduction to Process Algebra*. Texts in Theoretical Computer Science: An EATCS Series. Springer, Berlin, Germany, 2000.

34. R.J. van Glabbeek. The Linear Time – Branching Time Spectrum II: The Semantics of Sequential Systems with Silent Moves (extended abstract). In E. Best, editor, *CONCUR '93, 4th. International Conference on Concurrency Theory, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81, Hildesheim, Germany, August 1993. Springer, Berlin, Germany, 1993.

35. 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 Francisco, California, USA, August/September 1989. Elsevier Science, North-Holland, Amsterdam, The Netherlands, 1989. Full version appeared as [36].

36. R.J. van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996.

37. J.F. Groote and F.W. Vaandrager. An Efficient Algorithm for Branching and Stuttering Equivalence. In M.S. Paterson, editor, *Automata, Languages and Programming, 17th. International Colloquium, Proceedings*, volume 443 of *Lecture Notes in Computer Science*, pages 626–638, Warwick University, England, July 1990. Springer, Berlin, Germany, 1990.

38. D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.

39. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice–Hall International, London, UK, 1985.

40. I. Jacobson, M. Ericsson, and A. Jacobson. *The Object Advantage: Business Process Reengineering with Object Technology*. Addison-Wesley, Reading, Massachusetts, USA, 1991.

41. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use,* volume 1*, Basic Concepts*. EATCS monographs on Theoretical Computer Science. Springer, Berlin, Germany, 1992.

42. G. Kappel and M. Schrefl. Inheritance of Object Behavior - Consistent Extension of Object Life Cycles. In J. Eder and L.A. Kalinichenko, editors, *East/West Database Workshop, Proceedings of the 2nd. International Workshop*, Workshops in Computing, pages 289–300, Klagenfurt, Austria, September 1994. Springer, Berlin, Germany, 1995.

43. H. Kilov and W. Harvey, editors. *Object-Oriented Behavioral Specifications*, volume 371 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, Massachusetts, USA, 1996.

44. H. Kilov, B. Rumpe, and I. Simmonds, editors. *Behavioral Specifications of Businesses and Systems*, volume 523 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, Massachusetts, USA, 1999.

45. A.V. Kovalyov. On Complete Reducibility of Some Classes of Petri Nets. In *Application and Theory of Petri Nets 1990, 11th. International Conference, ICATPN'90, Proceedings*, pages 352–366, Paris, France, June 1990.

46. A.V. Kovalyov. An $O(|S| \times |T|)$-Algorithm to Verify if a Net is Regular. In J. Billington and W. Reisig, editors, *Application and Theory of Petri Nets 1996, 17th. International Conference, ICATPN'96, Proceedings*, volume 1091 of *Lecture Notes in Computer Science*, pages 366–379, Osaka, Japan, June 1996. Springer, Berlin, Germany, 1996.

47. C.A. Lakos. From Coloured Petri Nets to Object Petri Nets. In G. De Michelis and M. Diaz, editors, *Application and Theory of Petri Nets 1995, 16th. International Conference, Proceedings*, volume 935 of *Lecture Notes in Computer Science*, pages 278–297, Torino, Italy, June 1995. Springer, Berlin, Germany, 1995.

48. C.A. Lakos. Pragmatic Inheritance Issues for Object Petri Nets. In *TOOLS Pacific 1995, Proceedings*, pages 309–321, Melbourne, Australia, 1995. Prentice-Hall, 1995.

49. G.T. Leavens and D. Pigozzi. A Complete Algebraic Characterization of Behavioral Subtyping. *Acta Informatica*, 36(8):617–663, 2000.

50. B.H. Liskov and J.M. Wing. A Behavioral Notion of Subtyping. *ACM Transactions on Programming Languages and Systems*, 16(6):1811–1841, November 1994.

51. I. Maung. On Simulation, Subtyping and Substitutability in Sequential Object Systems. *Formal Aspects of Computing*, 7(6):620–651, 1995.

52. J.D. McGregor and D.M. Dyer. A Note on Inheritance and State Machines. *Software Engineering Notes*, 18(4):1–15, October 1993.

53. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, Berlin, Germany, 1980.

54. R. Milner. *Communication and Concurrency*. Prentice–Hall International, London, UK, 1989.

55. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

56. E. Najm and A. Nimour. A Calculus of Object Bindings. In H. Bowman and J. Derrick, editors, *Formal Methods for Open Object-based Distributed Systems, 2nd. IFIP International Conference, Proceedings*, Canterbury, UK, July 1997. Chapman & Hall, London, UK, 1997.

57. O. Nierstrasz. Regular Types for Active Objects. In O. Nierstrasz and D. Tsichritzis, editors, *Object-Oriented Software Composition*, pages 99–121. Prentice Hall, 1995.

58. Object Management Group. OMG Unified Modeling Language. OMG, http://www.omg.com/uml/.

59. J.L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1981.

60. C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Mathematik, Bonn, Germany, 1962. In German.

61. L. Pomello, G. Rozenberg, and C. Simone. A Survey of Equivalence Notions of Net Based Systems. In G. Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 410–472. Springer, Berlin, Germany, 1992.

62. F. Puntigam. Types for Active Objects Based on Trace Semantics. In E. Najm and J.-B. Stefani, editors, *Formal Methods for Open Object-based Distributed Systems, 1st. IFIP International Workshop, Proceedings*, pages 4–19, Paris, France, March 1996. Chapman & Hall, London, UK, 1996.

63. F. Puntigam. Non-regular Process Types. In P. Amestoy, P. Berger, M. Daydé, I. Duff, V. Frayssé, L. Giraud, and D. Ruiz, editors, *Euro-Par'99 Parallel Processing, 5th. International Euro-Par Conference, Proceedings*, volume 1685 of *Lecture Notes in Computer Science*, Toulouse, France, August/September 1999. Springer, Berlin, Germany, 1999.

64. W. Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS monographs on Theoretical Computer Science*. Springer, Berlin, Germany, 1985.

65. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science. Advances in Petri Nets*. Springer, Berlin, Germany, 1998.

66. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets II: Applications*, volume 1492 of *Lecture Notes in Computer Science. Advances in Petri Nets*. Springer, Berlin, Germany, 1998.

67. S. Rudkin. Inheritance in LOTOS. In K.R. Parker and G.A. Rose, editors, *Formal Description Techniques, IV, Proceedings of the IFIP TC6/WG6.1 Fourth International Conference, FORTE '91*, pages 409–424, Sydney, Australia, November 1991. North–Holland, Amsterdam, The Netherlands, 1992.

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

69. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, Reading, Massachusetts, USA, 1998.

70. M. Shaw, G. David, and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1996.

71. C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, Reading, Massachusetts, USA, 1998.

72. R. Valette. Analysis of Petri Nets by Stepwise Refinements. *Journal of Computer and System Sciences*, 18(1):35–46, February 1979.

73. P. Wegner and S.B. Zdonik. Inheritance as an Incremental Modification Mechanism or What like is and isn't like. In S. Gjessing and K. Nygaard, editors, *ECOOP '88, European Conference on Object-Oriented Programming, Proceedings*, volume 322 of *Lecture Notes in Computer Science*, pages 55–77, Oslo, Norway, August 1988. Springer, Berlin, Germany, 1988.

74. W.P. Weijland. *Synchrony and Asynchrony in Process Algebra*. PhD thesis, University of Amsterdam, Department of Mathematics and Computer Science, Amsterdam, The Netherlands, 1989.

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