2004 Society for Design and Process Science
Printed in the United States of America

SAFETY PROPERTIES IN PETRI NET MODULES

Julia Padberg
Institute for Software Technology and Theoretical Computer Science, Technical University Berlin, Germany

This paper introduces safety properties in the temporal logic sense (as defined in (Manna and Pnueli, 1995)) to Petri net modules. Petri net modules (Padberg, 2001; Padberg, 2002) have been achieved by a transfer of algebraic specification modules to Petri nets. They consist of three nets; the interface nets import and export, and the body of the module. The import net states the prerequisites the modules assumes. The body net represents the internal functionality. The export net gives an abstraction of the body that can be used by the environment. The interfaces IMP and EXP are related to the body BOD via morphisms. These modules conform with the basic concepts of components and component-based systems of Continuous Software Engineering (CSE) (Weber, 1999).
We make precise what it means that a Petri net module has specific safety properties. We differentiate between explicit and implicit properties. Explicit safety properties are stated additionally to the export net. Implicit are those properties that hold in the export net without being stated explicitly.The main advantage of our approach are module operations to compose larger modules from basic ones. We can show that the composition of modules preserves safety properties:
Given two modules with implicit or explicit safety properties then the composition of these modules is again a module with implicit or explicit safety properties.

Keywords: *Module, Petri nets, Safety property*

1. Introduction

Changing environments - commercial, technical or social - demand software systems that can be adapted to those changes with reasonable effort. Component-based systems have been proposed as an adequate support for that task. Today there is no doubt on the importance of component-based systems. There are various specification for the description of component-based software architectures. These approaches allow modeling components and composing them into architectures. However, they provide limited scope for formal analysis and reasoning. But the complexity of most software systems requires formal techniques for analysis and reasoning on local component properties as well as on global system properties.
The main motivation for Petri net modules is the modeling of component-based systems. Software components are an useful and widely accepted abstraction mechanism. Components are deployed during the entire software life cycle, from analysis to maintenance. The component concept as suggested in (Müller and Weber, 1998; Weber, 1999) for *Continuous Software Engineering (CSE)* is the basic concept for our approach.

Petri Net Modules and Component-Based Systems Petri net modules seem to be an adequate specification technique for describing component-based architectures as we have illustrated in our case study (Padberg and Buder, 2001). The 1-to-1 correspondence of Petri net modules to component concepts in the sense of (Müller and Weber, 1998; Weber, 1999) allows the specification of the components' behavior and its abstraction in the interfaces. As the underlying paradigms are essentially the same Petri net modules can directly be used to model the operational view (in the sense of operational behavior) of a component. The interfaces of Petri net modules are Petri nets, and not only nodes of a net. Hence the export allows presenting an abstraction of the modules behavior. And the import allows requiring a specific behavior of the modules to be imported. The import specifies what must be satisfied by the export interface of an imported module. But it does not specify specific modules to be imported. Hence every module is formally completely unrelated to other modules. So it can be easily exchanged by another module as long as the export specifications are compatible. The actual relations between the modules are established using the module operations (Padberg, 2001; Padberg, 2002).

Properties of Modules Verification of components and component-based systems has the same difficulties as any large system with infinite states. Temporal logic is an approach to verify infinite systems and/or to avoid the state-space explosion. Instead of checking the state space deductive proofs are used to verify the desired property. Various behavior properties have been characterized in terms of temporal logic.
We introduce Petri net modules with safety properties as a first steps towards reasoning on local component properties to infer global system properties. In this context the notion of safety is of central importance. Intuitively, a safety property in the sense of (Manna and Pnueli, 1995) means that "nothing bad" can happen in the system, and is expressed by a temporal logic formula. In this paper we use the invariant formula $\Box\phi$ to express simple safety properties. We present modules that state safety properties implicitly or explicitly in the export interface. The first main result is then that these modules guarantee those safety properties for their body as well. The main issue is then how the properties of a module relate to the properties of the whole system. We have to ensure that these properties are not violated during the construction of the system. That construction is achieved using module operations. So the module operations have to preserve the safety properties. The second main result states that the composition of modules does preserve safety properties.

Related Work Structuring concepts for Petri nets can be quite coarsely distinguished into those employing hierarchical concepts and those employing connector mechanisms. Using hierarchical concepts for structuring Petri nets the system is described by an abstract net while special functionalities are encapsulated into subnets and called like procedures by merging transitions or places and exchange of tokens. Examples of this approach can be found in (Jensen, 1992; Buchholz, 1994; He, 1996; Fehling, 1993). Connector mechanisms between modules of Petri nets describe the communication, coordination or cooperation via signals by a relation between input and output. The exchange of tokens or only read-operation of modules is described by special features like read arcs, inhibitor arcs and others (e.g. in (Christinsen and Hansen, 1994; Sibertin-Blanc, 1994; Desel *et al.*, 2000; Deiters and Gruhn, 1994)). In other approaches places and transitions of modules are merged by well-defined operations (e.g. (Kindler, 1995; Battiston *et al.*, 1991*b*; Battiston *et al.*, 1991*a*; Broy and Streicher, 1992)).
Using temporal logic for verifying properties of Petri nets e. g. (Chandy and Misra, 1988; Brauer *et al.*, 1991; Bradfield and Stirling, 1990; Bradfield, 1992; Chandy and Misra, 1988; Stirling, 1992; Reisig, 1998) has a long and fruitful tradition Compositional reasoning (or compositional verification) has been pursued in various ways. But in contrast to our approach those components are given only as a net with specific nodes as interfaces for connecting them. In (Kindler, 2002) this basic component approach is chosen to employ DAWN – (Distributed Algorithms Working Notation) a technique for verifying network algorithms – to verify component-based systems.
Other behavioral specifications of components have been used besides Petri Nets. E.g., State Machine based approaches (such as (Nierstrasz, 1995; Alur *et al.*, 1999)) and process calculi oriented notations. For example DARWIN (Magee *et al.*, 1995) has an operational semantics using the π-calculus and WRIGHT

(Allen *et al.*, 1998) uses CSP. These employ the behavioral specifications for the semantics and not so much as a specification technique in its on rights.
An behavioral extension (constrained based) of CORBA IDLs is given in (Krämer, 1998).

Outline of this Paper The remainder of this paper is organized as follows. In Section 2. we give short review of the notions of Petri net module and composition. Although these notions are given formally the paper is understandable without deeper knowledge of the formal background. We provide the basic definitions and give extensive explanations. Supplementary definitions and lemmata and proofs can be found in the Appendix. In Section 3. we introduce temporal logic formulae over places over a place/transition net. These formulae represent safety properties of the modeled system. We discuss what it means that a module has properties and then distinguish between implicit and explicit representation of safety properties. The first main result concerns Petri net modules, that present safety property via the export and guarantee that these properties hold in the body net of the module as well. The second main result states that these safety properties presented via the export are preserved under the composition of modules. Hence, we have a compositional approach to reasoning over modules. In Section 4. we conclude with a short summary, an evaluation of this work, and the discussion of future work.

2. Review of Petri Net Modules

We now give a short introduction to our approach and explain the notion of Petri net modules and the module operations.
First we give a short intuition of the underlying basics. Here we use the algebraic notion of Petri nets as introduced in (Meseguer and Montanari, 1990). Hence a Petri net is given by the set of transitions and the set of places and the pre- and post domain function. The pre- (and post-) domain function maps each transition into the free commutative monoid over the set of places, representing the places and the arc weight of the arcs in the pre-domain (respectively in the post-domain). The free commutative monoid over P can be considered as the set of finite multisets over P. So an element $w \in P^{\oplus}$ can be presented as a linear sum $w = \sum_{p \in P} \lambda_p \cdot p$ and we can extend the usual operations and relations as $\oplus$, $\ominus$, $\leq$, and so on. Moreover, we need to state how often is a basic element with in an element of the free commutative monoid given. We define this for an element $p \in P$ and a word $w \in P^{\oplus}$ with $w_{|p} = \lambda p \in P^{\oplus}$. Subnets $N' \in \mathcal{P}(N)$ of a given net N with $N' \subseteq N$ can be easily defined by subsets of places and transitions, where the pre- and post-domain of transitions may be extended. $\mathcal{P}(N)$ denotes the set of all subnets.
All these notions are stated precisely in the Appendix.

Definition 1 (Place/Transition Nets) *A place/transition net is given by* $N = (P, T, pre, post, M^0)$ *with* P *the set of places,* T *the set of transitions,* $pre, post : T \to P^{\oplus}$ *the pre- and post-domain of transitions.* $M^0 \in P^{\oplus}$ *describes the initial marking.* ◀

Based on the algebraic notion of Petri nets (Meseguer and Montanari, 1990) we use simple homomorphisms that are generated over the set of places. These morphisms map places to places and transitions to transitions. Morphisms are the basic entity in category theory; they can present the internal structure of objects and relate the objects. So they are the basis for the structural properties a category may have and can be used successfully to define various structuring techniques.
Morphisms are essential for the notion of modules and the definition of module operations. Two Petri net morphisms $m : IMP \to BOD$ and $r : EXP \rightsquigarrow BOD$ connect the interfaces to the body.
The import morphism m is a *plain morphism* and describes how and where the resources in the import interface are used in the body. This morphisms maps each place and transition in the import interface to its counterpart in the body. The initial marking of the source net needs to be place-wise smaller than the initial marking of the target net. Plain morphisms are presented by an arrow $\to$.

Definition 2 (Plain Morphisms) *A plain morphism $f : N_1 \to N_2$ is given by $f = (f_P, f_T)$ with $f_P : P_1 \to P_2$ and $f_T : T_1 \to T_2$ so that $pre_2 \circ f_T = f_P^{\oplus} \circ pre_1$ and post analogously.*
Moreover, for the initial marking we have for all $p \in P_1$: $f_P^{\oplus}(M^0_{1|p} \leq M^0_{2|f_P(p)}$ ◀

The export morphism r is a *substitution morphism* and describes how the functionality provided by the export interface is realized in the body. This is done by mapping each part of the export interface that represents a certain functionality to the the part of the body by which the functionality is realized. Substitution morphisms map places to places as well. But they can map a single transition to a whole subnet. Hence they *substitute* a transition by a net. These morphisms are more complicated and do not yield nice categorical properties. But they capture a very broad idea of refinement and hence are adequate for the relation between the export net and the body net. The initial marking has to satisfy the same condition as for plain morphisms. Subsequently substitution morphisms are presented by an undulate arrow $\rightsquigarrow$.

Definition 3 (Substitution Morphism) *A substitution morphism $f : N_1 \rightsquigarrow N_2$ is given by $f = (f_P, \mathfrak{f}_T)$ with $f_P : P_1 \to P_2$ and $\mathfrak{f}_T : T_1 \to \mathcal{P}(N_2)$ with $\mathfrak{f}_T(t) := N_2^t \subseteq N_2$ such that $N_2^t = (P_2^t, T_2^t, pre_2^t, post_2^t)$*

- *$f_P({}^{\bullet}t) \in P_2^t$ and*
- *$f_P(t^{\bullet}) \in P_2^t$*

Moreover, for the initial marking we have for all $p \in P_1$: $f_P^{\oplus}(M^0_{1|p}) \leq M^0_{2|f_P(p)}$

The composition of substitution morphisms $f : N_1 \rightsquigarrow N_2$ and $g : N_2 \rightsquigarrow N_3$ is given by:
$g \circ f := (g_P \circ f_P, \mathfrak{g}_T \circ \mathfrak{f}_T)$ where $g_T \circ f_T : T_1 \to \mathcal{P}(N_3)$ is defined by $g_T \circ \mathfrak{f}_T(t) := \bigcup_{x \in T_2^t} N_3^x$.
Since all $N_3^x \subseteq N_3$ this construction is well defined. ◀

Next we review our notion of Petri net modules. We use this name as this notion of module can easily be transferred to any variant of Petri nets. in this paper we introduce modules of place/transition nets in order to keep the notion simple.

Definition 4 (Petri Net Module) *A Petri net module $MOD = (IMP, EXP, BOD)$ is given by three place/transition nets that are related by morphisms; the plain morphism $m : IMP \to BOD$, and the substitution morphism $r : EXP \rightsquigarrow BOD$ as depicted adjacently.*

$$\begin{tikzcd}
& EXP \arrow[d, rightsquigarrow, "r"'] \\
IMP \arrow[r, "m"] & BOD
\end{tikzcd}$$

Fig. 1. Petri Net Module

◀

Example 5 (Simple Module MOD) *The focus of this paper is the formal foundation for compositional reasoning in Petri net modules. Hence the examples are used for clarifying and illustrating the presented notions and results. They are designed to be small and clear. So we have cut out any (more or less artificial) application to be modeled. The example module in Figure 2 does not present any meaningful application. Figure 2 illustrates a very simple net module. The import describes a single transition and the plain morphism $IMP \to BOD$ is an inclusion. The export of the module presents cyclic runs. The morphism $EXP \rightsquigarrow BOD$ abstracts from the more complex behavior of the body. The places a and b in EXP are mapped to a and b in BOD. Transition z is mapped to the subnet including the places a, b, d and the transitions w, x in BOD. Transition y is mapped to transition u. Hence, the export is an abstraction of the body. The complex structure of the cycle is hidden. But this module also shows a problem: Looking at the export it seems to be obvious that a token can be on place a or (exclusively) on place b.*

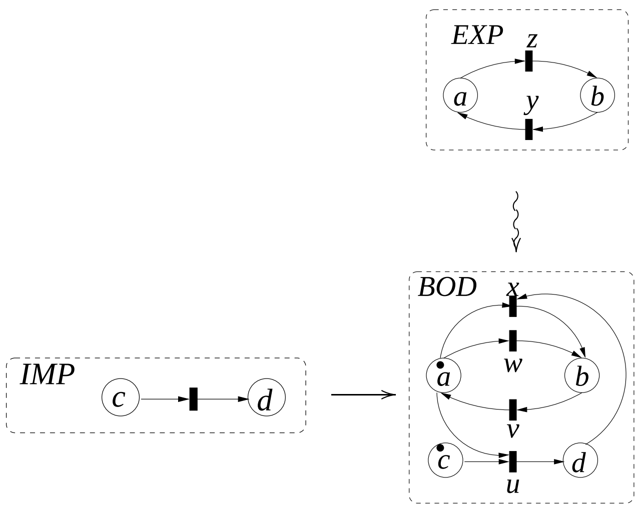

Fig. 2. Example of a Simple Module

Examining the body net BOD it is obvious that after firing of transition u the tokens are neither on place a nor on place b. Hence, the export may be a misleading abstraction. To prevent this we introduce subsequently modules with safety properties. Such modules guarantee that the safety properties presented via the export interface are satified as well in the body of the module. ◀

The composition of modules is one of the module operations defined in (Padberg, 2001; Padberg, 2002). From the practical point of view it is the most important one. The composition describes the import of a module into another module. The diagram for the construction of the compositiom is given in Figure 3. This module operation requires two modules $MOD_1 = (IMP_1, EXP_1, BOD_1)$, and $MOD_2 = (IMP_2, EXP_2, BOD_2)$. Moreover a morphism maps the import interface IMP_1 of the importing module MOD_1 to the export interface EXP_2 of the imported module MOD_2. This morphism $h : IMP_1 \to EXP_2$ matches the functionality requested by the importing module with the functionality provided by the imported module. The resulting module $MOD_3 := MOD_2 \odot_h MOD_1$ has the import interface of the imported module IMP_2, the export interface of the importing module EXP_1 and a constructed body BOD_3. This is the body of the importing module where the functionality specified in the import interface is substituted with the "implementation" provided in the body of the imported module.
The result of this composition is then $MOD_3 = (IMP_2, EXP_1, BOD_3)$, where the body BOD_3 is constructed by gluing together the body of the importing module BOD_1 and the body of th imported module BOD_2.

$$\begin{tikzcd}
 & & EXP_1 \arrow[d, squiggly, "r_1"'] \\
 & IMP_1 \arrow[r, "m_1"] \arrow[d, squiggly, "h"'] & BOD_1 \arrow[dd, squiggly, "r_2'"] \\
 & EXP_2 \arrow[d, squiggly, "r_2"] \arrow[r, phantom, "(1)"] & \\
IMP_2 \arrow[r, "m_2"] & BOD_2 \arrow[r, "m_1'"] & BOD_3
\end{tikzcd}$$

Fig. 3. Composition Diagram

$$\begin{tikzcd}
 & EXP_1 \arrow[d, squiggly] \\
IMP_2 \arrow[r] & BOD_3
\end{tikzcd}$$

Fig. 4. Resulting Module

So, the composed net in BOD_3 has at least as many nodes (either places or transitions) as the minimum of BOD_1 and BOD_2 and at most as many as BOD_1 and BOD_2 together. The morphisms specify which nodes of BOD_1 and BOD_2 are identified, namely those the have the same source node in IMP_1.

In categorical terms this construction is a pushout. In Figure 3 the square **(1)** presents such pushout.

3. Safety Properties in Petri Net Modules

Next, we formalize safety properties in order to formulate our theorems concerning their preservation. We recall formulas over markings and their translations via morphisms. An axiomatic expression is λp, denoting that $\lambda \in \mathbb{N}$ tokens are on place p. We then can build logic formulae over such axioms, e.g. $4a \implies 2b$ formalizes the statement that 4 tokens on place a imply 2 tokens on place b. Safety properties describe invariants of the net behavior. Hence we use the henceforth operator $\Box$ to express that a formula shall hold for all reachable markings. The formula $\Box a \vee b$ in the export in Figure 5 states that one token is always either on place a on place b.

Definition 6 (Formulas, Translations) *λp is a static formula for $\lambda \in \mathbb{N}$ and $p \in P$, static formulas are built up using the logical operators $\wedge$ and $\neg$. Let φ be a static formula over N. Then $\Box\varphi$ is an invariant formula.*
The validity of formulas is given w. r. t. the marking of a net. Let $M \in P^{\oplus}$ be an arbitrary marking of N then the formula λp holds in N under M written as $M \models_N \lambda p$ iff $\lambda p \leq M$. For $M \models_N \neg\varphi_1$ iff $\neg(M \models_N \varphi_1)$ and $M \models_N \varphi_1 \wedge \varphi_2$ iff $(M \models_N \varphi_1) \wedge (M \models_N \varphi_2)$. The invariant formula $\Box\varphi$ holds in N under M iff φ holds in all states reachable from M: $M \models_N \Box\varphi$ iff $\forall M' \in [M\rangle : M' \models_N \varphi$. We also write $N \models \Box\varphi$ instead of $M^0 \models_N \Box\varphi$.
The translation of formulas $\mathcal{T}_{\{}$ over N_1 along a morphism $f = (f_P, f_T) : N_1 \to N_2$ to formulas over N_2 is given for atoms by $\mathcal{T}_f(\lambda p) = \lambda f_P(p)$. The translation of formulas is given recursively by $\mathcal{T}_f(\neg\varphi) = \neg\mathcal{T}_f(\varphi)$, and $\mathcal{T}_f(\varphi_1 \wedge \varphi_2) = \mathcal{T}_f(\varphi_1) \wedge \mathcal{T}_f(\varphi_2)$, and $\mathcal{T}_f(\Box\varphi) = \Box\mathcal{T}_f(\varphi)$
◀

We now have to ensure specific conditions that guarantee morphisms preserving safety properties. Intuitively the next definition requires for substitution morphism to be place preserving: Any transition of the target net that has a place of the source net in its pre- or post-domain needs to have a source transition in the source net, so that the pre-domain and post-domain of the transition is preserved. In other words this definition ensures that neither arcs may be deleted nor “new” arcs to ”old” places are allowed. Hence we have chosen the term place preserving.

Definition 7 (Place Preserving Substitution Morphism) *A substitution morphism $f : N_1 \to N_2$ is place preserving if for all $t_2 \in T_2$ with $pre_2(t_2)_{|f_P} \oplus post_2(t_2)_{|f_P} \neq \epsilon$ we have: There is some $t_1 \in T_1$ s.t. $t_2 \in T_2^{t_1}$, and $f_P^{\oplus}(pre_1(t_1)) = pre_2(t_2)_{|f_P}$, and $f_P^{\oplus}(post_1(t_1)) = post_2(t_2)_{|f_P}$.*
◀

Clearly, to preserve safety properties the marking on the mapped places needs to stay the same. Places that are not in the image of the morphism may be marked arbitrarily.

Definition 8 (Marking Strict Substitution Morphism) *A substitution morphism $f : N_1 \to N_2$ is called marking strict if f_P is injective, and $f_P^{\oplus}(M_1^0) = M^0_{2|f_P}$.* ◀

We now can state that a substitution morphism that is place preserving and marking strict preserves safety properties up to the renaming $\mathcal{T}_f$ induced by the morphism. This is the first main result we present in this paper. This is the basis to define modules that present safety properties via the export.

Theorem 9 (Safety Property Preserving Morphism) *A substitution morphism $f : N_1 \to N_2$ that is place preserving and marking strict then the following holds:*

$$N_1 \models \Box\varphi \text{ implies } N_2 \models \Box\mathcal{T}_f(\varphi)$$

Hence, such a morphism is called safety property preserving (spp) morphism. ◀

The proof is analogous to the proof of Fact 4.16 in (Gajewsky *et al.*, 1999). Although the underlying morphism is different we can use the same argument, since the conditions we use in Definition 7 correspond to Fact 4.13 in (Gajewsky *et al.*, 1999).
This allows defining Petri net modules that preserve safety properties from the export net to the body net. As we desire a treatment of properties that is independent of the body net, we can use safety property preserving morphisms to relate the export net to the body net. If we require $r : EXP \to BOD$ to be safety property preserving, then any safety property holding in EXP will be preserved.

Definition 10 (Modules with Implicit Safety Properties) *A module $MOD = (IMP, EXP, BOD)$ where $r : EXP \to BOD$ is safety property preserving is a module with implicit safety properties.* ◀

Using the result of Theorem 9 we can now conclude that those safety properties $\Box\varphi$ holding in the export net EXP must also hold in BOD. The safety properties $\Box\mathcal{T}_r(\varphi)$ in BOD are translated along the morphism $r : EXP \to BOD$.

Corollary 11 (Implicit Safety Properties) *Given a module $MOD = (IMP, EXP, BOD)$ with implicit safety properties then for any safety property holding in the export net $EXP \models \Box\varphi$, the translated safety property holds in the body $BOD \models \Box\mathcal{T}_r(\varphi)$.* ◀

Nevertheless we can distinguish specific safety properties in the export in order to have an explicit representation of safety properties. We extend the export net with a set of safety properties Φ over the places of net EXP.

Definition 12 (Modules with Explicit Safety Properties) *A module $MOD = (IMP, (EXP, \Phi), BOD)$ where Φ is a set of safety formulas holding in EXP and $r : EXP \to BOD$ is a modules with explicit safety properties.* ◀

Corollary 13 (Explicit Safety Properties) *Given a module $MOD = (IMP, (EXP, \Phi), BOD)$ with explicit safety properties then for any safety property $\Box\varphi \in \Phi$, the translated safety property holds in the body $BOD \models \Box\mathcal{T}_r(\varphi)$.* ◀

We now have to ask what happens to safety properties when we compose modules. The main intention of this work is to simplify compositional reasoning by preserving safety properties throughout the construction of modules. First we give an example of a composition of modules with safety properties. Thereafter we present our second main result. Theorem 15 states that the composition of modules preserves safety properties as well.

Example 14 (Composition of Modules) *Again we give here an example that merely illustrates the notions and results of this paper but does not present any meaningful application. The example of a composition is illustrated in Figure 5.*
There is the module $MOD1 = (IMP1, EXP1, BOD1)$ with explicit safety properties. The safety property $\Box a \vee b$ is stated explicitly in the export of module $MOD1$. It states the fact that the token is either on place a or on place b. This can be seen immediately. In fact, the property could be formulated even stronger using an "exclusive or".

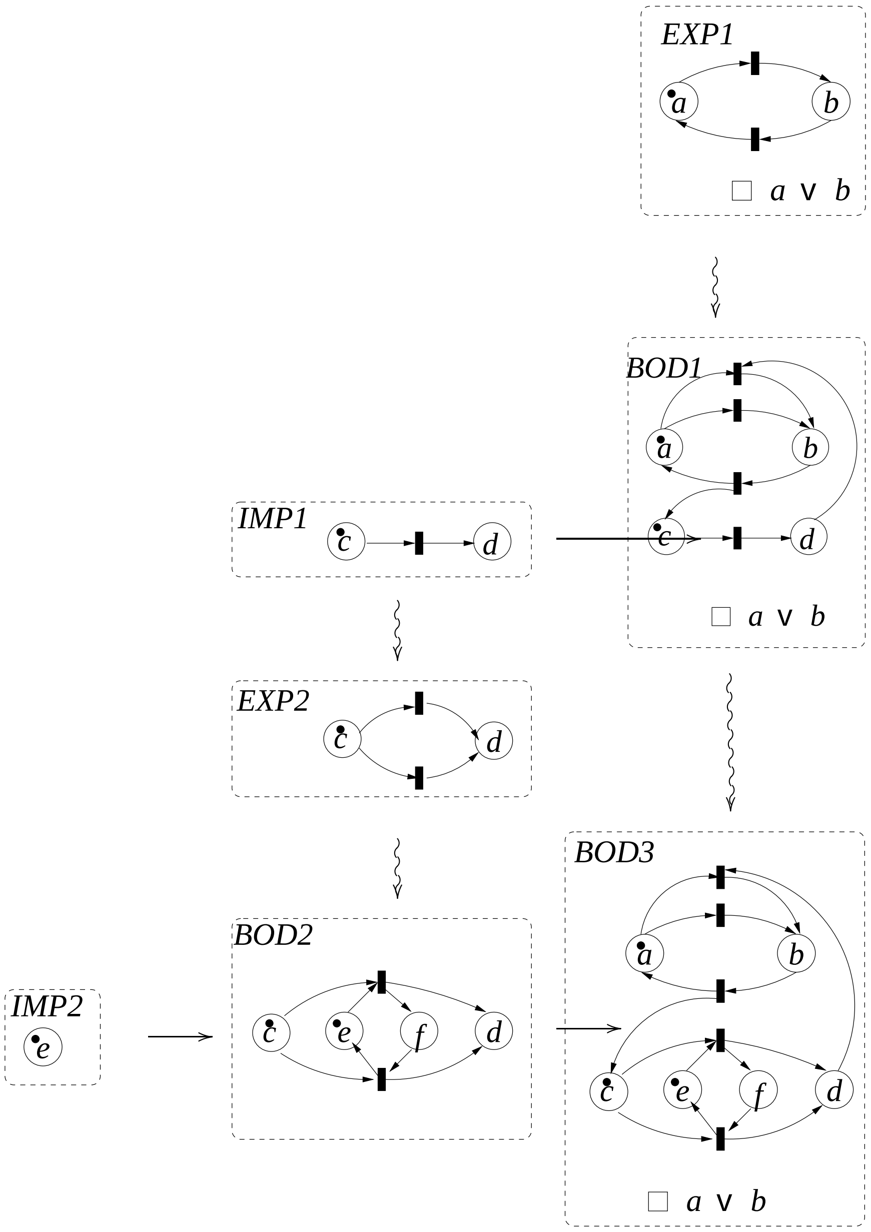

Fig. 5. Example for Composition

The morphism $EXP1 \rightsquigarrow BOD1$ is safety property preserving as it is injective on the places, preserves the marking and no "new" transitions are adjacent to "old" places. The module $MOD2 = (IMP2, EXP2, BOD2)$ is a module with implicit safety properties.
The resulting module $MOD3 = MOD2 \odot MOD1 = (IMP2, (EXP1, \{\Box a \vee b\}), BOD3)$ is constructed subsequently.
The new body $BOD3$ of the resulting modules is the net $BOD1$, where the transition between the places c and d is replaced by the net between the places c and d in the net $BOD2$. So, $BOD3$ can be considered as the gluing of $BOD1$ and $BOD2$ along the net $IMP1$. This exactly what the pushout construction of the composition diagram in Figure 3 describes formally. The resulting module $MOD3 = MOD2 \odot MOD1 = (IMP2, (EXP1, \{\Box a \vee b\}), BOD3)$ is a module with explicit safety properties because the morphism $EXP1 \rightsquigarrow BOD_3$ is safety property preserving.

◀

Theorem 15 (Safety Property Preserving Composition) *Given two modules MOD_1 and MOD_2 both with safety properties, let the composition morphism $h : IMP_1 \rightsquigarrow EXP_2$ be safety property preserving, then the resulting module $MOD_2 \odot_h MOD_1$ is a module with safety properties as well.*
If the module $MOD_1 = (IMP_1, (EXP_1, \Phi), BOD_1)$ is a module with explicit safety properties, then the resulting module $MOD_2 \odot_h MOD_1 = (IMP_2, (EXP_1, \Phi), BOD_3)$ is a module with explicit safety properties as well. ◀

Proof:
The construction of the composition is based mainly on the pushout (see **(1)** in Figure 3). The undulated arrows are safety property preserving substitution morphisms. Since pushouts are stable under safety property preserving morphisms (see Lemma 23 in the Appendix) we can conclude that $EXP_2 \rightsquigarrow BOD_3$ is safety property preserving. Hence, $MOD_2 \odot_h MOD_1$ is a module with safety properties.
The export of the resulting module is the export of the importing module MOD_1, so $MOD_2 \odot_h MOD_1 = (IMP_2, (EXP_1, \Phi), BOD_3)$ is a module with explicit safety properties. $\surd$

4. Conclusion

The work we have presented aims in the long run at a formal basis in terms of Petri nets for an architecture description language (ADL). Up to now we have notions for Petri net modules, module operations and evolution. In our view ADLs should involve a possibility of separation of concerns. Some ADLs provide process calculi oriented notations for their semantic foundation. But the operational view is not given explicitly. ADLs should allow modeling components from specific viewpoints. Especially the operational view is important, e.g. for specifying protocols as one of the main communication means between components. Most ADLs use merely one specification technique. An exception are those attempts to extend UML towards an ADL. This results in the neglect of the operational view of components.
In this paper we have presented a step towards an ADL with Petri nets as the formal foundation. We have combined notions of temporal logic, namely safety properties to Petri net modules. So the starting point for compositional reasoning is given. We have introduced Petri net modules with safety properties. These modules ensure that safety properties that are represented by the export are satified in the body of the modules well.

Evaluation of Results The development of the case study has clearly shown that the new concept of Petri net modules (Padberg, 2001; Padberg, 2002) is applicable for structuring large and complex net models. The results presented here are a starting point for the compositional reasoning over Petri net modules. Safety properties are one major criterion for ensuring correctness of a model. Hence, this is valuable extension.
Nevertheless this approach is yet far from being an ADL to be used in practice. In the long run one of the main steps towards a practical application is the extension to high-level nets as the modeling power of place/transition nets is limited. There are various high-level net variant as available. From the theoretical point of view algebraic high-level nets have a lot of nice properties and some of the required notions have been already developed. On the other hand there are high-level net variants with very good tool support. Those tools with an open structure and nice semantic interfaces allow the coupling with other tools and could be the basis for the possible implementation in CASE tools. Especially PEP (Best, 1997) has those nice features and moreover allows the validation of temporal logic formulas. Pushout as a special case of colimits can be efficiently computed by a tool as shown in (Wolz, 1997).

Future Work The next goals in our work being closely related to the results presented here are the following:

- Further Properties:
  Progress properties (Manna and Pnueli, 1995) or liveness in the sense of Petri nets are important

properties to be investigated when validating a model. Again we have to treat the problem that the export should give a valid abstraction of the body. An abstraction should allow to use the properties of the export for reasoning about the whole module. In that way the black box paradigm can be followed and the model still can be validated.

- Import-Export Relation of Properties:
  An essential issue of evolution of component-based systems is the relation between the import and the export of a component (for a more detailed discussion see (Große-Rhode *et al.*, 2000)). The import-export relation in interfaces in the context of interoperability is mainly concerned with the relation of the import of one module ond the export of another one. This is discussed for example as the concept of parameterised contracts in (Reussner, 2001). Here we want to focus on the relation between the import and the export of the same component. This relation describes the internal dependencies that are usually hidden in components and make the exchange of components awkward. We want to state these dependencies explicitly for Petri net modules, for example as am implication between properties required in the import and properties guaranteed in the export.

5. References

Allen, Robert, Remi Douence and David Garlan 1998. Specifying and analyzing dynamic software architectures. In: *Proceedings of the 1998 Conference on Fundamental Approaches to Software Engineering (FASE'98)*. Lecture Notes in Computer Science 1382. Springer Verlag. Lisbon, Portugal. pp. 221–238.

Alur, de Alfaro, Henzinger and Mang 1999. Automating modular verification. In: *CONCUR: 10th International Conference on Concurrency Theory*. LNCS 1664, Springer-Verlag.

Battiston, E., F. De Cindio and G. Mauri 1991*a*. OBJSA Nets: A Class of High-Level Nets Having Objects as Domains. In: *Advances in Petri Nets* (Rozenberg/Jensen, Ed.). Springer.

Battiston, E., F. De Cindio, G. Mauri and L. Rapanotti 1991*b*. Morphisms and Minimal Models for OBJSA Nets. In: *12^th International Conference on Application and Theory of Petri Nets*. Gjern, Denmark. pp. 455–476. extended version: Technical Report i. 4.26, Progretto Finalizzato Sistemi Informatici e Calcolo Parallelo. Consiglio Nazionale delle Ricerche (CNR), Italy, Jan, 1991.

Best, E. 1997. Partial order verification with PEP.. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **29**, 305–328.

Bradfield, J. 1992. *Verifying Temporal Properties of Systems*. Birkhuser. Boston, Basel, Berlin.

Bradfield, J. and C. Stirling 1990. Verifying temporal properties of processes. In: *Lecture Notes in Computer Science; CONCUR'90, Theories of Concurrency: Unification and Extension. (Conference, 1990, Amsterdam, The Netherlands)* (J. C. M. Baeten et al., Ed.). Springer Verlag. Berlin, Germany. pp. 115–125.

Brauer, Wilfried, Robert Gold and Walter Vogler 1991. A survey of behaviour and equivalence preserving refinements of Petri nets. *Lecture Notes in Computer Science; Advances in Petri Nets 1990* **483**, 1–46.

Broy, M. and T. Streicher 1992. Modular functional modelling of Petri nets with individual tokens. *Advances in Petri Nets*.

Buchholz, P. 1994. Hierarchical high level Petri nets for complex system analysis. In: *Application and Theory of Petri Nets*. Vol. LNCS 815. Springer. pp. 119–138.

Chandy, K. M. and J. Misra 1988. *Parallel Program Design: A Foundation*. Addison-Wesley.

Christinsen, S. and N.D. Hansen 1994. Coloured Petri nets extended with channels for synchronous communication. In: *Application and Theory of Petri Nets*. Vol. LNCS 815. Springer. pp. 159–178.

Deiters, W. and V. Gruhn 1994. The FUNSOFT Net Approach to Software Process Management. *International Journal on Software Engineering and Knowledge Engineering* **4**(2), 229–256.

Desel, J., G. Juhás and R. Lorenz 2000. Process semantics of Petri nets over partial algebra. In: *Proceedings of the XXI International Conference on Applications and Theory of Petri Nets* (M. Nielsen and D. Simpson, Eds.). Vol. LNCS 1825. Springer. pp. 146–165.

Fehling, R. 1993. A concept of hierarchical Petri nets with building blocks. In: *Advances in Petri Nets'93*. Springer. pp. 148–168. Lecture Notes in Computer Science 674.

Gajewsky, M., K. Hoffmann and J. Padberg 1999. Place Preserving and Transition Gluing Morphisms in Rule-Based Refinement of Place/Transition Systems. Technical Report 99-14. Technical University Berlin.

Große-Rhode, M., R.-D. Kutsche and F. Bübl 2000. Concepts for the evolution of component-based software systems. Technical Report 2000/11. TU Berlin.

He, X. 1996. A Formal Definition of Hierarchical Predicate Transition Nets. In: *Application and Theory of Petri Nets*. Vol. LNCS 1091. Springer. pp. 212–229.

Jensen, K. 1992. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Vol. 1: Basic Concepts.. EATCS Monographs in Theoretical Computer Science ed.. Springer Verlag.

Kindler, E. 1995. Modularer Entwurf verteilter Systeme mit Petrinetzen. PhD thesis. Technische Universität München, Institut für Informatik.

Kindler, E. 2002. Dawn for component based systems - just a different view. In: *PROMISE 2002 - Prozessorientierte Methoden und Werkzeuge fr die Entwicklung von Informationssystemen* (M. Weske J. Desel, Ed.). pp. 7–13.

Krämer, Bernd 1998. Synchronization constraints in object interfaces. In: *Information Systems Interoperability* (Bernd Krämer, Michael P. Papazoglou and Heinz W. Schnmidt, Eds.). pp. 111–141. Research Studies Press. Taunton, England.

Magee, Jeff, Naranker Dulay, Susan Eisenbach and Jeff Kramer 1995. Specifying distributed software architectures. In: *Proceedings of ESEC '95 - 5th European Software Engineering Conference*. IEEE. Sitges, Spain. pp. 137–53.

Manna, Zohar and Amir Pnueli 1995. *Temporal Verification of Reactive Systems: Safety*. Springer Verlag.

Meseguer, J. and U. Montanari 1990. Petri Nets are Monoids. *Information and Computation* **88**(2), 105–155.

Müller, H. and Weber, H., Eds. 1998. *Continuous Engineering of Industrial Scale Software Systems*. IBFI, Schloß Dagstuhl. Dagstuhl Seminar Report #98092.

Nierstrasz, Oscar 1995. Regular types for active objects. In: *Object-Oriented Software Composition* (Oscar Nierstrasz and Dennis Tsichritzis, Eds.). pp. 99–121. Prentice Hall.

Padberg, J. 2001. Place/Transition Net Modules: Transfer from Algebraic Specification Modules. Technical Report TR 01-3. Technical University Berlin.

Padberg, J. 2002. Petri net modules. *Journal on Integrated Design and Process Technology* **6**(4), 105–120.

Padberg, J. and M. Buder 2001. Structuring with Petri Net Modules: A Case Study. Technical Report TR 01-4. Technical University Berlin.

Reisig, Wolfgang 1998. *Elements of Distributed Algorithms: Modelling and Analysis with Petri Nets*. Springer Verlag.

Reussner, Ralf H. 2001. The use of parameterised contracts for architecting systems with software components. In: *Proceedings of the Sixth International Workshop on Component-Oriented Programming (WCOP'01)* (Wolfgang Weck, Jan Bosch and Clemens Szyperski, Eds.).

Sibertin-Blanc, C. 1994. Cooperative Nets. In: *Application and Theory of Petri Nets'94*. pp. 471–490. Springer LNCS 815.

Stirling, C. 1992. Modal and temporal logics. In: *Handbook of Logic in Computer Science*. Vol. 2, Background: Computational structures. Clarendon Press, Oxford.

Weber, H. 1999. Continuous Engineering of Communication and Software Infrastructures.*Lecture Notes in Computer Science 1577*. pp. 22–29. Springer Verlag. Berlin, Heidelberg, New York.

Wolz, D. 1997. Colimit Computations for Graph Structures and Algebraic Specification Languages. PhD Thesis, TU Berlin, FB Informatik.

A Technicalities

In this appendix we give for the sake of completeness all the definitions, lemmata and proofs required for the two main results. These notions are not neccessary for the understanding of the paper. They are only required for following the central and new proof. So we do not give additional comments.

Definition 16 (Commutative Free Monoids) *Given a set P then the free commuative monoid $(P^{\oplus}, \epsilon, \oplus)$ is constructed freely according to the following equations for $u, v, w \in P^{\oplus}$:*

1. $w \oplus \epsilon = w$ *neutrality*
2. $w \oplus v = v \oplus w$ *commutativity*
3. $w \oplus (v \oplus u) = (w \oplus v) \oplus u$ *associativity*

Elements $w \in P^{\oplus}$ can be given by finite linear sums: $w = \sum_{p \in P} \lambda_p \cdot p$ with $\lambda_p \in \mathbb{N}$. Hence, the usual operations can be extended from natural numbers to linear sums. We use:

$w \oplus w'$	*addition*
$w \ominus w'$	*substraction*
$w \leq w'$	*comparison*

◀

Definition 17 (Restriction of a Linear Sum) *For $w = \sum_{p \in P} \lambda_p \cdot p$ we define the restriction to $p \in P$: $w_{|p} = \lambda_p \cdot p$ And for $P' \subseteq P$ we define: $w_{|P'} = \sum_{p \in P'} \lambda_p \cdot p$ For a mapping $f : P_1 \to P_2$ we use as a abbreviation:$w_{|f(P)} = w_{|f}$* ◀

Definition 18 (Pre-Set, Post-Set) *The pre-set of a net element is given by $\bullet t = \{p | pre(t)_{|p} \neq \epsilon\}$ or $\bullet p = \{t | post(t)_{|p} \neq \epsilon\}$, and analogously for the post-set.* ◀

Definition 19 (Subnets) *$N' \subseteq N$ if and only if $P' \subseteq P$ and $T' \subseteq T$ as well as for pre- and postdomain $pre'(t) = pre(t)_{|P'}$ for all $t \in T'$, post analogously.*
The set of all subnets of N is given by $\mathcal{P}(N) := \{N' \subseteq N\}$. ◀

Definition 20 (Net of a Transition) *Given a transition $t \in T$ for some net N, then $net(t)$ the net surrounding t is given by : $net(t) := (P^t, T^t, pre^t, post^t)$ with*

- *$P^t = \bullet t \cup t\bullet$,*
- *$T^t = \{t\}$, and*
- *$pre^t : T^t \to P^{t\oplus}$ with $pre^t(t) = pre_1(t)$, analogously $post^t$* ◀

Definition 21 (PO-Compatibility Condition (Padberg, 2001)) *Given a plain morphism $f : N_0 \to N_1$ and a substitution morphism $g : N_0 \rightsquigarrow N_2$ the PO-compatibility condition is satisfied if: For all $t_0, t_0' \in T_0$ we have $f_T(t_0) = f_T(t_0')$ implies $\mathfrak{g}_T(t_0) \cong \mathfrak{g}_T(t_0')$.* ◀

Lemma 22 (Pushouts (Padberg, 2001)) *Given a plain morphism $f : N_0 \to N_1$ and a substitution morphism $g : N_0 \rightsquigarrow N_2$ so that the* PO-compatibility condition *is satisfied, then we have the pushout (N_3, f', g')*

$$\begin{tikzcd}
N_0 \arrow[r, "f"] \arrow[d, rightsquigarrow, "g"'] \arrow[dr, phantom, "(1)"] & N_1 \arrow[d, rightsquigarrow, "g'"] \\
N_2 \arrow[r, "f'"] & N_3
\end{tikzcd}$$

$N_3 := (P_3, t_3, pre_3, post_3, M_3^0)$ is given by

- *$P_3 = P_1 +_{P_0} P_2$ is pushout in* **Set**,
- *$T_3 := (T_1 \setminus f_T(T_0)) \uplus T_2$, hence we have: $t_3 \in T_3$ implies $t_3 \in T_1 \dot{\vee}\, t_3 \in T_2$ and,*
- *$pre_3 = \begin{cases} g'_P(pre_1(t_3)) & ; t_3 \in T_1 \\ f'_P(pre_2(t_3)) & ; t_3 \in T_2 \end{cases}$*

  $post_3$ is defined analogously.
- *The initial marking M_3^0 is defined by $M_3^0 = \sum_{p \in P_3} M^0_{3|p}$. And $M^0_{3|p}$ is given by:*

  $$M^0_{3|p} = \begin{cases} {f'_P}^{\oplus}(M^0_{2|p_2}) & ; \text{if } p = {f_P}'(p_2) \text{ and } p \notin g'_P(P_1) \\ {g'_P}^{\oplus}(M^0_{1|p_1}) & ; \text{if } p = g'(p_1) \text{ and } p \notin g(P_2) \\ max({f'_P}^{\oplus}(M^0_{2|p_2}), {g'_P}^{\oplus}(M^0_{1|p_1})) & \\ & ; \text{if } p = g'(p_1) = {f_P}'(p_2) \end{cases}$$

  where obviously define $max(\lambda_1 p, \lambda_2 p) = max(\lambda_1, \lambda_2) \cdot p$.

The morphisms are given by $f' = ({f_P}', f'_T)$ and $g' = (g'_P, \mathfrak{g}'_T)$. ${f_P}'$ and g'_P are defined by the pushout P_3 f'_T and $\mathfrak{g}'_T$ are given below :

$f'_T : T_2 \to \mathcal{P}(N_3)$ *with*

$$f'_T(t) = (f'_P, id_{T_2})(net(t))$$

So f is plain.

$\mathfrak{g}'_T : T_1 \to \mathcal{P}(N_3)$ *with*

$$\mathfrak{g}'_T(t) = \begin{cases} f'_T(t) \circ \mathfrak{g}_T(t_0) & ; t \in f_T(T_0) \\ (g'_P, id_{T_1})(net(t)) & ; else \end{cases}$$

◀

Lemma 23 (Pushouts are Stable under SPP-Morphims) *Given a substitution morphism $f : N_1 \to N_2$, that is safety propery preserving and given a pushout (1) as in Lemma 22. Then we have:*
g is safety property preserving implies g' is safety property preserving.

◀

Proof:
We have to show g' is safety property preserving, provided g is safety property preserving:

1. g' is place preserving. Let $\epsilon \neq pre_3(t) \geq p_3$ for some $p_3 \in g'_P(P_1)$. Then there are two cases as $t \in T_1 \dot{\vee} t \in T_2$.

   (a) $t \in T_1$:
   Since $T_3 := (T_1 \setminus f_T(T_0)) \uplus T_2$ we conclude $t \notin f_T(T_0)$. Due to the definiton of $\mathfrak{g}'_T : T_1 \to T_2$ we have $\mathfrak{g}'_T(t) = (g'_P, id_{T_1})(net(t))$. Due to the definition of $net(t)$ we conclude:
   $pre_3(t) = g'^{\oplus}_P(pre_1(t))$ and $post_3(t) = g'^{\oplus}_P(post_1(t))$.
   And hence: $pre_3(t)_{|g'_P} = g'^{\oplus}_P(pre_1(t))$ and $post_3(t)_{|g'_P} = g'^{\oplus}_P(post_1(t))$.

   (b) $t_3 \in T_2$:
   This implies some $p_2 \in P_2$ and some $p_1 \in P_1$ with $f'_P(p_2) = p_3 = g'_P(p_1)$.
   Due to the pushout properties of P_3 there is some $p_0 \in P_0$ with $f_P(p_0) = p_1$ and $g_P(p_0) = p_2$.
   As g_P is place-preserving we know there is some $t_0 \in T_0$ with $t \in T_2^{t_0}$ so that
   $pre_2(t)_{|g_P} = g^{\oplus}_P(pre_0(t_0))$, and
   $post_2(t)_{|g_P} = g^{\oplus}_P(post_0(t_0))$.

   By definition of $\mathfrak{g}'_T$ with $\mathfrak{g}'_T(f_T(t_0)) = f'_T \circ \mathfrak{g}_T(t_0)$ there is $f_T(t_0) \in T_1$ with $t \in T_3^{f_T(t_0)}$.
   We now conclude:

   $$\begin{aligned} pre_3(t)_{|g'_P} &= f_P{'}^{\oplus}(pre_2(t))_{|g'_P} &&= f_P{'}^{\oplus}(pre_2(t)_{|g_P}) \\ &= f_P{'}^{\oplus}(g^{\oplus}_P(pre_0(t_0)) &&= g'_P{}^{\oplus}(f^{\oplus}_P(pre_0(t_0)) = g'_P{}^{\oplus}(pre_1(f_T(t_0)) \end{aligned}$$

   and the same for :

   $$\begin{aligned} post_3(t)_{|g'_P} &= f_P{'}^{\oplus}(post_2(t))_{|g'_P} &&= f_P{'}^{\oplus}(post_2(t)_{|g_P}) \\ &= f_P{'}^{\oplus}(g^{\oplus}_P(post_0(t_0)) &&= g'_P{}^{\oplus}(f^{\oplus}_P(post_0(t_0)) = g'_P{}^{\oplus}(post_1(f_T(t_0)) \end{aligned}$$

2. g' is marking strict:
   g'_P is injective, as g_P is injective and pushouts in $\mathbf{Set}$ preserve injections.
   Moreover, we need to show $M^0_{3|g'_P} = g'_P{}^{\oplus}(M^0_1)$. We only need to investigate $p_3 \in g'_P(P_1)$.
   We then have two cases:

   (a) $M^0_{3|g'_P} = g'_P{}^{\oplus}(M^0_1)$ for $p_3 \in P_3 \setminus f'_P(P_2)$,

   (b) and for $p_3 = g'_P(p_1) = f'_P(p_2)$ we have:
   $M^0_{3|g'_P} = max(f'_P{}^{\oplus}(M^0_{2|p_2}), g'_P{}^{\oplus}(M^0_{1|p_1})) = g'_P{}^{\oplus}(M^0_1)$
   due to the following estimation:

   $$\begin{aligned} g'_P{}^{\oplus}(M^0_{1|f_P(p_0)}) &\geq g'_P{}^{\oplus} \circ f^{\oplus}_P(M^0_{0|p_0}) \\ &= f'_P{}^{\oplus} \circ g^{\oplus}_P(M^0_{0|p_0}) \\ &= f'_P{}^{\oplus}(M^0_{2|_2}) \end{aligned}$$

√