Formal Modelling and Analysis of Reconfigurable Object Nets Based on the RON Editor

Sarkaft Shareef

December 5, 2010

Diploma Thesis

Supervisors: Prof. Dr. Hartmut Ehrig
Dr. Claudia Ermel

Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin

Eidesstattliche Erklärung

Die selbständige und eigenhändige Ausfertigung versichert an Eides statt
Berlin, 5. Dezember 2010

Sarkaft Shareef

Acknowledgement

I would like to thank Prof. Dr. Hartmut Ehrig, Dr. Claudia Ermel, Frank Hermann, Tony Modica and Karsten Gabriel for their professional support and constructive criticism. Furthermore I would like to thank Grischa Weberstädt for his help with proof reading my thesis. Special thanks to my wife Hozan for her support and understanding.

Abstract

The first contribution of this thesis is to give a formal definition of the reconfigurable object nets. We give the definition of the corresponding algebraic higher-order nets with individual tokens, which are algebraic high-level nets with individual tokens with fixed signature and algebra.

The second contribution is to realize techniques to analyze the existence of permutation equivalences in the firing sequences of the algebraic high-level nets.

Zusammenfassung

Der erste Beitrag dieser Diplomarbeit besteht darin, eine formale Definition von Rekonfigurierbaren Objekt-Netzen anzugeben. Wir geben die Definition der dazugehörigen Algebraischen Higher-Order Netze mit Individuellen Token, welche Algebraische High-Level-Netze mit Individuellen Token sind, die eine feste Signatur und Algebra haben.

Der zweite Beitrag besteht darin, Techniken zu realisieren, die die Existenz von permutativen Äquivalenzen in den Schaltsequenzen von Algebraischen High-Level-Netzen anlysieren.

Contents

1 Introduction

The relevance of visual languages which support the modelling of dynamic systems adapting to a changing environment has increased during the last years [BEMS08]. RON Editor [RON10] is a visual editor and simulator developed as Eclipse plug-in to support the visual specification of the "controlled rule-based transformation" [BM08] of the place/transition nets (P/T nets). Moreover, it simulates the mobility of the P/T nets and rules as well as the firing behaviour of the low- and high-level transitions.

Reconfigurable Object Nets (RONs) are high-level nets with two types of tokens: object nets which are place/transition nets (P/T nets) and rules. Thus RONs follow the paradigm "nets as tokens" [Val98] on the one hand and "nets and rules as tokens" [HME05] on the other hand which allow us to refine and change the structure of these object nets (reconfiguration) at runtime by integrating the rule-based net transformation of object nets with the transition firing.

Often there are several different alternative orders of execution steps of dynamic systems with equivalent behaviours, i.e. the same input produces the same output in arbitrary order of executing the same steps [Her08]. For investigating whether a permutation equivalent order of a given execution sequence exists, the techniques of analysis according to [HCEK10] (*Efficient Process Analysis of Transformation Systems Based on Petri nets*) can be used. The verification and analysis of the existence of such a permutation equivalence for a given scenario or a given firing sequence in RONs will fail to work, because the complete formal definition of RONs hast not been realized yet. A formal basis for RONs was given in [HME05] two years before the development of RON editor. There are several aspects of RONs missing in [HME05]. Furthermore the notion of *individual tokens* [MGE+10] has to be considered in the formal definition of RONs in order to be able to realize the analysis based on permutation equivalence [HCEK10].

The first aim of this thesis is to give the formal definition and the formal modelling of RONs as Algebraic High-Level nets (AHL) [PER95] in order to be able to find a possible (maybe more efficient) permutation equivalence of a firing sequence in RON using the analysis techniques in [HCEK10]. We intend to formalize RONs as algebraic higher-order nets based on individual tokens approach. We review the formalization of Place/Transition nets with Individual tokens (PTI nets) and lift the results to the Algebraic High-Level nets with Individual tokens (AHLI nets) [MGE+10]. The formalization of RONs completes with the definition of Algebraic Higher-Order nets with Individual tokens ($AHOI_{PTI}$) which are AHLI nets with a given constant signature and algebra.

The second aim of this thesis is to use the techniques of analysis introduced in [HCEK10] to verify the existence of possibly permutation equivalent firing sequences [HCEK10]. For this purpose we translate the firing sequences into transformation sequences using the property of *Equivalence of Canonical Transformations and Firing of AHLI Nets* in [MGE+10]. After proving the existence of effective unions in category **AHLINets** we analyze the resulting transformation sequence by using the techniques of analysis of permutation equivalence of transformation sequences. The process skeleton of the transformation sequences must be built as process net (P/T net).

After giving an overview on the RON editor and RONs in Section 2 supported with an example, we review the formal definition of PTI nets in Section 3. In Section 4 we give the definition of Algebraic Higher-Order nets with Individual tokens ($AHOI_{PTI}$) by presenting its signature and algebra after reviewing the definition of AHLI nets. We give the formal definition and modelling of RONs in Section 5 by giving the formalization of the low- and high-level nets as well as the formalization of firing behaviour of the low- and high-level transitions. In Section 6 we realize the analysis based on permutation equivalence and demonstrate it with an example. In Section 7 we give the summary of the thesis and the possible future work.

2 Introduction to Reconfigurable Object Nets (RONs)

In this section we give a short introduction to Reconfigurable Object Nets (RONs) [RON10]. First we describe the construction of Reconfigurable Object Nets and their editor. After giving the different types of their high-level transitions and firing behaviour we demonstrate a (Producer/Consumer) example as RON.

The Reconfigurable Object Nets Editor (RON editor) was developed in a student project in summer 2007 at the Technical University of Berlin as plug-in for Eclipse [RON10]. This visual editor uses as plug-ins the Eclipse Modeling Framework (EMF) [EMF10] and the Graphical Editor Framework (GEF) [GEF10]. The main aim of developing such a visual editor is to support the visual specification of the "controlled rule-based net transformation" [BM08] of the place/transition nets (short P/T nets). Furthermore the editor contains a simulator using the graph transformation engine *The **A**ttributed **G**raph **G**rammar System* (**AGG**) [AGG10] "to simulate the application of rules and thus firing of high-level transitions in the RONs created with the editors" [RON10].

2.1 Reconfigurable Object Nets (RONs)

Reconfigurable Object Nets (RONs) are high-level nets. They have two types of tokens: object nets and rules [BEMS08]. Object nets are P/T nets which can be moved through the high-level net by firing the high-level transitions [RON10]. The structure of an object net can also be changed by applying a net transformation rule to it. Rules my also be moved through the high-level net by firing the high-level transitions in order to change the structure of the object nets. There are also two types of places: rule places and object net places. According to this classification there are rules as tokens on rule places and there are object nets as tokens on the other one. Hence Reconfigurable Object Nets (RONs) follow the paradigm "nets as tokens" [Val98] on the one hand and "nets and rules as tokens" [HME05] on the other hand so that we can change the structure of the object nets (reconfiguration) [BM08].

2.2 RON Editor

The environment of the Reconfigurable Object Nets Editor (RON editor) [BEHM07] consists of four main components [BEHM07, BM08, BEMS08]. We give a short introduction to these four components below:

RON Tree View: This part contains the complete RON model as tree view. The graphical views of each button can be opened by double-click [BEHM07, BM08, BEMS08]. The view [1] in Figure 1 on page 11 shows the tree view of RON editor.

Object Net Editor: This part is the editor for object nets, i.e. a graphical place/transition net editor [BEHM07, BM08, BEMS08]. We can simulate the firing behaviour of the transitions of the object nets. (see view [2] in Figure 1 on page 11).

Transformation Rule Editor: This is the editor for transformation rules, it consists of three editor panels [BM08] (view [3] in Figure 1 on page 11). One for the negative

application condition (NAC) [EEPT06], one for the left-hand side (LHS) and the third for the right-hand side (RHS) [BEMS08]. Note that each panel is an object net editor with additional button "Map" that enables us to match the object net of LHS to the object net of RHS or to the object net of NAC.

High-Level Net Editor: In this part of the editor the high-level net of a RON can be drawn [BEMS08]. The high-level net part controls the behaviour of the object nets as well as the rule applications to modify these object nets. In View [4] of Figure 1 we can recognize that every button of the transitions (of kind STANDARD, FIRE, APPLYRULE and SPLIT) has its own graphical icon for visualization [BM08]. Furthermore there are also two more icons: one is the blue coloured container marked by "O" for object net places and the other is the green coloured container marked by "R" for rule places [BEHM07].

The transitions of the high-level net of a RON can be fired by double-click [BM08]. For simulations of the transition of the kind APPLYRULE there is an internal extension of the RON editor [RON10] by a converter to AGG. This engine is able to analyze and perform algebraic graph transformations [AGG10]. The simulation of the remaining transition kinds, i.e. STANDARD, FIRE and SPLIT is implemented directly in the RON editor [BEHM07, BM08].

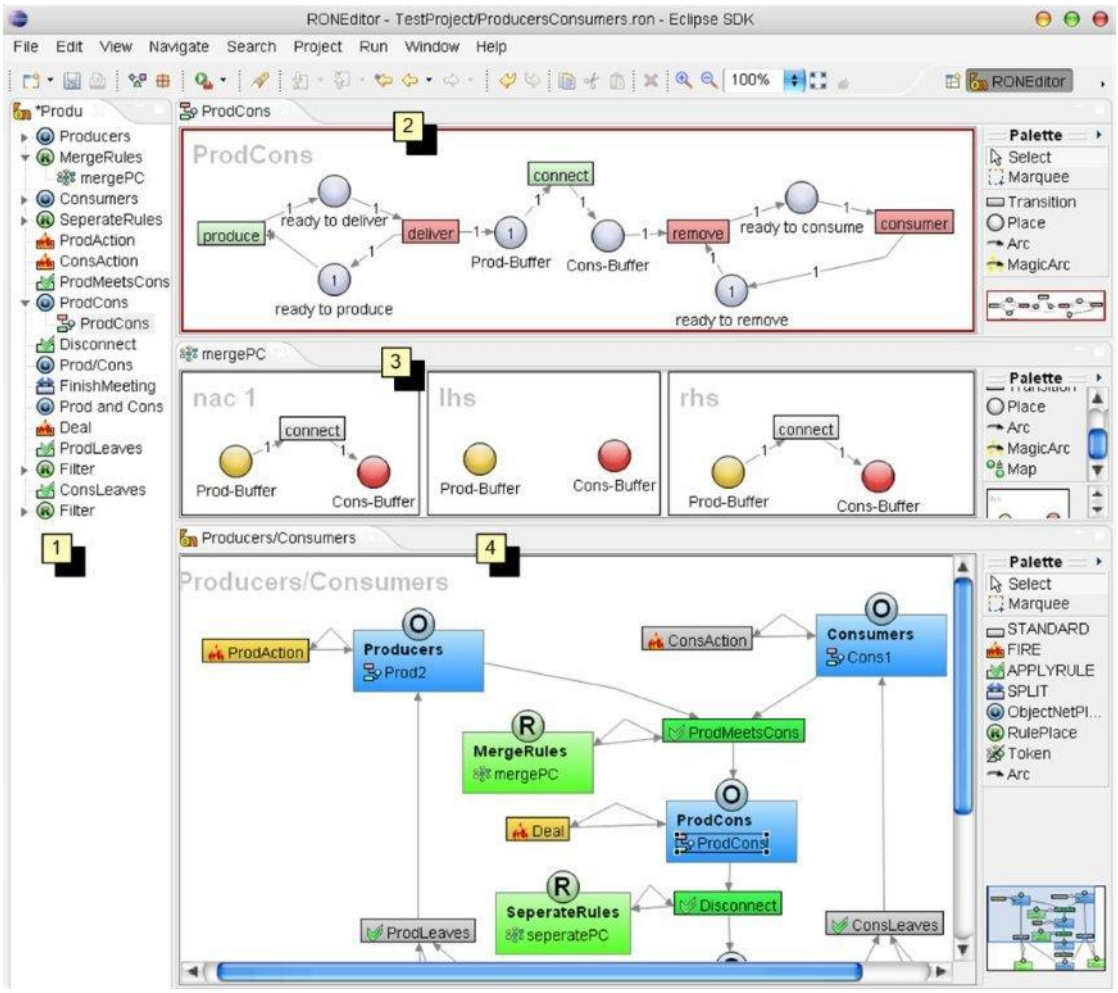


Figure 1: RON Editor

2.3 High-Level Transition Types and firing behaviour

There are four kinds of high-level transitions in RONs according to their firing behaviour and functionality [RON10]. The transitions of kind STANDARD, the transitions of kind FIRE, the transitions of kind APPLYRULE and the transitions of kind SPLIT.

2.3.1 The transitions of kind STANDARD

The conditions that must be satisfied for the transitions of this kind to be enabled to be fired are:

- In its pre-domain there is at most one rule place and arbitrarily many object net places.

- In the post-domain there are arbitrarily many rule places if there is at least one rule place on the pre-domain. Otherwise we have no rule places.
- In the post-domain there are arbitrarily many object net places if there is at least one object net place on the pre-domain. Otherwise there are no object net places connected to transition.
- A transition is activated if there is at least one token on each pre-domain place.
- The firing behaviour of this transition is to take a token from each object net place in its pre-domain. The object nets in the post-domain are the result of building the disjoint union of the object nets of the post-domain before firing with the new coming object nets after firing this transition. Each rule place in post-domain will get a rule from the rule place in its pre-domain.

The main function of the transitions of the kind STANDARD is to move the object nets through the high-level net as well as to apply additional functions on these object nets or rules, e.g. to copy an object net or a rule, to delete an object net, to build the disjoint union of object nets etc [RON10]. According to these additional functions, we can classify the transitions of kind STANDARD into a rich variety of transitions, which are actually STANDARD transitions. In order to give a precise picture of the functionality and strategy of these transitions we give an example with illustration for each one of them below.

Copy Object Net: This transition takes an object net from its pre-domain, copies it and puts a copy of this object net on every object net place in its post-domain i.e. after firing this transition of kind STANDARD we get an object net on each object net place on the post-domain for each object net of the object net place in its pre-domain.

Example. Figure 2 shows how the transition *copy* works.

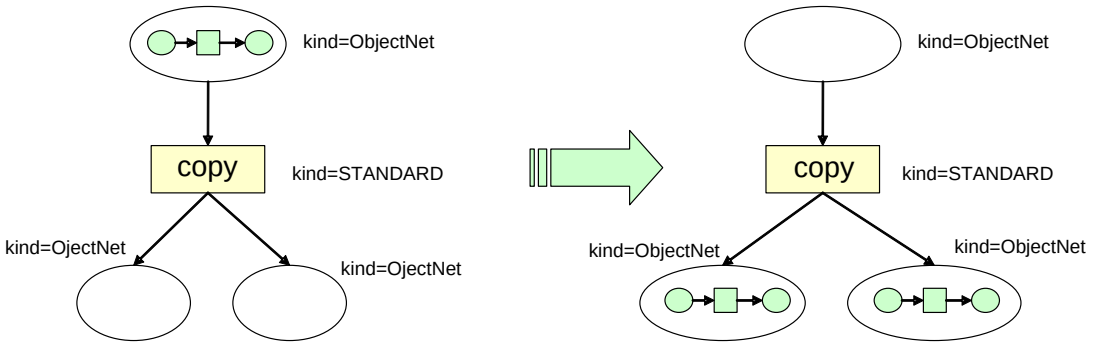


Figure 2: Copy Object Net

Copy Object Net and Move Rule: This transition of kind STANDARD takes object nets from object net places in its pre-domain and a rule from its rule place in the

pre-domain. It copies the object nets and puts a copy of them on each object net place in the post-domain. It also puts the rule, that has been taken from the rule place in its pre-domain, into the the rule place in its post-domain.

Example. Figure 3 shows how the transition *copy&move* works.

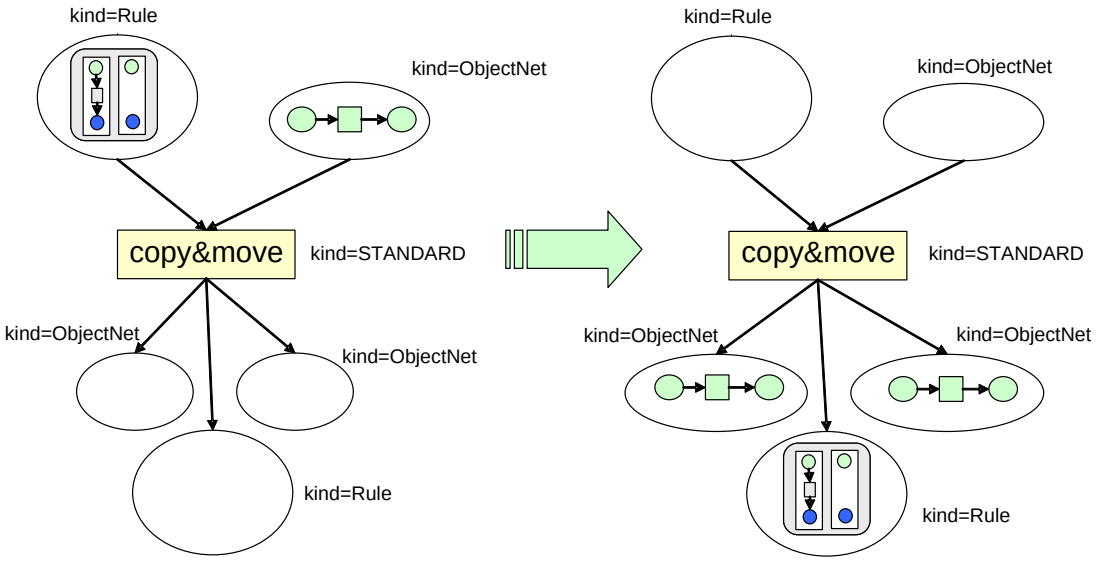


Figure 3: Copy Object Net and Move Rule

Move Object Net: By this function the transition takes the object nets from object net places in its pre-domain and puts them on an object net place in its post-domain.

Example. Figure 4 on the next page shows how the transition *move* works.

Delete Object Net: This transition of kind STANDARD deletes object nets by taking them from the pre-domain and putting them nowhere. Note that this transition of kind STANDARD has no places in the post-domain, i.e. this transition just swallows the object nets.

Example. Figure 5 on the following page shows how the transition *delete* works.

Union of Object Nets: This transition takes the object nets of different object net places in the pre-domain and puts them together on an object net place in the post-domain by building the disjoint union of them.

Example. Figure 6 on page 15 shows how the transition *union* works.

Union and Copy of Object Nets: This transition of the kind STANDARD transition builds the disjoint union of the objects net of each object nets place in its pre-domain, duplicates the result of the disjoint union of them and puts a copy of the resulting object net on each object net place in the post-domain so that we gain a copy of that object net on every object net place of its post-domain..

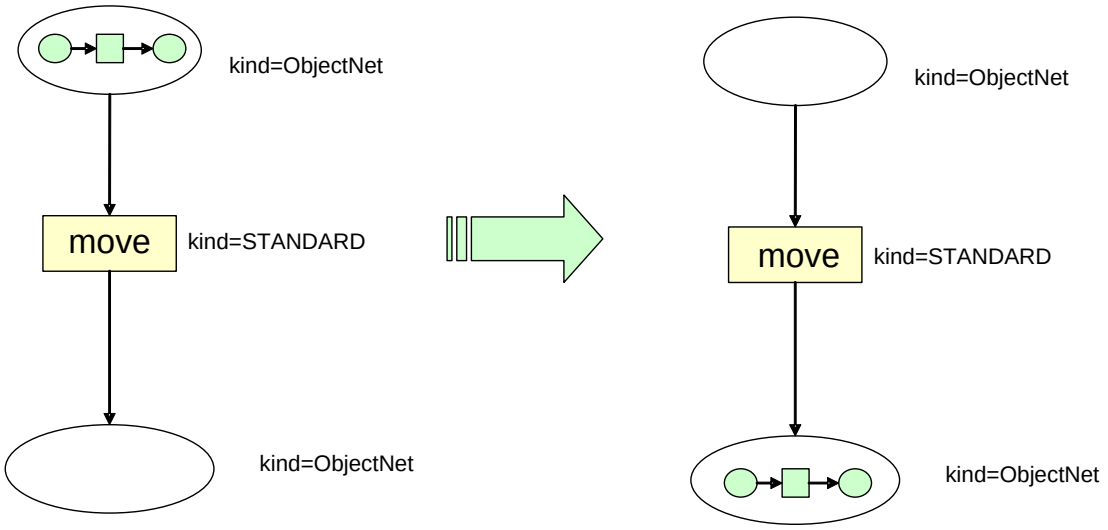


Figure 4: Move Object Net

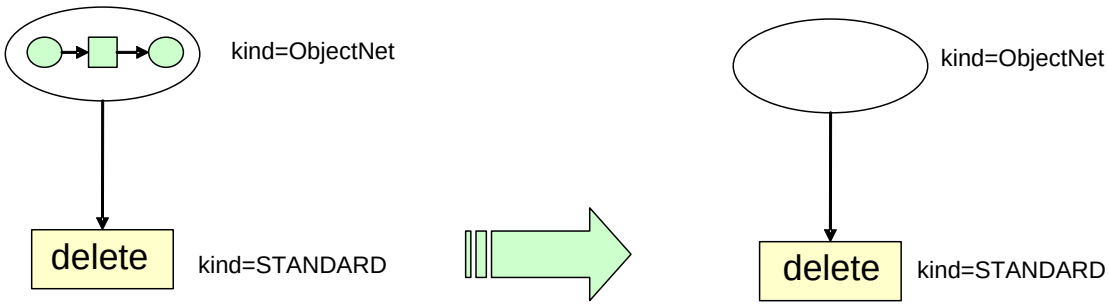


Figure 5: Delete Object Net

Example. Figure 7 on the facing page shows how the transition *union&copy* of kind STANDARD works.

Union and Copy of Object Nets and Copy of Rule: This function of the transition kind STANDARD builds the disjoint union of each object nets in the pre-domain and puts a copy of the resulting object net on each object net place in the post-domain after duplicating it. Moreover, it takes a rule from the rule place in its pre-domain, duplicates it and puts a copy of this rule on each rule place in its post-domain. After firing this transition we have on every object net place of the post-domain a copy of the disjoint union of each object net from the pre-domain. Furthermore, we have a copy of the rule on every rule place in the post-domain.

Example. Figure 8 on page 16 shows how the transition *union&copy* works.

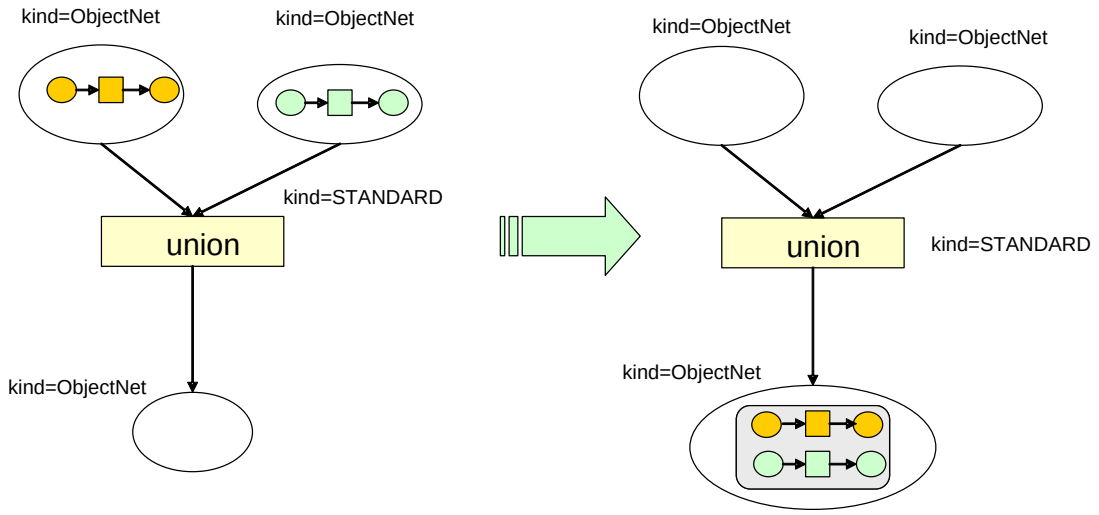


Figure 6: Union of Object Nets

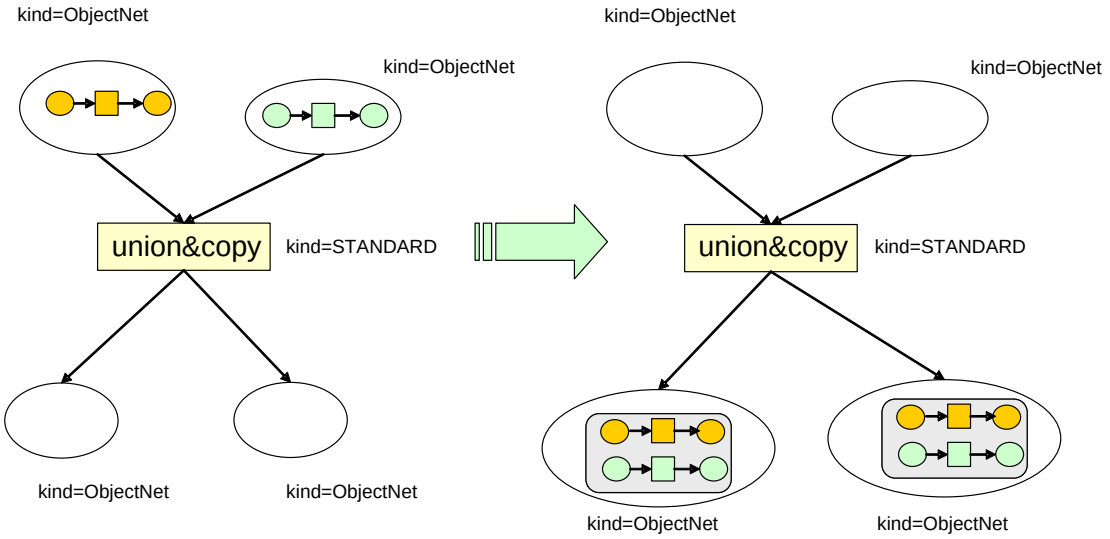


Figure 7: Union and Copy of Object Nets

2.3.2 The FIRE Transtion

The main function of the transitions of kind FIRE is to fire a selected transition of an object net. This kind of transition has exactly one object net place in the pre-domain and arbitrary many object net places in its post-domain. The object net places on pre-domain and post-domain may be the same one. They can be also different of each other. The transitions of kind FIRE can be enabled to fire, if there is at least one object net on the pre-domain and at least one transition (low-level transition) of this object net is

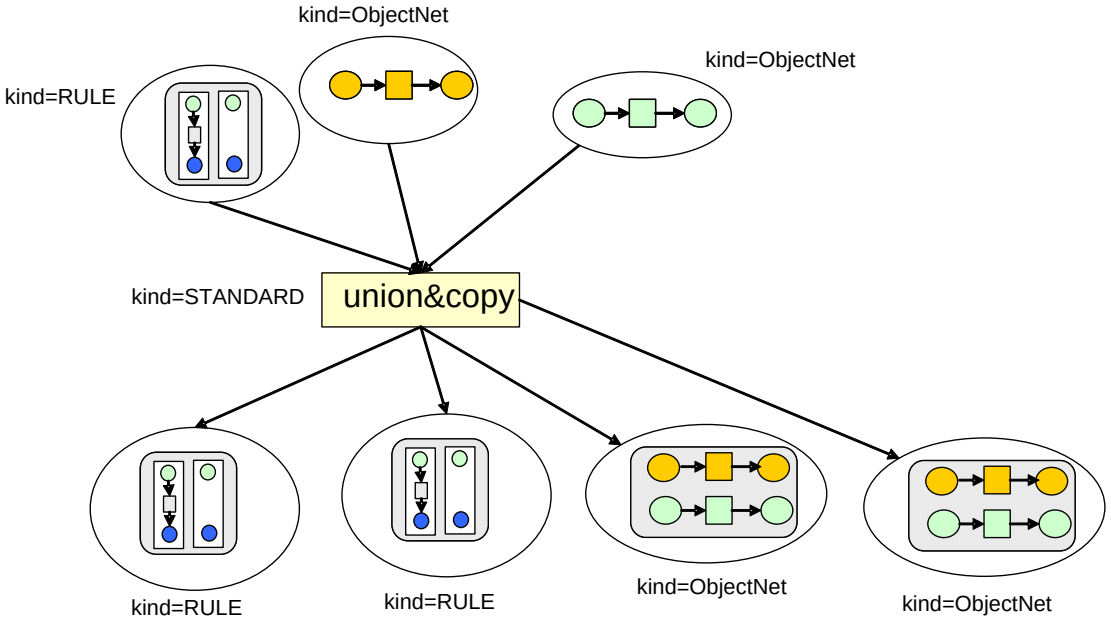


Figure 8: Union and Copy of Object Nets and Copy of Rule

enabled to be fired [RON10].

Select Object Net and fire enabled Transition: By this option the transition takes an object net from the pre-domain which has been selected by the user (we assume that there is more than one object net on the object net place of its pre-domain) and fires the enabled transition of this object net. If there are more than one enabled transition of the object net, a transition must be selected manually by the user of the Editor. Hence the firing of the high-level transition involves the firing of the low-level transition, i.e. the transition of the object net. After firing this transition (the high-level transition) we get the object net in the post-domain place such that its enabled (selected) transition (the low-level transition) has been fired.

Example. Figure 9 shows how the transition *ON_ action* of kind *FIRE* fires an enabled transition of an object net (the upper one).

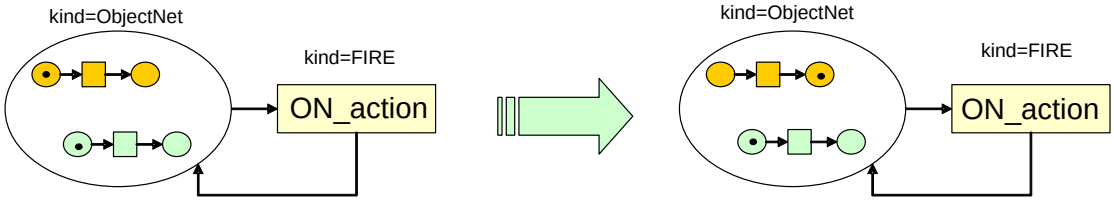


Figure 9: Select Net, Fire enabled Transition

Fire Transition and Copy Object Net: This is an option of the transition FIRE so that it fires an enabled transition of an object Net (low-level transition) from its pre-domain as well as duplicates the resulting object net after triggering its transition and puts the copies on different object nets places in its post-domain.

Example. Figure 10 shows how the transition *fire&copy* of kind FIRE fires an enabled transition of an object net and copies the object net after triggering its transition.

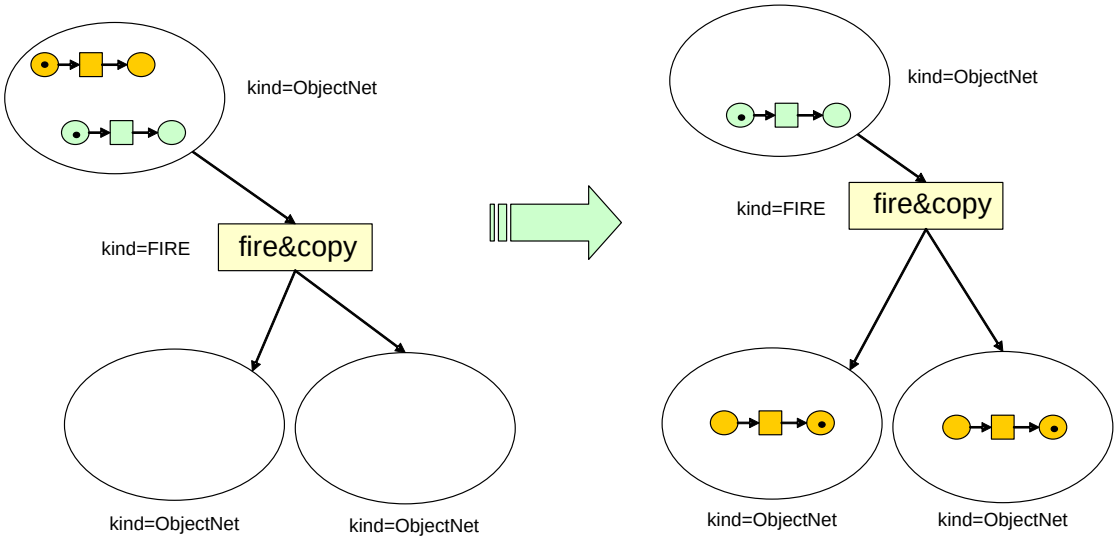


Figure 10: Fire and Copy Object Net

2.3.3 The APPLYRULE Transition

The requirements by the transitions of kind APPLYRULE to be enabled to fired are to have at least one object net place and exactly one rule place in pre-domain and in post-domain arbitrary many object net places and arbitrary many rule places.

This kind of transition is enabled to be fired iff there is at least one object net on each object net place in its pre-domain and there is at least one rule on the rule place in its pre-domain as well and the rule is applicable to the disjoint union of all involved object nets in this firing step [BM08]. If the conditions of being enabled to be fired are satisfied the transition of kind APPLYRULE takes one object net from each object net place in its pre-domain and one rule from the rule place in its pre-domain. After building the disjoint union of these object nets, the transition applies the taken rule to the resulting object net of the built disjoint union of each object nets (Sect. 3.4 on page 31). It puts a copy of the modified object net through the rule-based transformation on every object net place in its post-domain. The rule is also put on every rule place on the post-domain [RON10].

Example. Figure 11 shows how the transition *union&apply* of kind APPLYRULE works.

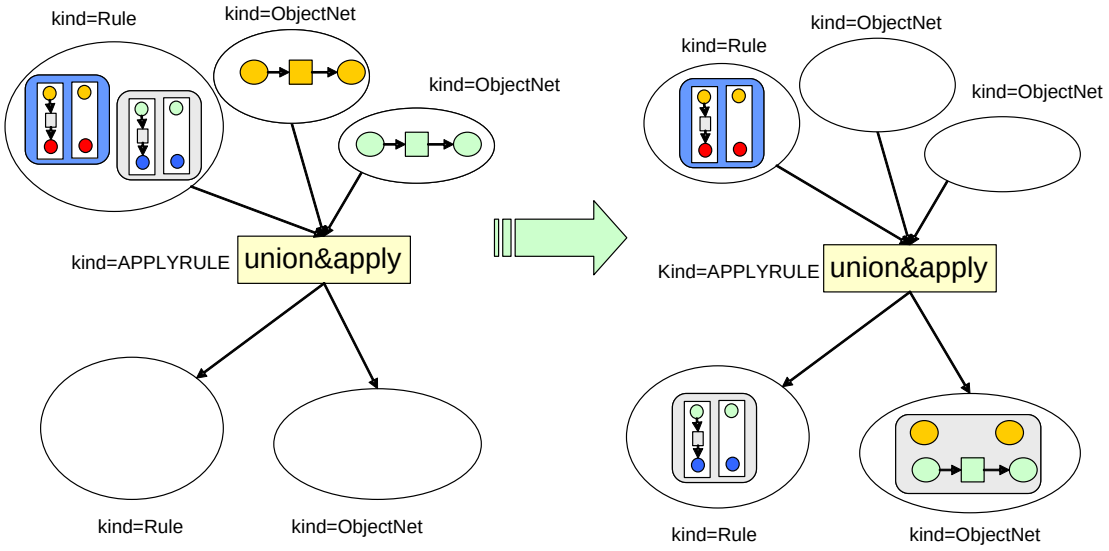


Figure 11: Apply one Rule to the union of two Object Nets

2.3.4 The SPLIT Transition

This kind of transition is able to take an object net which consists of the disjoint union built of n components of object nets and splits it into n separate object nets [RON10]. This transition of kind SPLIT is enabled to be fired iff there is exactly one object net place in its pre-domain and one in its post-domain. And there must exist no rule places, neither on its pre-domain nor on its post-domain. This transition is enabled to be fired if there an object net (may be empty) on the object net place in its pre-domain.

When the conditions to be enabled to be fired for this transition are satisfied, the transition can fire. It takes one selected object net in its pre-domain. If this selected object net is empty or consists of one single component of object net, the transition takes this object net from its pre-domain and puts it on the object net place its post-domain. Otherwise, if the selected object net consists of the disjoint union of more than one object net components the transition takes this object net, separates its components and puts every component as a single object net on the object net place on its post-domain.

Example. Figure 12 on the facing page shows how the transition of kind SPLIT splits an *object net* which consists of three components into three *object nets*.

2.4 Example: Producer/Consumer

In this sections we demonstrate the modelling of distributed producer and consumer system [BEHM07]. The producers and consumers are modelled as object nets (P/T

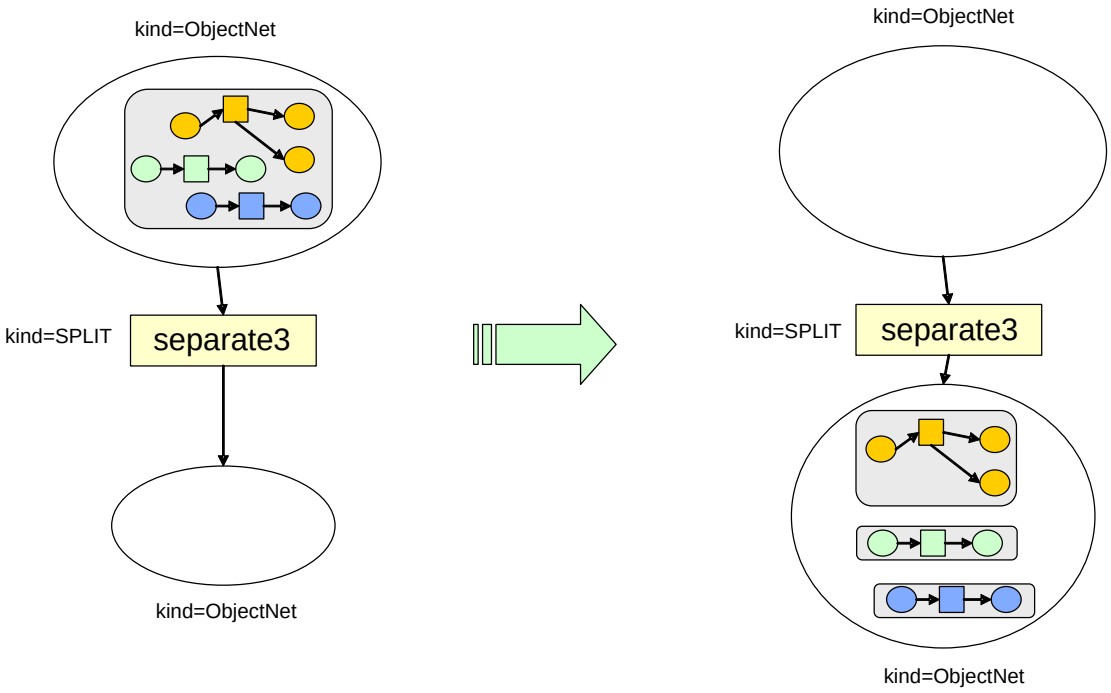


Figure 12: Split an Object Net into three components

nets). The High-Level support their mobilities and activities, i.e. they can be moved through the High-Level net, their structure can be changed and their firing behaviour can be simulated (Figure 13 on the next page).

Example. Figure 13 on the following page depicts the modelling of *producer/consumer*.

In initial state the producer and consumer are separated from each other, i.e. they are independent object nets without any interaction between them. They are also, in initial state, placed on different places of the High-Level net (Figure 15 on page 21). The producer nets are placed on the place *Producers* and the consumer nets on the place *Consumers* of kind ObjectNet. A producer net can produce items by firing the transitions *produce* and *deliver* successively and putting tokens on its place *prodBuffer* by firing the High-Level transition *prodAction* of kind FIRE (2.3.2 on page 15) which triggers the enabled transitions of producer nets. A consumer net can also consume items from its place *consBuffer* by firing the transitins *remove* and *consume* successively. The High-Level transition *consAction* of kind FIRE is responsible for triggering the transitions of consumer nets on this place *Consumers* if they are enabled.

Example. Figure 14 on page 21 depicts the *object net* visualization in *RON-Editor.*.

Example. Figure 15 on page 21 shows the initial marking of the Producer/Consumer net.

A producer net may be glued to an consumer net by firing the High-Level transition *prodMeetsCons* of kind APPLYRULE (2.3.3 on page 17). This High-Level transition

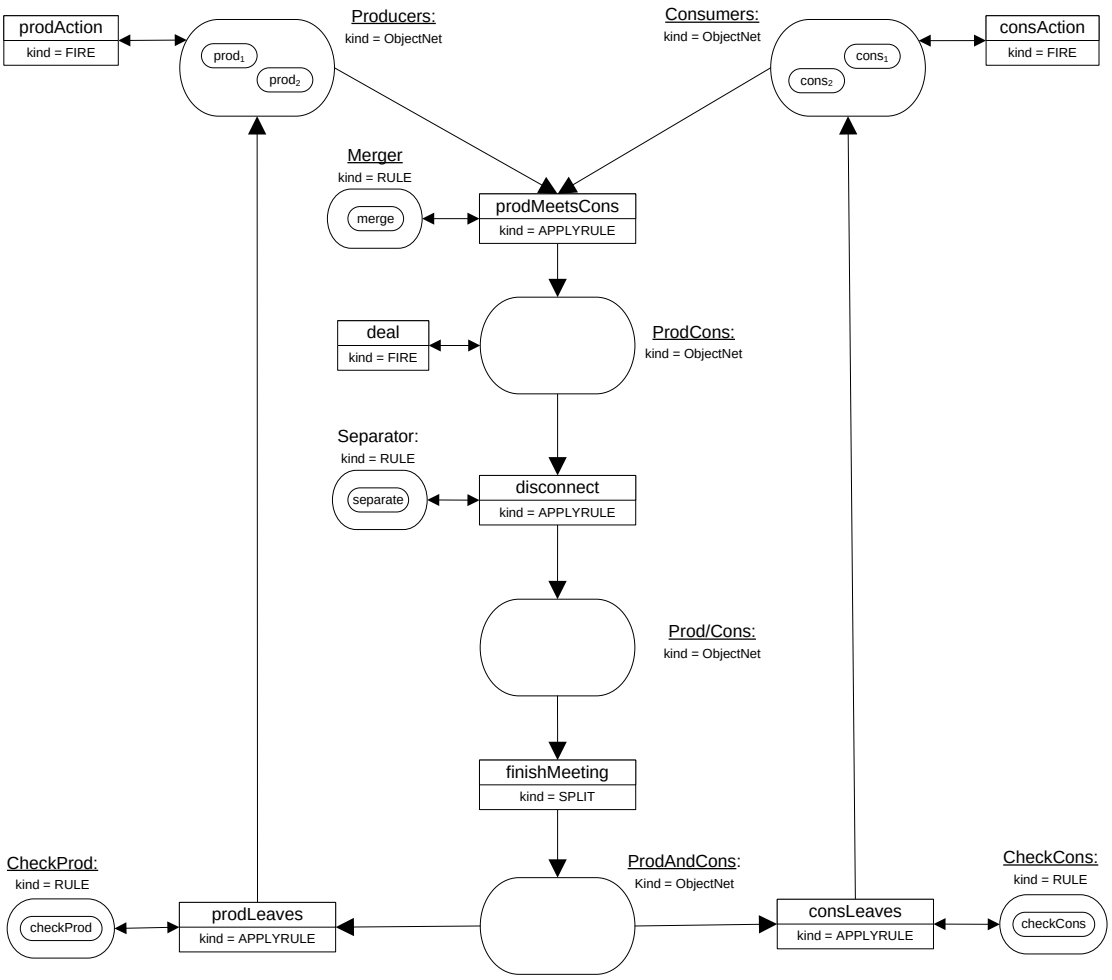


Figure 13: Producer and Consumer

takes a producer net from the place *Producers* and a consumer net from the place *Consumers* in its pre-domain and merges them together by applying the rule *merge* from the rule place *Merger* of kind Rule (Figure 16 on page 22 and Figure 17 on page 22). This rule adds a new transition between the *prodBuffer* and *consBuffer* if they are not connected.

Example. Figure 16 on page 22 shows the rule *merge*

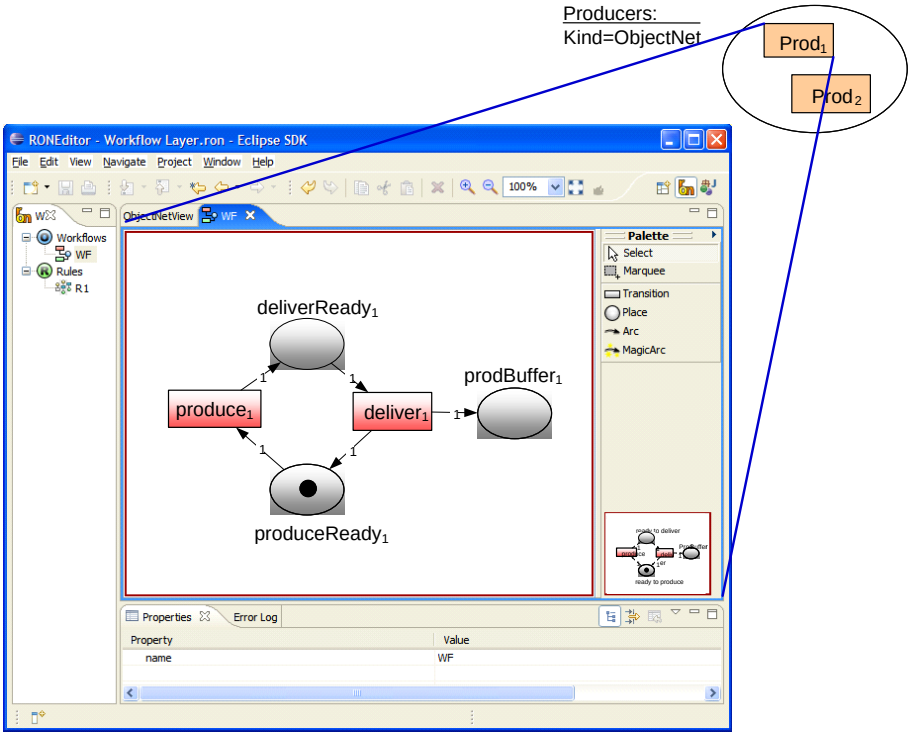


Figure 14: Object Net in RON-Editor

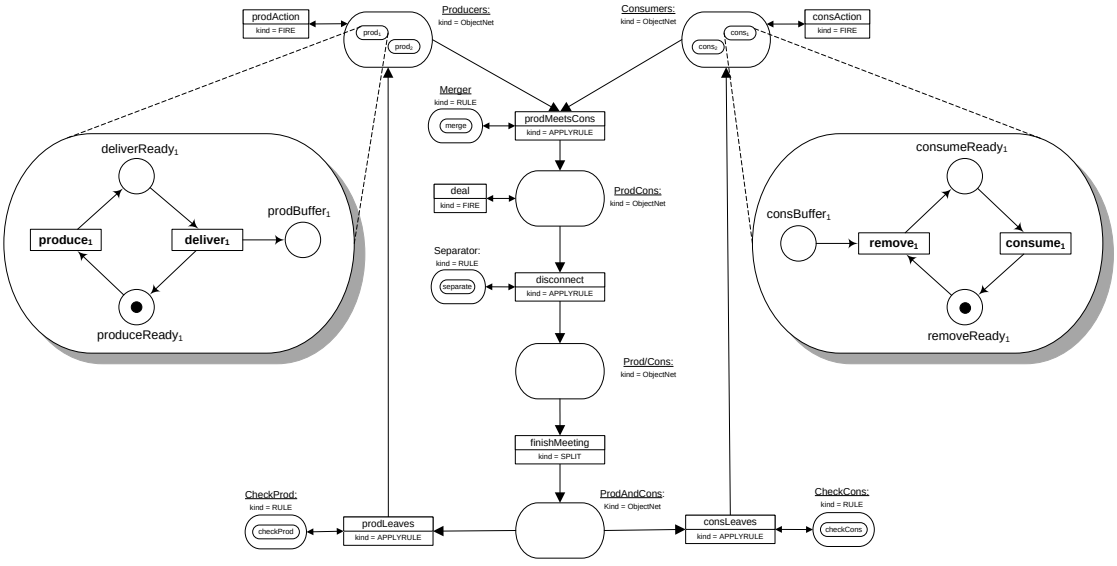


Figure 15: Producer and Consumer

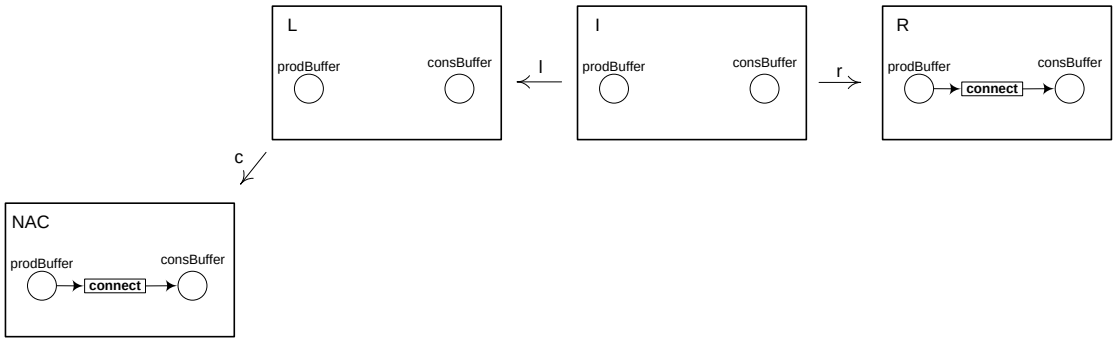


Figure 16: The Rule merge

Example. Figure 17 shows the visualization of rule *merge* in RON-Editor.

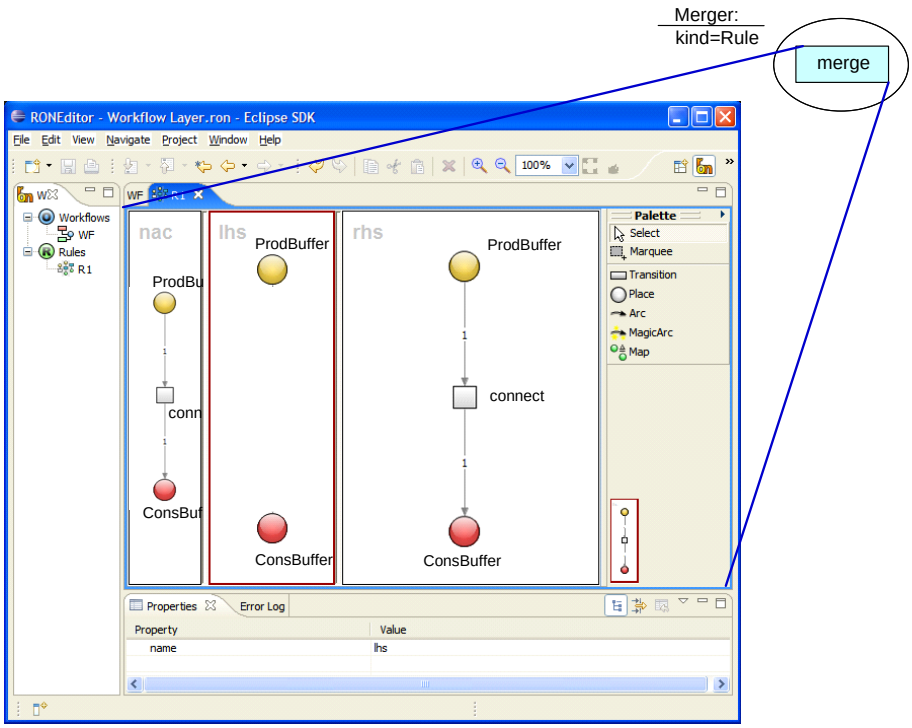


Figure 17: The Rule merge in RON-Editor

After the successful rule application and rule-based transformation (see Sect. 3.4 on page 31) we get a connected producer/consumer net *prodcons* (Figure 19 on page 25) on the place *ProdCons* of kind ObjectNet on the post-domain. Now producer and consumer can interact with each other by firing the High-Level tansition *deal* of kind FIRE which triggers an enabled (selected) transition of the new glued object net *prodcons* (Figure **??** on page ??). The consumer can get the items produced from the producer in *prodBuffer*

by firing the transition *connect*. Note that the other Low-Level transitions can also be selected and fired if they are enabled to be fired, i.e. *produce, deliver, consume* and *remove*. Thats mean that the producer is able to produce more items and put them on *prodBuffer* and the consumer is also able to remove the items from *consBuffer*.

Example. Figure 18 on the next page shows the transition *deal* fires the transition *connect*.

The transition *disconnect* of kind APPLYRUlE (Sect. 2.3.3 on page 17) separates (disconnects) the place *prodBuffer* of producer net from the place *consBuffer* of consumer net. It, the transition *disconnect*, takes an object net from the place *ProdCons* and the rule *separate* form the place *Separator* of kind Rule and applies it to the taken producer/consumer object net. The rule *separate* deletes the transition *connect* which glues the producer net *prod* with the consumer net *cons* (Figure 20 on page 25). After the rule *separate* has been applied we gain an object net consisting of two components of object nets, a producer component and a consumer component, i.e. the disconnected object net is still on object net consisting of the disjoint union of a producer component and a consumer component (Figure 21 on page 25).

Example. Figure 19 on page 25 shows the object net $prod_1 cons_1$.

Example. Figure 20 on page 25 shows the rule *separate*

Example. Figure 21 on page 25 shows the object net after deleting the transition *connect*
.

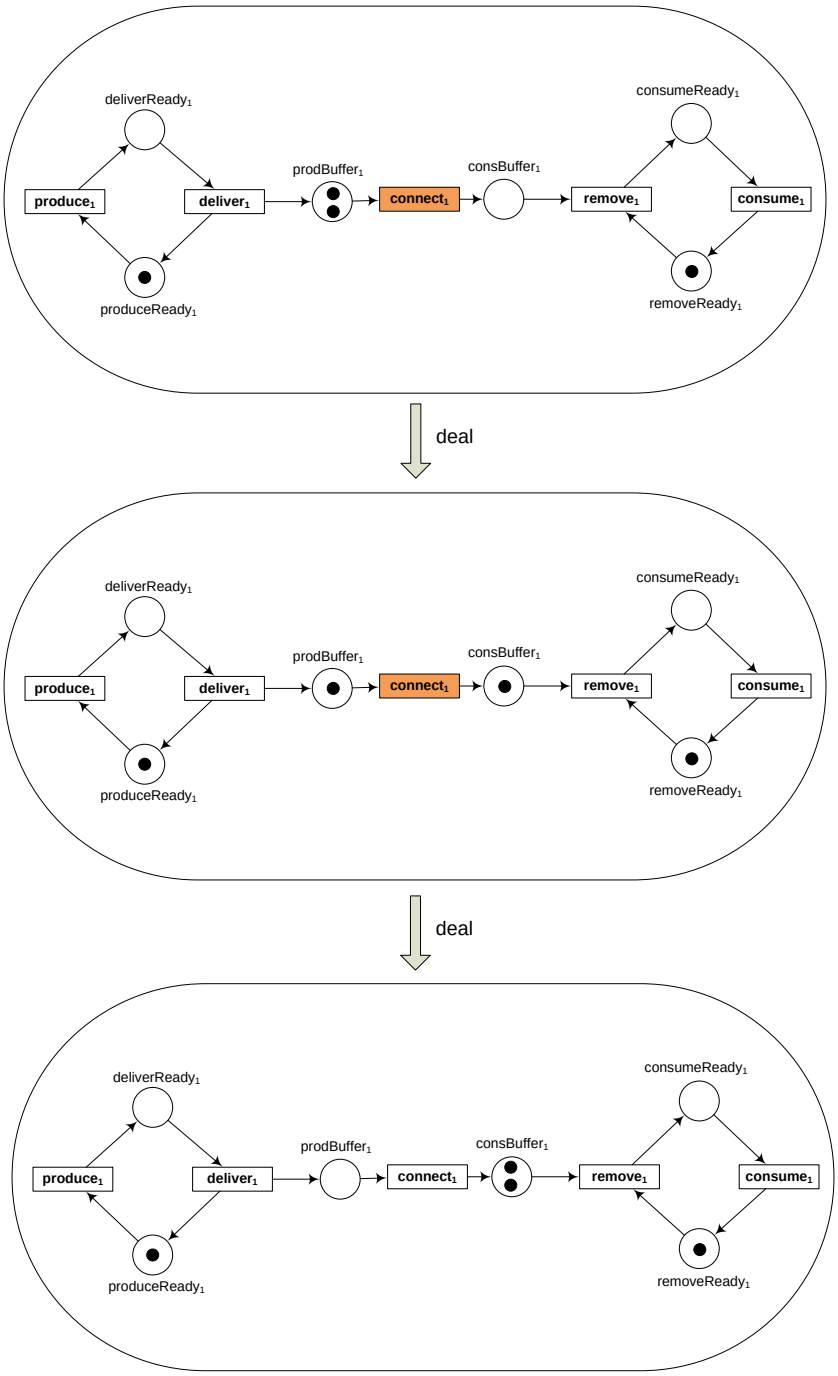


Figure 18: Deal

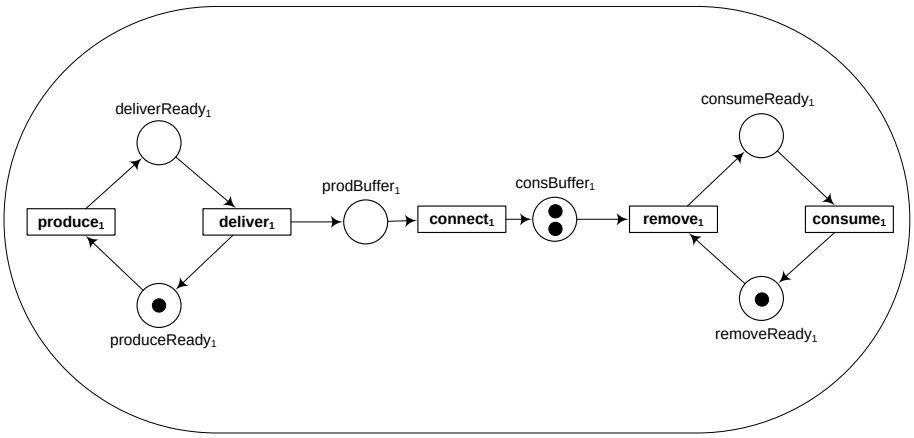


Figure 19: $prod_1 cons_1$

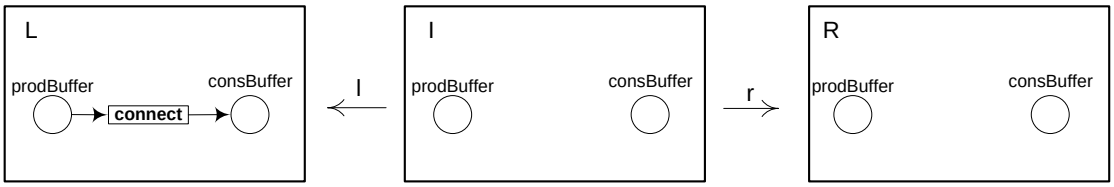


Figure 20: The Rule separate

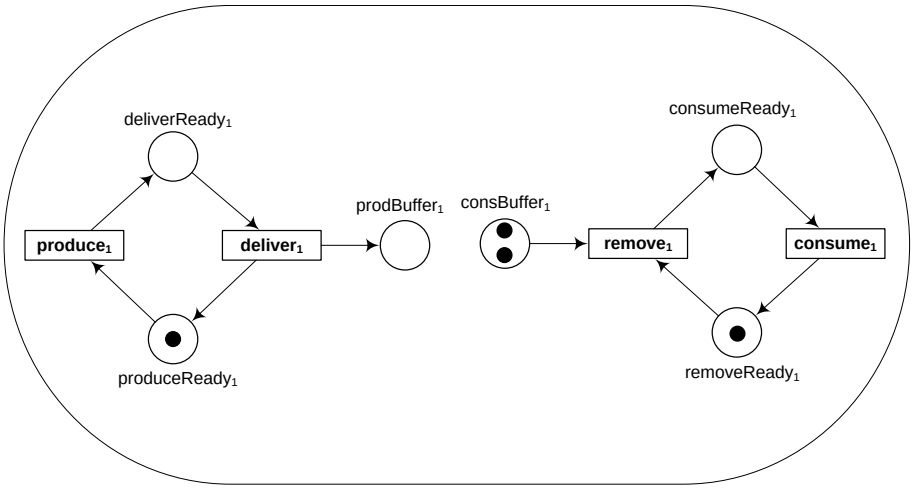


Figure 21: Object net consists of two components

The transition *finishMeeting* of kind SPLIT (Sect. 2.3.4 on page 18) takes the object nets from the place *Prod/Cons* in its pre-domain consisting of producer net and consumer net as components, splits them into two independent producer net and consumer net (Figer 22) and puts them on *ProdAndCons* of kind ObjectNet in its post-domain.

Example. Figure 22 shows the object nets $prod_1$ and $cons_1$ after splitting them.

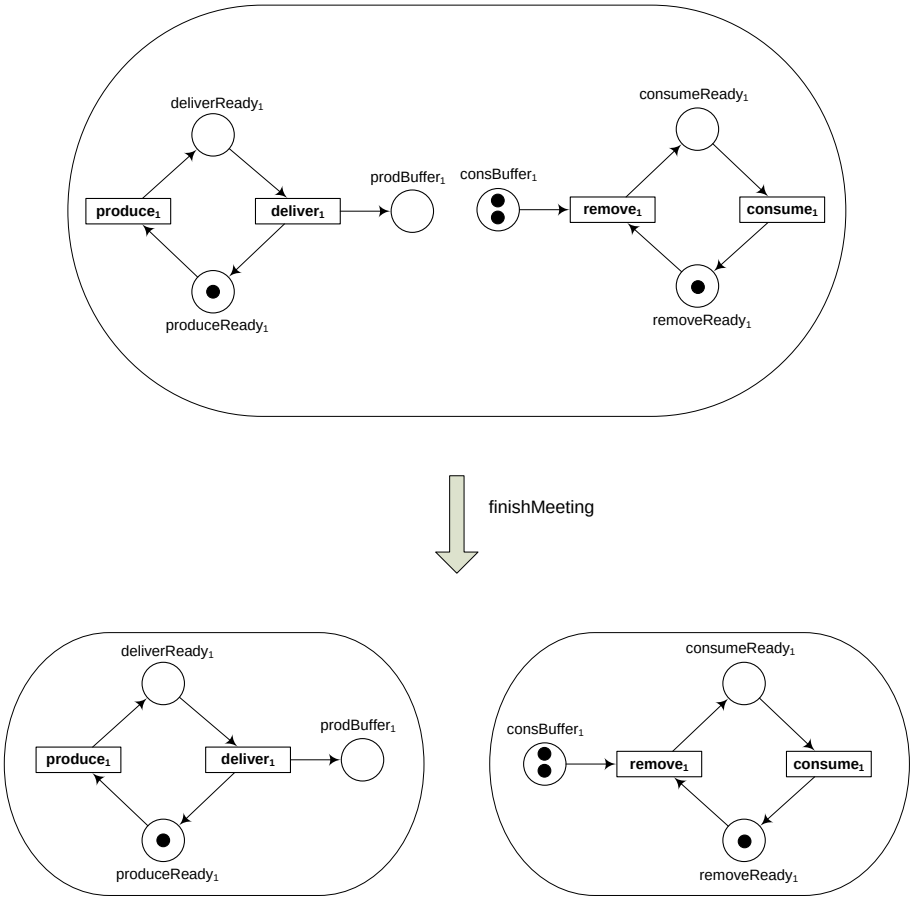


Figure 22: Object nets $prod_1$ and $cons_1$

From the place *ProdAndCons* the producer nets and consumer nets can return to the places of the initial state, i.e. the producer nets to place *Producers* and the consumer nets to place *Consumers* by firing the transitions *prodLeaves* and *consLeaves* of kind APPLYRULE respectively. These transitions take the object nets from the *ProdAndCons* and apply the rule *checkProd* (Figure 23 on the next page) on transition *prodLeaves* and the rule *checkCons* (Figure 24 on the facing page) on transition *consLeaves*. These rules do not change the structure of the object net but it serve as compare rules, in order to prevent that a producer net goes to the place *Consumers* and vice versa.

Example. Figure 23 on the next page shows the rule *checkProd*

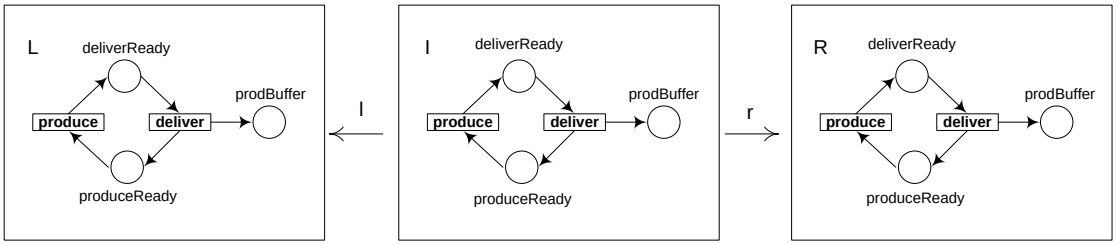


Figure 23: The Rule checkProd

Example. Figure 24 shows the rule *checkCons*

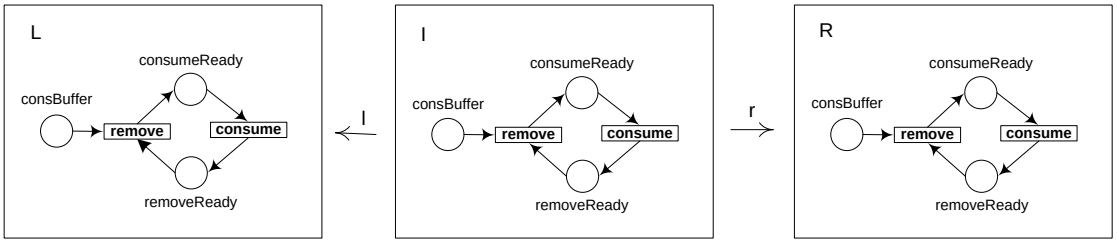


Figure 24: The Rule checkCons

3 Place/Transition Nets with Individual Tokens (PTI Nets)

In this section we review the formalization of low-level Petri nets equipped with markings of individual tokens [MGE+10]. After describing their firing behaviour and introducing rules for the transformation of marked nets we show how special transformations correspond to transition firing steps.

3.1 Definition and Firing Behaviour of PTI Nets

With the definition of nets with individual tokens, we follow the concept "Petri nets are monoids" from [MM90]:

Definition 3.1 (Place/Transition Nets with Individual Tokens (PTI)).
We define a marked P/T net with individual tokens, short PTI net, as

$$NI = (P, T, pre, post, I, m),$$

where

- $N = (P, T,\ pre, post : T \to P^{\oplus})$ is a classical P/T net, where $P^{\oplus}$ is the free commutative monoid over P,
- I is the (possibly infinite) set of individual tokens of NI, and

- $m : I \to P$ is the marking function, assigning the individual tokens to the places.

Further, we introduce some additional notations:

- the environment of a transition $t \in T$ as $ENV(t) = PRE(t) \cup POST(t) \subseteq P$ with

$$PRE(t) = \{p \in P | pre(t)(p) \neq 0\},$$
$$POST(t) = \{p \in P | post(t)(p) \neq 0\},$$

Now that we have marked Petri nets, also called Petri systems, we have to define their behaviour as firing steps. Because we have individual tokens, we have to consider a possible firing step in the context of a selection of tokens.

Definition 3.2 (Firing of PTI Nets)**.**
A transition $t \in T$ in a PTI net

$$NI = (P, T, pre, post, I, m)$$

is *enabled* under a *token selection* (M, m, N, n), where

- $M \subseteq I$,
- m is the token mapping of NI,
- N is a set with $(I \setminus M) \cap N = \emptyset$,
- $n : N \to P$ is a function,

if it meets the following *token selection condition*:

$$\left[\sum_{i \in M} m(i) = pre(t)\right] \wedge \left[\sum_{i \in N} n(i) = post(t)\right]$$

If an enabled t fires, the follower marking (I', m') is given by

$$I' = (I \setminus M) \cup N, \quad m' : I' \to P \text{ with } m'(x) = \begin{cases} m(x), & x \in I \setminus M \\ n(x), & x \in N \end{cases}$$

leading to $NI' = (P, T, pre, post, I', m')$ as the new PTI net in the *firing step* $NI \overset{t}{\rightarrowtail} NI'$ via (M, m, N, n).

Remark (Token Selection). The purpose of the token selection is to specify exactly which tokens should be consumed and produced in the firing step. Thus, $M \subseteq I$ selects the individual tokens to be consumed, and N contains the set of individual tokens to be produced. Clearly, $(I \setminus M) \cap N = \emptyset$ must hold because it is impossible to add an individual token to a net that already contains this token. m and n relate the tokens to their carrying places. It would be sufficient to consider only the restriction $m_{|M}$ here or to infer it from the net but as a compromise on symmetry and readability we denote m in the token selection.

For the next subsection we need a category of Petri systems.

3.2 Example

Example. Figure 25 shows the graphical representation of a PTI net NI with $I = \{x_1, y_1, x_2, x_3\}$, $m(x_1) = m(x_2) = p_1$, $m(x_3) = p_2$, $m(y_1) = p_3$.

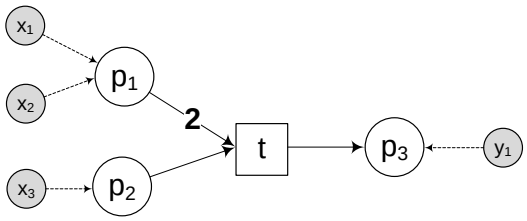


Figure 25: PTI net NI before firing the transition t

In our net N the transition t is *enabled* under the *token selection* (M, m, N, n), where

- $M = \{x_1, x_2, x_3\}$
- $N = \{y_2\}$
- $n(y_2) = \{p_3\}$.

After the enabled transition t fires, we get the following marking (I', m'), where

$$I' = (I \backslash M) \cup N = (\{x_1, y_1, x_2, x_3\} \backslash \{x_1, x_2, x_3\}) \cup \{y_2\} = \{y_1, y_2\}$$

$m' : I' \to P$ with

$$m'(y_1) = \{p_3\}$$
$$m'(y_2) = \{p_3\}$$

and the new PTI net

$$NI' = (P,\ T,\ pre,\ post,\ I',\ m') .$$

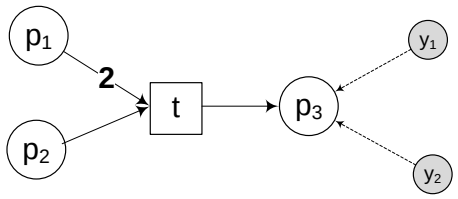


Figure 26: PTI net NI' after firing the transition t

3.3 PTI Net Morphisms and Category PTINets

Definition 3.3 (PTI Net Morphisms and Category **PTINets**).
Given two PTI nets $NI_i = (P_i, T_i, pre_i, post_i, I_i, m_i)$, $i \in \{1, 2\}$, a PTI net morphism is a triple of functions $f = (f_P : P_1 \to P_2, f_T : T_1 \to T_2, f_I : I_1 \to I_2) : NI_1 \to NI_2$, such that the following diagrams commute (componentwise for pre_i and $post_i$):

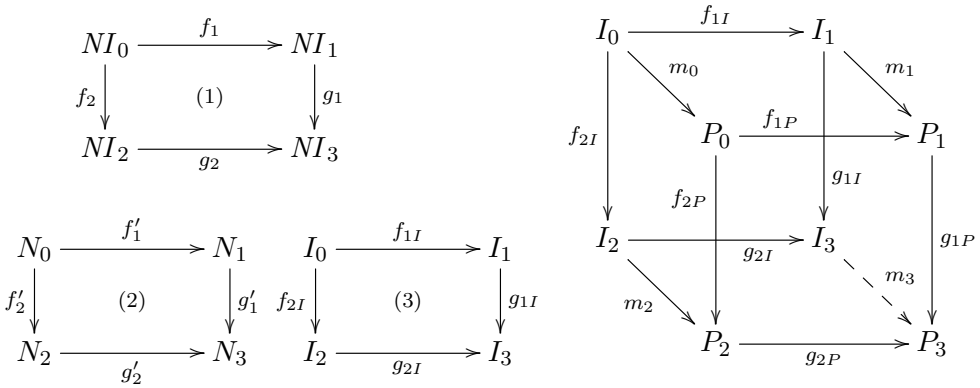
$$\begin{array}{ccc} T_1 & \overset{pre_1}{\underset{post_1}{\rightrightarrows}} & P_1^{\oplus} \\ {\scriptstyle f_T}\downarrow & = & \downarrow{\scriptstyle f_P^{\oplus}} \\ T_2 & \overset{pre_2}{\underset{post_2}{\rightrightarrows}} & P_2^{\oplus} \end{array} \qquad \begin{array}{ccc} I_1 & \xrightarrow{m_1} & P_1 \\ {\scriptstyle f_I}\downarrow & = & \downarrow{\scriptstyle f_P} \\ I_2 & \xrightarrow{m_2} & P_2 \end{array}$$

or, explicitly, that $f_P^{\oplus} \circ pre_1 = pre_2 \circ f_T, f_P^{\oplus} \circ post_1 = post_2 \circ f_T, f_P \circ m_1 = m_2 \circ f_I$.

The category **PTINets** consists of all PTI nets as objects with all PTI net morphisms.

Remark. For a general PTI net morphism f we do not require that its token component f_I is injective in order to have pushouts, pullbacks, and a $\mathcal{M}$-adhesive category. But later we may require injectivity of f_I for rule and match morphisms and to have morphisms preserving firing behaviour.

Fact 3.4 (Construction of Pushouts in **PTINets**).
Pushouts in **PTINets** are constructed componentwise in **PTNets** and **Sets**. So, (1) is a PO in **PTINets** iff (2) is a PO in **PTNets** and (3) is a PO in **Sets** with the components of the **PTINets** morphisms, where $m_3 : I_3 \to P_3$ is induced by PO object I_3 in the commuting cube below (whose front is the PO of place sets).



If f_{1X}, for $X \in \{P, T, I\}$, is injective then g_{2X} is as well; analogously for components of f_2 and g_1.

Example. Figure 27 on page 36 shows two pushouts in **PTINets**.

3.4 PTI Nets Transformation

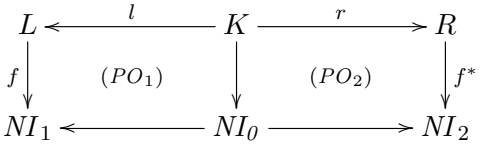
This section is about rule-based transformation of PTI nets. We use the double pushout approach, which has also been used in [EHP+08] and stems from [EEPT06]. We are going to characterize the applicability of rules at some match by initial pushouts.

Definition 3.5 (PTI Transformation Rules)**.**
A PTI transformation rule is a span of injective **PTINets** morphisms

$$\varrho = (L \xleftarrow{l} K \xrightarrow{r} R).$$

Remark. In contrast to [EHP+08], rule morphisms for PTI rules are not required to be marking-strict in order to obtain an $\mathcal{M}$-adhesive category [MGE+10]. This allows to arbitrarily change the marking of a PTI net by applying rules with corresponding places and individual tokens in L and R.

Definition 3.6 (PTI Transformation)**.**
Given a PTI transformation rule $\varrho = (L \xleftarrow{l} K \xrightarrow{r} R)$ and a PTI net NI_1 with a PTI net morphism $f : L \to NI_1$, called the match, a direct PTI net transformation $NI_1 \overset{\varrho,f}{\Longrightarrow} NI_2$ from NI_1 to the PTI net NI_2 is given by the following double-pushout diagram (DPO) diagram in the category **PTINets**:



To be able to decide whether a rule is applicable at a certain match, we formulate a gluing condition for PTI nets, such that there exists a pushout complement of the left rule morphisms and the match if (and only if) they fulfill the gluing condition. The correctness of the gluing condition is shown via a proof on initial pushouts over matches, according to [EEPT06].

3.5 Gluing Condition in Category PTINets

Definition 3.7 (Gluing Condition in **PTINets**).
Given PTI nets K, L, and NI and PTI morphisms $l : K \to L$ and $f : L \to NI$. We define the set of identification points[1]

$$IP = IP_P \cup IP_T \cup IP_I$$

with

- $IP_P = \{x \in P_L \mid \exists x' \neq x : f_P(x) = f_P(x')\}$,
- $IP_T = \{x \in T_L \mid \exists x' \neq x : f_T(x) = f_T(x')\}$,
- $IP_I = \{x \in I_L \mid \exists x' \neq x : f_I(x) = f_I(x')\}$,

the set of dangling points[2]

$$DP = DP_T \cup DP_I$$

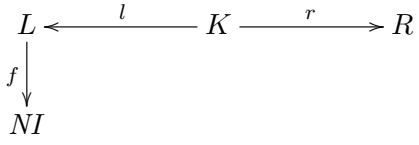
with

- $DP_T = \{p \in P_L \mid \exists t \in T_{NI} \setminus f_T(T_L) : f_P(p) \in ENV(t)\}$,
- $DP_I = \{p \in P_L \mid \exists i \in I_{NI} \setminus f_I(I_L) : f_P(p) = m_{NI}(i)\}$,

and the set of gluing points[3]

$$GP = l_P(P_K) \cup l_T(T_K) \cup l_I(I_K)$$

We say that l and f satisfy the gluing condition if $IP \cup DP \subseteq GP$. Given a PTI rule $\varrho = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$, we say ϱ and f satisfy the gluing condition iff l and f satisfy the gluing condition.



In order to construct the initial pushout for a match $f : L \to NI$, we define the *boundary* over the match f, which is the minimal subnet containing all places, transitions, and individual tokens that must not be deleted by the application of a rule with left hand side L such that there exists a pushout complement.

[1] That is, all elements in L that are mapped non-injectively by f.
[2] That is, all places in L that would leave a dangling arc, if deleted.
[3] That is, all elements in L that have a preimage in K.

Definition 3.8 (Boundary in **PTINets**).
Given a morphism $f : L \to NI$ in **PTINets**. The boundary of f is a PTI net

$$B = (P_B, T_B, pre_B, post_B, I_B, m_B)$$

with

- $P_B = DP_T \cup DP_I \cup IP_P \cup P_{IP_T} \cup P_{IP_I}$
- $P_{IP_T} = \{p \in P_L \mid \exists t \in IP_T : p \in ENV(t)\}$
- $P_{IP_I} = \{p \in P_L \mid \exists i \in IP_I : p = m_L(i)\}$
- $T_B = IP_T$
- $pre_B(t) = pre_L(t)$
- $post_B(t) = post_L(t)$
- $I_B = IP_I$
- $m_B(i) = m_L(i)$

together with an inclusion $b : B \to L$.

Well-definedness.

$pre_B, post_B : T_B \to P_B^{\oplus}$:
The well-definedness follows from the well-definedness of Definition **A.9** (Boundary in **PTNets**) [MGE+10] and the fact that B has the same set of transitions as the boundary in Definition **A.9** (Boundary in **PTNets**) [MGE+10] and the set of places in Definition Definition **A.9** (Boundary in **PTNets**)[MGE+10] is a subset of P_B.

$m_B : I_B \to P_B$:
Let $i \in I_B$. Then we have $i \in IP_I$ and hence

$$m_B(i) = m_L(i) \in P_{IP_I} \subseteq P_B$$

$b : B \to L$:
We obtain an inclusion morphism $b : B \to L$ from the fact that $pre_B, post_B$ and m_B are restrictions of the respective functions in L.

□

The following facts about the gluing condition and initial pushouts hold in all $\mathcal{M}$-adhesive categories (**PTINets**, $\mathcal{M}$) whose morphism class $\mathcal{M}$ of monomorphisms contains at least inclusions (for concrete instantiations see "**Petri Nets with Individual Tokens are $\mathcal{M}$-adhesive Categories**"[MGE+10]). The next fact completes the construction of initial pushouts for matches and shows that the construction of the boundary in Def. 3.8 complies to the categorical notion of boundaries in Def. **A.1**(Boundary, Initial Pushout)[MGE+10].

3.6 Initial Pushout in PTINets

Fact 3.9 (Initial Pushout in **PTINets**).
Given a morphism $f : L \to NI$ in **PTINets**, the boundary B of f and the PTI net

$$C = (P_C, T_C, pre_C, post_C, I_C, m_C)$$

with

- $P_C = (P_{NI} \setminus f_P(P_L)) \cup f_P(b_P(P_B))$
- $T_C = (T_{NI} \setminus f_T(T_L)) \cup f_T(b_T(T_B))$
- $I_C = (I_{NI} \setminus f_I(I_L)) \cup f_I(b_I(I_B))$
- $pre_C(t) = pre_{NI}(t)$
- $post_C(t) = post_{NI}(t)$
- $m_C(i) = m_{NI}(i)$

Then diagram (1) where $g := f|_B$ is an initial pushout in **PTINets**.

$$\begin{array}{ccc} B & \xrightarrow{b} & L \\ {\scriptstyle g}\downarrow & (1) & \downarrow{\scriptstyle f} \\ C & \xrightarrow[c]{} & NI \end{array}$$

Proof. (see Proof B.1 in [MGE+10]). □

The following two facts show the correspondence between the gluing condition in **PTINets** and the categorical gluing condition (see Def. A.2 in [MGE+10] "Categorical Gluing Condition"), which is a necessary and sufficient condition for the (unique) existence of pushout complements in all $\mathcal{M}$-adhesive categories.

Fact 3.10 (Characterization of Gluing Condition in **PTINets**).
Let $l : K \to L$ and $f : L \to NI$ be morphisms in **PTINets** with $l \in \mathcal{M}$.
The morphisms l and f satisfy the gluing condition in **PTINets** if and only if they satisfy the categorical gluing condition.

Proof. (see Proof B.2 in [MGE+10]). □

Fact 3.11 (Gluing Condition for PTI Transformation).
Given a PTI rule $\varrho = (L \xleftarrow{l} K \xrightarrow{r} R)$ and a match $f : L \to NI$ into a PTI net $NI = (N, I, m : I \to P_{NI})$. The rule ϱ is applicable on match f, i.e. there exists a (unique) pushout complement NI_0 in the diagram below, iff ϱ and f satisfy the gluing condition in **PTINets**.

$$\begin{array}{ccccc} L & \xleftarrow{\quad l \quad} & K & \xrightarrow{\quad r \quad} & R \\ {\scriptstyle f}\downarrow & (PO) & \downarrow{\scriptstyle f'} & & \\ NI & \prec\!\!- - - - & NI_0 & & \end{array}$$

Proof. By Fact 3.10 ϱ and f satisfy the gluing condition in **PTINets** if and only if ϱ and f satisfy the categorical gluing condition, which by Fact A.3 ind [MGE+10] is a necessary and sufficient condition for the (unique) existence of the pushout complement NI_0. □

3.7 Correspondence of Transition Firing and Rules

Now that we can manipulate a net's marking with rules, we have a look at rules that simulate firing steps for a certain transition under a token selection. We show that firing of a transition corresponds to a canonical application of a special rule construction, called transition rules. With this correspondence we can easily show that token-injective PTI morphisms preserve firing steps.

Definition 3.12 (PTI Transition Rules)**.**
We define the *transition rule* for a transition $t \in T$ enabled under a token selection $S = (M, m, N, n)$ in a PTI net $NI = (P, T, pre, post, I, m)$ as the rule

$$\varrho(t, S) = (L_t \xleftarrow{l} K_t \xrightarrow{r} R_t), \text{ with}$$

- the common fixed net structure $PN_t = (P_t, \{t\}, pre_t, post_t)$, where $P_t = ENV(t)$, $pre_t(t) = pre(t)$ and $post_t(t) = post(t)$,
- $L_t = (PN_t, M, m_t : M \to P_t)$, with $m_t(x) = m(x)$,
- $K_t = (PN_t, \emptyset, \emptyset : \emptyset \to P_t)$,
- $R_t = (PN_t, N, n_t : N \to P_t)$, with $n_t(x) = n(x)$,
- l, r being the obvious inclusions on the rule nets.

m_t and n_t are well-defined because t is enabled under S: The token selection condition implies that $\forall x \in M : m(x) \in PRE(t)$ and due to the construction of PN_t we have $PRE(t) \subseteq ENV(t) = P_t$. The argument for n_t works analogously.

Note that t is enabled under S in L_t.

Example. Figure 27 shows a PTI transition rule $\varrho(t, S) = (L \xleftarrow{l} K \xrightarrow{r} R)$ for transition t in NI.

Definition 3.13 (Canonical DPO Transformation of PTI Nets)**.**
We call a direct transformation $NI_1 \overset{\varrho, f}{\Longrightarrow} NI_2$ by rule $\varrho = (L \xleftarrow{l} K \xrightarrow{r} R)$ with l, r being inclusions *canonical* if

- f_I is injective,
- the morphisms in the span $(NI_1 \leftarrow NI_0 \rightarrow NI_2)$ of the DPO transformation diagram below are inclusions, and
- $I_2 = I_0 \cup (I_R \setminus r(I_K))$.

$$\begin{array}{ccccc} L & \xleftarrow{l} & K & \xrightarrow{r} & R \\ {\scriptstyle f}\downarrow & (PO_1) & \downarrow & (PO_2) & \downarrow{\scriptstyle f^*} \\ NI_1 & \longleftarrow & NI_0 & \longrightarrow & NI_2 \end{array}$$

Remark. For each rule ϱ being applied to some PTI net NI at a token-injective match f, there exists a canonical transformation diagram, which is just the particular equivalent DPO diagram with "as less isomorphic changes as possible".

Because of the injectivity of the rule morphisms the lower span in the diagram is already injective as well. To construct the canonical transformation diagram we first regard the last condition on I_2, which demands that $(I_R \setminus r(I_K)) \cap I_0 = \emptyset$ because of pushout PO_2. For this, it is sufficient to replace the set of tokens in R that are created by the rule, i.e. $I_R \setminus r(I_K)$, such that is disjoint to the tokens preserved by the rule, i.e. $(I_1 \setminus f(I_L)) \cup f(l(I_K))$.[4] With this we now can simply replace arbitrary NI_0 and NI_2, being some pushout complement object of PO_1 and pushout object of PO_2, with nets such that the lower span morphisms become inclusions.

Example. The diagram in Fig. 27 shows the two pushouts in **PTINets** resulting from applying the PTI transition rule $\varrho(t, S) = (L \xleftarrow{l} K \xrightarrow{r} R)$ to the net NI with identical token morphism component. Moreover, this transformation is canonical.

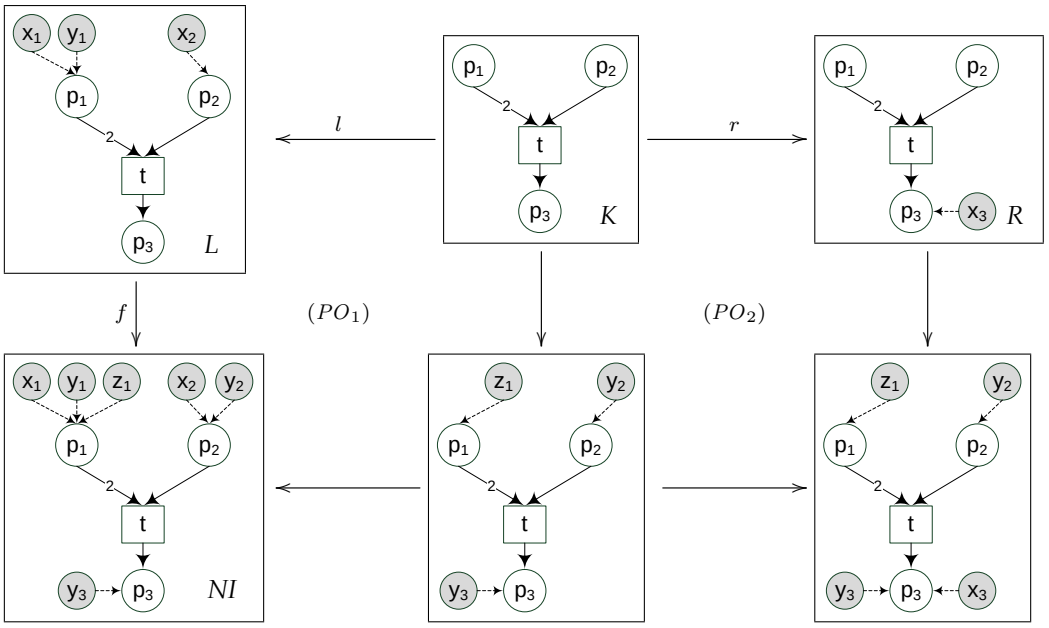


Figure 27: Canonical direct DPO-transformation in **PTINets** simulating a firing step

[4] This modification of the rule is passable since changing the tokens in R does not affect the firing behaviour of NI_2 (up to the token selections) nor the applicability of the rule.

Theorem 3.1 (Equivalence of Canonical Transformations and Firing of PTI Nets)**.**

1. Each firing step $NI \overset{t}{\rightarrowtail} NI'$ *via a token selection* $S = (M, m, N, n)$ *corresponds to a canonical direct transformation* $NI \overset{\varrho(t,S),f}{\Longrightarrow} NI'$ *via the transition rule* $\varrho(t, S)$*, matched by the inclusion match* $f : L_{\varrho(t,S)} \to NI$*.*

2. Each canonical direct transformation $NI \overset{\varrho(t,S),f}{\Longrightarrow} NI_1$*, via some transition rule* $\varrho(t, S) = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ *with* $t \in T_{NI}$ *and token selection* $S = (M, m, N, n)$*, and a token-injective match* $f : L \to NI$*, implies that the transition* $f_T(t)$ *is enabled in* NI *under* $\left(f_I(M), \left(f_P \circ m \circ f_I^{-1}\right), N, \left(f_P^* \circ m_1 \circ f_I^{*-1}\right)\right)$ *with firing step* $NI \overset{f_T(t)}{\rightarrowtail} NI_1$*.*

Proof.

1. Given the firing step $NI \overset{t}{\rightarrowtail} NI'$ via $S = (M, m, N, n)$, the canonical direct transformation of the transition rule $\varrho(t, S) = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ is $NI \overset{\varrho(t,S),f}{\Longrightarrow} NI_1$ as in the DPO diagram in Fig. 28. The match f, d, and d' are inclusions.

$$\begin{array}{ccccc}
L = (PN_t, M, m_t) & \overset{l}{\hookleftarrow} & K = (PN_t, \emptyset, \emptyset) & \overset{r}{\hookrightarrow} & R = (PN_t, N, n_t) \\
\downarrow{\scriptstyle f} & (PO_1) & \downarrow & (PO_2) & \downarrow{\scriptstyle f^*} \\
NI = (PN, I, m) & \underset{d}{\hookleftarrow} & NI_0 = (PN, I_0, m_0) & \underset{d'}{\hookrightarrow} & NI_1 = (PN, I_1, m_1)
\end{array}$$

Figure 28: DPO diagram for canonical direct transformation of NI with $\varrho(t, S)$ in **PTINets**

According to Fact 3.4 on page 30, we have $I_0 = I \setminus M$ and $I_1 = I_0 \cup (N \setminus \emptyset)$ as in the DPO diagram of the **Sets** components in Fig. 29. The firing step condition $(I \setminus M) \cap N = \emptyset$ grants us the last condition for canonical transformations $I_0 \cup (N \setminus r(\emptyset)) = I_1$.

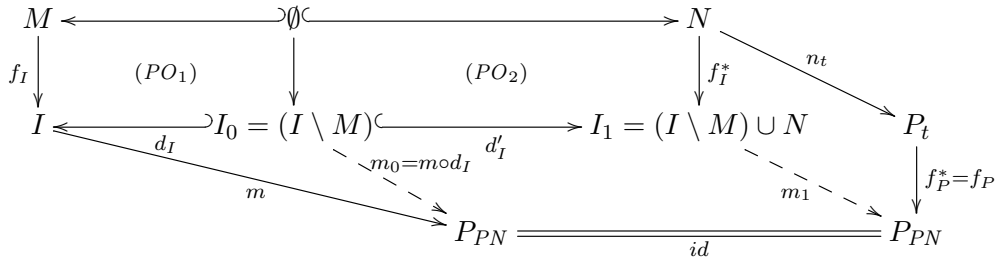


Figure 29: DPO diagram in **Sets** for the token components of the transformation in Fig. 28

For m_1 as induced morphism for the pushout object I_1 follows that

$$m_1(x) = \begin{cases} m_0(x) = m(x) \text{ for } x \in I \setminus M \\ n_t(x) = n(x) \text{ for } x \in N \end{cases}$$

hence $I_1 = I', m_1 = m'$ and $NI_1 = NI'$, according to Def. 3.2 on page 28.

Remark. f_I being injective is not only sufficient for the existence of (PO_1), but also necessary, because $I_K = \emptyset$ (see Fact 3.11 on page 34).

2. Given a canonical direct transformation $NI \overset{\varrho(t,S),f}{\Longrightarrow} NI_1$ as in the DPO diagrams in Figs. 28 on the preceding page and 29 on the previous page, $f_T(t) \in T_{NI}$ is enabled under

$$\left(f_I(M), \left(f_P \circ m \circ f_I^{-1}\right), N, \left(f_P^* \circ m_1 \circ f_I^{*-1}\right)\right)$$

if

1. $f_I(M) \subseteq I$,
2. $\left(f_P^* \circ m_1 \circ f_I^{*-1}\right) : N \to P_{PN}$,
3. $(I \setminus f_I(M)) \cap N = \emptyset$,
4. $\sum\limits_{i \in f_I(M)} \left(f_P \circ m \circ f_I^{-1}(i)\right) = pre_{NI}(f_T(t))$
5. $\sum\limits_{i \in N} \left(f_P^* \circ m_1 \circ f_I^{*-1}(i)\right) = post_{NI}(f_T(t))$
6. leading to the follower marking $I' = (I \setminus M) \cup N$
   with $m'(x) = \begin{cases} m_0(x) = m(x) \text{ for } x \in I \setminus M \\ m_1(x) = n(x) \text{ for } x \in N \end{cases}$

By construction of the transition rule $\varrho(t, S)$ and its application to NI we have 1., 2., and for 6. that $I' = I_1, m' = m_1$. The canonical transformation property grants us 3. It remains to show 4. and 5., i.e. that $f_I(M)$ and N represent the correct numbers of tokens in the environment of $f_T(t)$ to enable it:

$$\begin{aligned}
&\sum_{i \in f_I(M)} f_P \circ m \circ f_I^{-1}(i) \\
&= \sum_{i \in M} f_P \circ m(i) && (f_I \text{ inj.}) \\
&= f_P^{\oplus} \sum_{i \in M} m_t(i) && (\forall i \in M : m_t(i) = m(i), \text{due to def. } \varrho(t,S)) \\
&= f_P^{\oplus} \circ pre_{PN_t}(t) && (t \text{ enabled in } L) \\
&= pre_{NI} \circ f_T(t) && (f \text{ PTI morph.})
\end{aligned}$$

and analogously,

$$\begin{aligned}
&\sum_{i\in N} f_P^* \circ m_1 \circ f_I^{*-1}(i) \\
&= \sum_{i\in N} f_P^* \circ m_1(i) && (f_I^*(N) = N) \\
&= f_P^{*\oplus} \sum_{i\in N} n_t(i) && (\forall i \in N : n_t(i) = m_1(i), \text{due to constr. } m_1) \\
&= f_P^{*\oplus} \circ pre_{PN_t}(t) && (t \text{ enabled in } L) \\
&= post_{NI} \circ f_T(t) && (f \text{ PTI morph.})
\end{aligned}$$

□

The second item of the previous theorem covers all possible canonical direct transformations by any transition rule $\varrho(t, S)$ in an arbitrary net NI, without aasuming that $t \in NI$ or $M \subseteq I_{NI}$. For the special case that a transition rule $\varrho(t, S)$ is applied on an inclusion match, the second item reduces to the following corollary, which is more similar to the theorem's first item.

Corollary 3.2 (Equivalence of Canonical Transformations and Firing of PTI Nets)**.**
Given a canonical transformation $NI \overset{\varrho(t,S),f}{\Longrightarrow} NI_1$ such that the match $f : L \to NI$ is an inclusion, then t is enabled in NI under S with firing step $NI \overset{t}{\rightarrowtail} NI_1$.

This follows directly from 2. of Theorem 3.1.

Theorem 3.3 (Token-injective PTI Net Morphisms preserve Firing Steps)**.**
For each PTI net morphism $f : NI_1 \to NI_2$ with injective f_I component and each firing step $NI_1 \overset{t}{\rightarrowtail} NI_1'$ there exists a firing step $NI_2 \overset{f_T(t)}{\rightarrowtail} NI_2'$ and a PTI net morphism $f' : NI_1' \to NI_2'$ (depicted in diagram (1) below).

$$\begin{array}{ccc}
NI_1 & \overset{t}{\rightarrowtail} & NI_1' \\
{\scriptstyle f}\downarrow & (1) & \downarrow{\scriptstyle f'} \\
NI_2 & \underset{f_T(t)}{\rightarrowtail} & NI_2'
\end{array}$$

Proof. Given $f : NI_1 \to NI_2$ and $NI_1 \overset{t}{\rightarrowtail} NI_1'$ via some S as above, we have by the first part of Theorem 3.1 on page 37 the canonical direct transformation given by the pushouts (1) and (2) with $\varrho(t, S) = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ in Fig. 30 on the next page.

Note that $f : NI_1 \to NI_2$ satisfies the gluing condition w.r.t this rule and f, because $l'_P = id_{P_1}, l'_T = id_{T_1}$[5] and none of the individuals of NI_1 is an identification point ($IP_I = \emptyset$) due to injectivity of f_I. This allows to construct the canonical transformation with extended rule $NI_1 \overset{l'}{\leftarrow} NI_{D1} \overset{r'}{\rightarrow} NI_1'$ along pushouts (3) and (4). Hence, also $(1 + 3)$ and $(2 + 4)$ are pushouts of a canonical direct transformation via $\varrho(t, S)$. The second part of Theorem 3.1 on page 37 implies $NI_2 \overset{f_T(t)}{\rightarrowtail} NI_2'$. □

[5] This means that all places and transitions of NI_1 are gluing points.

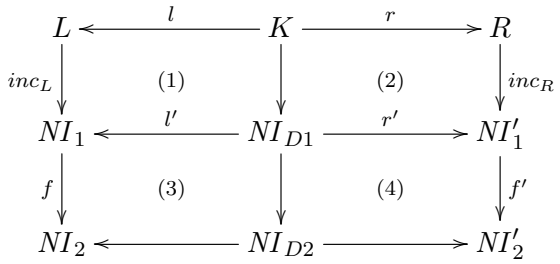


Figure 30: DPO diagrams for canonical direct transformation of NI_1 and NI_2 with $\varrho(t, S)$

Remark (Firing-preserving diagrams for t correspond to an extension diagram for $\varrho(t, S)$). The diagram in Fig. 30 corresponds to an extension diagram for rule $\varrho(t, S)$ and f (as depicted below) because $(1) - (4)$ are pushouts.

$$\begin{array}{ccc} NI_1 & \overset{\varrho(t,S),inc}{\Longrightarrow} & NI'_1 \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f'} \\ NI_2 & \overset{\varrho(t,S),f\circ inc}{\Longrightarrow} & NI'_2 \end{array}$$

4 Algebraic High-Level Nets with Individual Tokens and Algebraic Higher-Order Nets with Individual Tokens

In this section we review the the definition of Algebraic High-Level Nets with Individual Tokens [MGE+10] in order to specify and define the Algebraic Higher-Order Nets with individual Tokens (short $AHOI_{PTI}$ nets) by giving the definition of such $AHOI_{PTI}$ nets and the definition of their signature and algebra

4.1 Algebraic High-Level Nets with Individual Tokens

We lift the results of the previous section to high-level nets whose tokens represent values of an algebra to a signature [EM85]. The rule-based transformation of *collective* algebraic high-level nets from [PER95] is the foundation for the approach presented in the following.

4.1.1 Definition and Firing Behaviour

Definition 4.1 (Algebraic High-Level Nets with Individual Tokens)**.**
We define a marked AHL net with individual tokens, short AHLI net, as

$$ANI = (\Sigma, P, T, pre, post, cond, type, A, I, m),$$

where

- $AN = (\Sigma, P, T, pre, post, cond, type, A)$ is a classical AHL net with

  - signature $\Sigma = (S, OP, X)$ of sorts S, operation symbols OP and variables $X = (X_s)_{s \in S}$,
  - sets of places P and transitions T,
  - $pre, post : T \to (T_{OP}(X) \otimes P)^{\oplus}$[6], defining the transitions' pre- and postdomains,
  - $cond : T \to \mathcal{P}_{fin}(Eqns(S, OP, X))$ assigning a finite set of Σ-equations (L, R, X) as firing conditions to each transition,
  - $type : P \to S$ typing the places of the signature's sorts,
  - a Σ-algebra A,

- I is the (possibly infinite) set of individual tokens of ANI, and
- $m : I \to A \otimes P$ is the marking function, assigning the individual tokens to the data elements on the places. $m(I)$ defines the actual set of data elements on the places of ANI. m does not have to be injective.

Further, we introduce some additional notations:

- $Var(t) \subseteq X$ is the set of variables occurring in equations and on the environment arcs of t,
- $CP = (A \otimes P) = \{(a, p) \in A \times P | a \in A_{type(p)}\}$ as the set of consistent value/place pairs,
- $CT = \{(t, asg) \in T \times (Var(t) \to A) | \forall (L, R, X) \in cond(t) : \overline{asg}(L) = \overline{asg}(R)\}$ as the set of consistent transition assignments, i.e. all firing conditions of t are valid when evaluated with the variable assignment asg[7],
- $ENV(t) = PRE(t) \cup POST(t) \subseteq (T_{OP}(X) \otimes P)$ as the environment of a transition $t \in T$ where

$$PRE(t) = \{(term, p) \in (T_{OP}(X) \otimes P) | pre(t)(term, p) \neq 0\}$$
$$POST(t) = \{(term, p) \in (T_{OP}(X) \otimes P) | post(t)(term, p) \neq 0\}$$

- $ENV_P(t) = \pi_P(ENV(t)) \subseteq P$ the place environment of t,
- $pre_A, post_A : CT \to CP^{\oplus}$, defined by

$$pre_A(t, asg) = (\overline{asg} \otimes id_P)^{\oplus}(pre(t)),$$
$$post_A(t, asg) = (\overline{asg} \otimes id_P)^{\oplus}(post(t))$$

[6] $T_{OP}(X) \otimes P = \{(t, p) \in T_{OP}(X) \times P | t \in T_{OP, type(p)}(X)\}$, i.e the pairs where term t is of sort $type(p)$.
[7] where $\overline{asg} : T_{OP}(X) \to A$ is the evaluation of Σ-terms over variables in X to values in A. Technically, $\overline{asg} = xeval(asg)_A$ results from a free construction over asg.

Similarly, we define the sets[8]

$$PRE_A(t, asg) = \{(a,p) \in (A \otimes P) | pre_A(t, asg)(a,p) \neq 0\}$$
$$POST_A(t, asg) = \{(a,p) \in (A \otimes P) | post_A(t, asg)(a,p) \neq 0\}$$

We can express e.g. the concrete required *number* of a token (a,p) for t to fire under assignment asg with $pre_A(t, asg)(a,p)$ by interpreting the monoid $pre_A(t, asg)$ as a function $CP \to \mathbb{N}$. Similarly, we get the produced *number* of (a,p) with $post_A(t, asg)(a,p)$.

Remark (Individual tokens vs. classical algebraic data tokens). Each AHLI net with finite individual token marking (I, m) can be interpreted as an AHL net with marking

$$M = \sum_{(a,p) \in A \otimes P} |m^{-1}(a,p)|(a,p) = \sum_{i \in I} m(i)$$

where $|m^{-1}(a,p)|$ denotes the cardinality of individual tokens in I that m maps to (a,p).

In AHL nets, the tokens are of the form (a,p), s.t. they have already a kind of identity, depending on their data values. The main difference to AHLI nets is that we can distinguish tokens of the same algebraic value on the same place. Moreover, the individuals equip data tokens with identities. When firing a transition, we now can relate the input and output tokens so that a token's history can be traced along the firing steps.

Example. Figure 31 on the facing page shows the graphical representation of an AHLI net with

- signature $\Sigma = (\{s_1, s_2, s_3\},\ \{t_{11} :\to s_1, t_{12} :\to s_1, t_2 :\to s_2, t_3 :\to s_3\})$,
- algebra carrier sets $A_{s_1} = \{a_1, b_1, c_1\}, A_{s_2} = \{a_2, b_2\}, A_{s_3} = \{a_3, b_3, c_1\}$
- $pre(t) = (t_{11}, p_1) \oplus (t_{12}, p_1) \oplus (t_2, p_2), post(t) = (t_3, p_3)$,
- $type(p_1) = s_1, type(p_2) = s_2, type(p_3) = s_3$,
- $cond(t) = \emptyset$,
- $I = \{x_1, y_1, x_2, x_3\}$, $m(x_1) = m(y_1) = (a_1, p_1), m(x_2) = (a_2, p_2), m(x_3) = (a_3, p_3)$,

Note that the algebraic value of an individual token is given next to the dashed arc to its carrying place. In the following, if a transition has an empty set of conditions we just denote the transition name without an explicit $\emptyset$ below.

Similar as for low-level PTI nets, we now define firing steps for a transition and a token selection. In addition, we have to take into account assignments evaluating the variables on the arcs and in the transition conditions to algebra values.

[8] Obviously, these sets are the same as $(\overline{asg} \otimes id_P) \circ PRE(t)$ and $(\overline{asg} \otimes id_P) \circ POST(t)$, respectively.

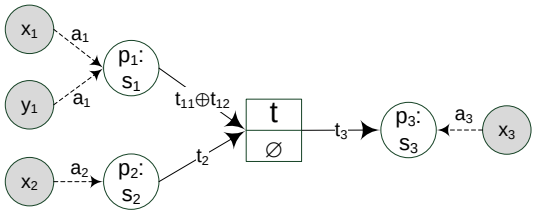


Figure 31: Example AHLI net

Definition 4.2 (Firing of AHLI nets)**.**
A consistent transition assignment $(t, asg) \in CT$ for an AHLI net

$$ANI = (\Sigma, P, T, pre, post, cond, type, A, I, m)$$

is *enabled* under a *token selection* (M, m, N, n), where

- $M \subseteq I$,
- m is the token mapping of ANI,
- N is a set with $(I \setminus M) \cap N = \emptyset$,
- $n : N \to A \otimes P$ is a function,

if it meets the following *token selection condition*:

$$\sum_{i \in M} m(i) = pre_A(t, asg) \wedge \sum_{i \in N} n(i) = post_A(t, asg)$$

If such an asg-enabled t fires, the follower marking (I', m') is given by

$$I' = (I \setminus M) \cup N, \quad m' : I' \to A \otimes P \text{ with } m'(x) = \begin{cases} m(x), & x \in I \setminus M \\ n(x), & x \in N \end{cases}$$

leading to $ANI' = (AN, I', m')$ as the new AHLI net in the *firing step* $ANI \overset{t,asg}{\rightarrowtail} ANI'$ via (M, m, N, n).

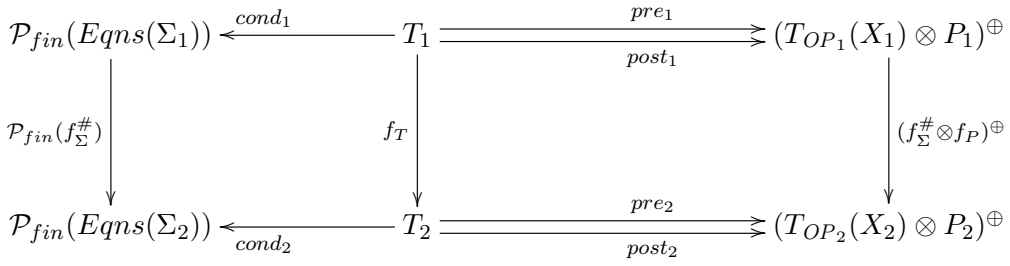
Remark (Token Selection). The purpose of the token selection is to specify exactly which tokens should be consumed and produced in the firing step. Thus, $M \subseteq I$ selects the individual tokens to be consumed, and N contains the set of individual tokens to be produced. Clearly, $(I \setminus M) \cap N = \emptyset$ must hold because it is impossible to add an individual token to a net that already contains this token. m and n relate the tokens to their place/value pairs. It would be sufficient to consider only the restriction $m_{|M}$ here or to infer it from the net but as a compromise on symmetry and readability we denote m in the token selection.

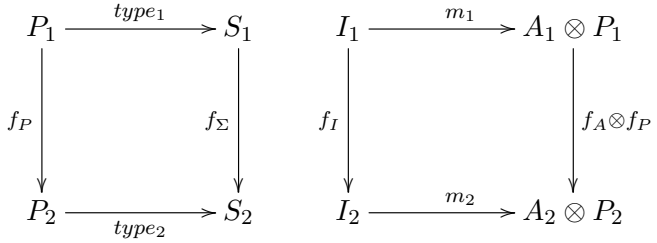
As a preparation for the transformation in the next subsection, we define a category of AHLI nets.

Definition 4.3 (AHLI Net Morphisms and Category **AHLINets**).
Given two AHLI nets $ANI_i = (\Sigma_i, P_i, T_i, pre_i, post_i, cond_i, type_i, A_i, I_i, m_i)$, $i \in \{1,2\}$, an AHLI net morphism $f : ANI_1 \to ANI_2$ is a pentuple

$$f = (f_\Sigma : \Sigma_1 \to \Sigma_2, f_P : P_1 \to P_2, f_T : T_1 \to T_2, f_A : A_1 \to V_{f_\Sigma}(A_2), f_I : I_1 \to I_2)$$

such that the following diagrams commute (componentwise for pre_i and $post_i$)[9]:





For a transition assignment $(t, asg : X_1 \to A_1)$ we call

$$asg_f = f_A \circ asg \circ f^{-1}_{\Sigma|Var(t)} : Var(f_T(t)) \to A_2$$

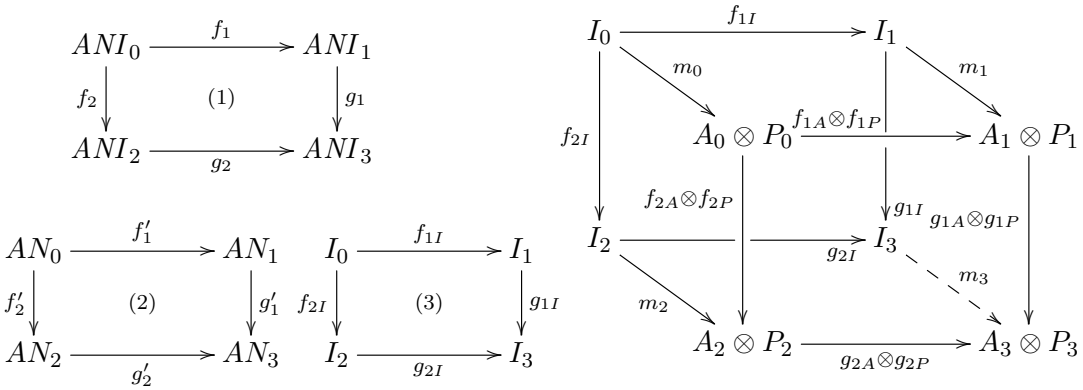
the *translation* of asg along f if f_Σ is bijective on the variables in $Var(t)$. Actually, for all transitions $t \in T_1$ all $f_{\Sigma|Var(t)} : Var(t) \to Var(f_T(t))$ are already surjective because of the commutativity of the diagrams above. So it is sufficient to demand injectivity for $f_{\Sigma|Var(t)}$ such that asg_f is well-defined.

The category **AHLINets** consists of all AHLI nets as objects and all AHLI net morphisms.

Remark (Generalized algebra morphisms). Because $V_{f_\Sigma}(A_2)$ just forgets some carrier sets of A_2 if we considered them as a family of sets, we may use f_A also with A_2 as codomain omitting the postponed family of identities $(\iota_s : (V_{f_\Sigma}(A_2))_s \to A_{2,f_\Sigma(s)})_{s \in S}$. Moreover, in the following constructions the algebra parts of the pushout cospan (and the pullback span resp.) result from general constructions in the Grothendieck construct with objects $(\Sigma, A \in \mathbf{Alg}(\mathbf{\Sigma}))$ and generalized algebra morphisms. See [EBO92, TBG91] for amalgamation of algebras and limits/colimits in Grothendieck constructs and also [EM85, EOP06] for details on the usage of free functors.

[9] V_{f_Σ} is the forgetful functor induced by signature homomorphism f_Σ, such that $f_A : A_1 \to V_{f_\Sigma}(A_2)$ is a generalized Σ_1-homomorphism. $f_\Sigma^\#$ is the extension of f_Σ to terms and equations.

Fact 4.4 (Construction of Pushouts in **AHLINets**).
Pushouts in **AHLINets** are constructed componentwise in **AHLNets** and **Sets**. So, (1) is a PO in **AHLINets** iff (2) is a PO in **AHLNets** and (3) is a PO in **Sets** with the components of the **AHLINets** morphisms, where $m_3 : I_3 \to A_3 \otimes P_3$ is induced by PO object I_3 in the commutating cube below (whose front's place components let the front commutate because of the PO in the net structure).



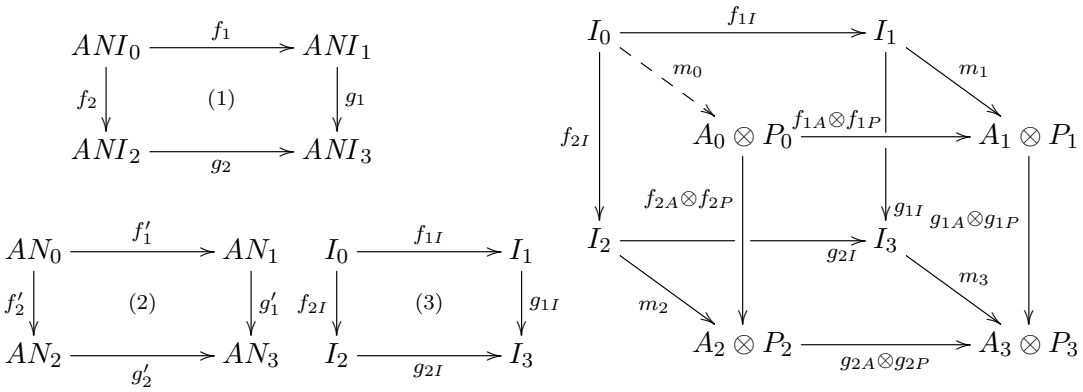
If f_{1X}, for $X \in \{P, T, I\}$, is injective then g_{2X} is as well and similar for components of f_2 and g_1 and the other diagrams.

For the construction of pushouts in **AHLNets** we refer to [PER95].

Example. Figure 32 on page 52 shows two pushouts in **AHLINets**.

Fact 4.5 (Construction of Pullbacks in **AHLINets**).
Pullbacks in **AHLINets** along injective **AHLNets** morphisms[10] are constructed componentwise in **AHLNets** and **Sets**.

Consider a commutative square (1) in **AHLINets** with g'_1 being injective. (1) is a PB in **AHLINets** iff (2) is a PB in **AHLNets** and (3) is a PB in **Sets** with the components of the **AHLINets** morphisms, where $m_0 : I_0 \to A \otimes P_0$ is induced by PB object P_0 in the commutating cube below (whose front's place components let the front commutate because of the PB in the net structure).



[10] We require morphisms that are injective on the net structure in order to obtain componentwise pullbacks in **AHLNets**. See [EEPT06] for details.

Example. The pushouts in Fig. 32 on page 52 are pullbacks, too.

4.2 Transformation of AHLI Nets

This section is about rule-based transformation of AHLI nets. For this, we use the double pushout approach, which has also been used for *collective* AHL nets in [PER95] and which has been investigated in the context of $\mathcal{M}$-adhesive systems in [EEPT06]. We are going to characterize the applicability of rules at some match by initial pushouts.

Definition 4.6 (AHLI Transformation Rules)**.**
An AHLI transformation rule is a span of injective **AHLINets** morphisms

$$\varrho = (L \xleftarrow{l} K \xrightarrow{r} R).$$

Remark. AHLI Rule morphisms are not required to be marking-strict in order to obtain an $\mathcal{M}$-adhesive category (see Sect. 5 in [MGE+10]). This allows to arbitrarily change the marking of an AHLI net by applying rules with corresponding places and individual tokens in L and R.

Definition 4.7 (AHLI Transformation)**.**
Given an AHLI transformation rule $\varrho = (L \xleftarrow{l} K \xrightarrow{r} R)$ and an AHLI net ANI_1 with a AHLI net morphism $m : L \to ANI_1$, called the match, a direct AHLI net transformation $ANI_1 \overset{\varrho,m}{\Longrightarrow} ANI_2$ from ANI_1 to the AHLI net ANI_2 is given by the following double-pushout diagram (DPO) diagram in the category **AHLINets**:

$$\begin{array}{ccccc} L & \xleftarrow{\quad l \quad} & K & \xrightarrow{\quad r \quad} & R \\ {\scriptstyle m}\downarrow & (PO_1) & \downarrow & (PO_2) & \downarrow{\scriptstyle m^*} \\ ANI_1 & \longleftarrow & ANI_0 & \longrightarrow & ANI_2 \end{array}$$

To be able to decide whether a rule is applicable at a certain match, we formulate a gluing condition for AHLI nets, such that there exists a pushout complement of the left rule morphisms and the match if (and only if) they fulfill the gluing condition. The correctness of the gluing condition is shown via a proof on initial pushouts over matches, according to [EEPT06].

Definition 4.8 (Gluing Condition in **AHLINets**)**.**
Given AHLI nets K, L, and ANI and AHLI morphisms $l : K \to L$ and $f : L \to ANI$. We define the set of identification points[11]

$$IP = IP_P \cup IP_T \cup IP_I$$

with

- $IP_P = \{x \in P_L \mid \exists x' \neq x : f_P(x) = f_P(x')\}$,

[11] That is, all elements in L that are mapped non-injectively by f.

- $IP_T = \{x \in T_L \mid \exists x' \neq x : f_T(x) = f_T(x')\}$,
- $IP_I = \{x \in I_L \mid \exists x' \neq x : f_I(x) = f_I(x')\}$,

the set of dangling points[12]

$$DP = DP_T \cup DP_I$$

with

- $DP_T = \{p \in P_L \mid \exists t \in T_{ANI} \setminus f_T(T_L) : f_P(p) \in ENV_P(t)\}$,
- $DP_I = \{p \in P_L \mid \exists i \in I_{ANI} \setminus f_I(I_L) : f_P(p) = \pi_P(m_{ANI}(i))\}$,

and the set of gluing points[13]

$$GP = l_P(P_K) \cup l_T(T_K) \cup l_I(I_K)$$

We say that l and f satisfy the gluing condition if $IP \cup DP \subseteq GP$. Given an AHLI rule $\varrho = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$, we say that ϱ and f satisfy the gluing condition iff l and f satisfy the gluing condition.

$$\begin{array}{ccccc} L & \xleftarrow{\quad l \quad} & K & \xrightarrow{\quad r \quad} & R \\ {\scriptstyle f}\downarrow & & & & \\ ANI & & & & \end{array}$$

In order to construct the initial pushout for a match $f : L \to ANI$, we define the *boundary* over the match f, which is the minimal subnet containing all places, transitions, and individual tokens that must not be deleted by the application of a rule with left hand side L such that there exists a pushout complement.

Definition 4.9 (Boundary in **AHLINets**).
Given two AHLI nets

$$L = (\Sigma_L, P_L, T_L, pre_L, post_L, cond_L, type_L, A_L, I_L, m_L),$$
$$ANI = (\Sigma_{ANI}, P_{ANI}, T_{ANI}, pre_{ANI}, post_{ANI}, cond_{ANI}, type_{ANI}, A_{ANI}, I_{ANI}, m_{ANI})$$

and an AHLI morphism $f : L \to ANI$. The boundary of f is an AHLI net

$$B = (\Sigma_B, P_B, T_B, pre_B, post_B, cond_B, type_B, A_B, I_B, m_B)$$

with

- $\Sigma_B = \Sigma_L$,
- $P_B = DP_T \cup DP_I \cup IP_P \cup P_{IP_T} \cup P_{IP_I}$

[12] That is, all places in L that would leave a dangling arc, if deleted.
[13] That is, all elements in L that have a preimage in K.

- $P_{IP_T} = \{p \in P_L \mid \exists t \in IP_T : p \in ENV_P(t)\}$
- $P_{IP_I} = \{p \in P_L \mid \exists i \in IP_I : p = \pi_P(m_L(i))\}$
- $T_B = IP_T$
- $pre_B(t) = pre_L(t)$
- $post_B(t) = post_L(t)$
- $cond_B(t) = cond_L(t)$
- $type_B(p) = type_L(p)$
- $A_B = A_L$
- $I_B = IP_I$
- $m_B(i) = m_L(i)$

together with $b : B \to L = (b_\Sigma, b_P, b_T, b_A, b_I)$ where $b_\Sigma = id_{\Sigma_B}$, $b_A = id_{A_B}$, and the remaining parts are inclusions.

Well-definedness.

$pre_B, post_B : T_B \to (T_{OP_B}(X_B) \otimes P_B)^\oplus$:
Let $t \in T_B$ and let $(term, p) \leq pre_B(t)$. Then there is $(term, p) \leq pre_L(t)$ which means that $term \in T_{OP_L}(X_L)$ and $p \in P_L$. Then from $\Sigma_B = \Sigma_L$ follows that $term \in T_{OP_B}(X_B)$. Furthermore there is $t \in IP_T$ which by the fact that $p \in ENV_P(t)$ means that $p \in P_{IP_T} \subseteq P_B$.
So pre_B is well-defined. The proof for $post_B$ works completely analogously.

$cond_B : T_B \to \mathcal{P}_{fin}(Eqns(\Sigma_B))$:
Let $t \in T_B$. Then we have

$$cond_B(t) = cond_L(t) \in \mathcal{P}_{fin}(Eqns(\Sigma_L)) = \mathcal{P}_{fin}(Eqns(\Sigma_B))$$

$type_B : P_B \to S_B$:
Let $p \in P_B$. Then we have

$$type_B(p) = type_L(p) \in S_L = S_B$$

$m_B : I_B \to A_B \otimes P_B$:
Let $i \in I_B$ and let $(a, p) = m_B(i)$. Then there is $(a, p) = m_L(i)$ which means that $a \in A_{type_L(p)} = A_{type_B(p)}$. The fact that $p \in P_B$ follows from the fact that $i \in IP_I$ and hence $p \in P_{IP_I} \subseteq P_B$.

inclusion $b : B \to L$:
We obtain an inclusion morphism $b : B \to L$ from the fact that pre_B,$post_B$,$cond_B$,$type_B$, and m_B are restrictions of the respective functions in L.

□

The following facts about the gluing condition and initial pushouts hold in all $\mathcal{M}$-adhesive categories (**AHLINets**, $\mathcal{M}$) whose morphism class $\mathcal{M}$ of monomorphisms contains at least inclusions with identities for signature and algebra parts (for concrete instantiations see Sect. 5 in [MGE+10]). The next fact completes the construction of initial pushouts for matches and shows that the construction of the boundary in Def. 4.9 on page 47 complies to the categorical notion of boundaries in Def. 3.9 in [MGE+10].

Fact 4.10 (Initial Pushout in **AHLINets**).
Given a morphism $f : L \to ANI$ in **AHLINets**, the boundary B of f and the AHLI net

$$C = (\Sigma_C, P_C, T_C, pre_C, post_C, cond_C, type_C, A_C, I_C, m_C)$$

with

- $\Sigma_C = \Sigma_{ANI}$
- $P_C = (P_{ANI} \setminus f_P(P_L)) \cup f_P(b_P(P_B))$
- $T_C = (T_{ANI} \setminus f_T(T_L)) \cup f_T(b_T(T_B))$
- $pre_C(t) = pre_{ANI}(t)$
- $post_C(t) = post_{ANI}(t)$
- $cond_C(t) = cond_{ANI}(t)$
- $type_C(p) = type_{ANI}(p)$
- $A_C = A_{ANI}$
- $I_C = (I_{ANI} \setminus f_I(I_L)) \cup f_I(b_I(I_B))$
- $m_C(i) = m_{ANI}(i)$

Then diagram (1) is an initial pushout in **AHLINets**, where $g := f|_B$ and c is an inclusion with $c_\Sigma = id_{\Sigma_C}$ and $c_A = id_{A_C}$.

$$\begin{array}{ccc} B & \xrightarrow{b} & L \\ {\scriptstyle g}\downarrow & (1) & \downarrow{\scriptstyle f} \\ C & \xrightarrow[c]{} & ANI \end{array}$$

Proof. see Sect. B.3 in [MGE+10]. □

The following two facts show the correspondence between the gluing condition in **AHLINets** and the categorical gluing condition (see Def. A.2 in [MGE+10]), which is a necessary and sufficient condition for the (unique) existence of pushout complements in $\mathcal{M}$-adhesive categories.

Fact 4.11 (Characterization of Gluing Condition in **AHLINets**).
Let $l : K \to L$ and $f : L \to ANI$ be morphisms in **AHLINets** with $l \in \mathcal{M}$.
The morphisms l and f satisfy the gluing condition in **AHLINets** if and only if they satisfy the categorical gluing condition.

Proof. see Section B.4 in [MGE+10]. □

Fact 4.12 (Gluing Condition for AHLI Transformation).
Given an AHLI rule $\varrho = (L \xleftarrow{l} K \xrightarrow{r} R)$ and a match $f : L \to ANI$ into an AHLI net $ANI = (AN, I, m : I \to P_{AN})$. The rule ϱ is applicable on match f, i.e. there exists a (unique) pushout complement ANI_0 in the diagram below, iff ϱ and f satisfy the gluing condition in **AHLINets**.

$$\begin{array}{ccccc} L & \xleftarrow{\quad l \quad} & K & \xrightarrow{\quad r \quad} & R \\ {\scriptstyle f}\downarrow & (PO) & \downarrow{\scriptstyle f'} & & \\ ANI & \nleftarrow & ANI_0 & & \end{array}$$

Proof. By Fact 4.11 the rule ϱ and match f satisfy the gluing condition in **AHLINets** if and only if ϱ and f satisfy the categorical gluing condition, which by Fact A.3 in [MGE+10] is a sufficient and necessary condition for the existence of a (unique) pushout complement ANI_0. □

4.3 Correspondence of Transition Firing and Rules

Now that we can manipulate a net's marking with rules, we have a look at rules that simulate firing steps for a certain transition under a token selection. We show that firing of a transition corresponds to a canonical application of a special rule construction, called transition rules. With this correspondence we can easily show that token-injective AHLI morphisms preserve firing steps.

Definition 4.13 (AHLI Transition Rules).
Given an AHLI net

$$ANI = (\Sigma, P, T, pre, post, cond, type, A, I, m)$$

we define the *transition rule* for a consistent transition assignment $(t, asg) \in CT_{ANI}$, enabled under the token selection $S = (M, m, N, n)$, as the rule

$$\varrho(t, S, asg) = (L_t \xleftarrow{l} K_t \xrightarrow{r} R_t)$$

with

- the common fixed AHL net part $AN_t = (\Sigma, P_t, \{t\}, pre_t, post_t, type_t, A)$, where $P_t = ENV_P(t), pre_t(t) = pre(t), post_t(t) = post(t), type_t(p) = type(p)$,
- $L_t = (AN_t, M, m_t : M \to A \otimes P_t)$, with $m_t(x) = m(x)$,

- $K_t = (AN_t, \emptyset,\ \emptyset : \emptyset \to (A \times P_t))$,
- $R_t = (AN_t, N, n_t : N \to A \otimes P_t)$, with $n_t(x) = n(x)$,
- l, r being the obvious inclusions on the rule nets.

m_t and n_t are well-defined because t is enabled under S: The token selection condition implies that $\forall x \in M : m(x) \in PRE_A(t, asg)$ and due to the construction of AN_t we have $PRE_A(t, asg) \subseteq (A \otimes ENV_P(t)) = (A \otimes P_t)$. The argument for n_t works analogously.

Note that (t, asg) is enabled under S in L_t.

Remark. The structure of a transition rule depends only on the transition and the token selection, for which there may exist several enabled transition assignments. Therefore different consistent transition assignments may have the same correspondent transition rule. Nevertheless, we denote an AHLI transition rule as $\varrho(t, S, asg)$ rather than $\varrho(t, S)$ to remember the concrete assignment this rule is intended to simulate.

Example. Figure 32 shows an AHLI transition rule $\varrho(t, S, asg) = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$.

Definition 4.14 (Canonical DPO Transformation of AHLI Nets)**.**
We call a direct transformation $ANI_1 \overset{\varrho, f}{\Longrightarrow} ANI_2$ by rule $\varrho = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ with l, r being inclusions *canonical* if

- f_I is injective,
- f_Σ is injective on the set of variables of Σ_{ANI_1}[14]
- the morphisms in the span $(ANI_1 \leftarrow ANI_0 \rightarrow ANI_2)$ of the DPO transformation diagram below are inclusions, and
- $I_2 = I_0 \cup (I_R \setminus r(I_K))$.

$$\begin{array}{ccccc} L & \xleftarrow{\quad l \quad} & K & \xrightarrow{\quad r \quad} & R \\ {\scriptstyle f}\downarrow & (PO_1) & \downarrow & (PO_2) & \downarrow{\scriptstyle f^*} \\ ANI_1 & \longleftarrow & ANI_0 & \longrightarrow & ANI_2 \end{array}$$

Remark. For each rule ϱ being applied to some AHLI net ANI at a token-injective match f, there exists a canonical transformation diagram, which is just the particular equivalent DPO diagram with "as less isomorphic changes as possible".

Because of the injectivity of the rule morphisms the lower span in the diagram is already injective as well. To construct the canonical transformation diagram we first regard the last condition on I_2, which demands that $(I_R \setminus r(I_K)) \cap I_0 = \emptyset$ because of pushout PO_2. For this, it is sufficient to replace the set of tokens in R that are created by the rule, i.e. $I_R \setminus r(I_K)$, such that it is disjoint to the tokens preserved by the rule,

[14] See the "Context Condition" in [EP97].

i.e. $(I_1 \setminus f(I_L)) \cup f(l(I_K))$.[15] With this we now can simply replace arbitrary ANI_0 and ANI_2, being some pushout complement object of PO_1 and pushout object of PO_2, with nets such that the lower span morphisms become inclusions.

Example. The following diagram shows the two pushouts in **AHLINets** resulting from applying the AHL transition rule $\varrho(t, S, asg) = (L \xleftarrow{l} K \xrightarrow{r} R)$ for

$$asg = \{t_{11} \mapsto a_1, t_{12} \mapsto b_1, t_2 \mapsto a_2, t_3 \mapsto a_3\}$$

to the net ANI. All nets in this diagram have the same signature and algebra as the example net in Fig. 31 on page 43. Moreover, this transformation is canonical.

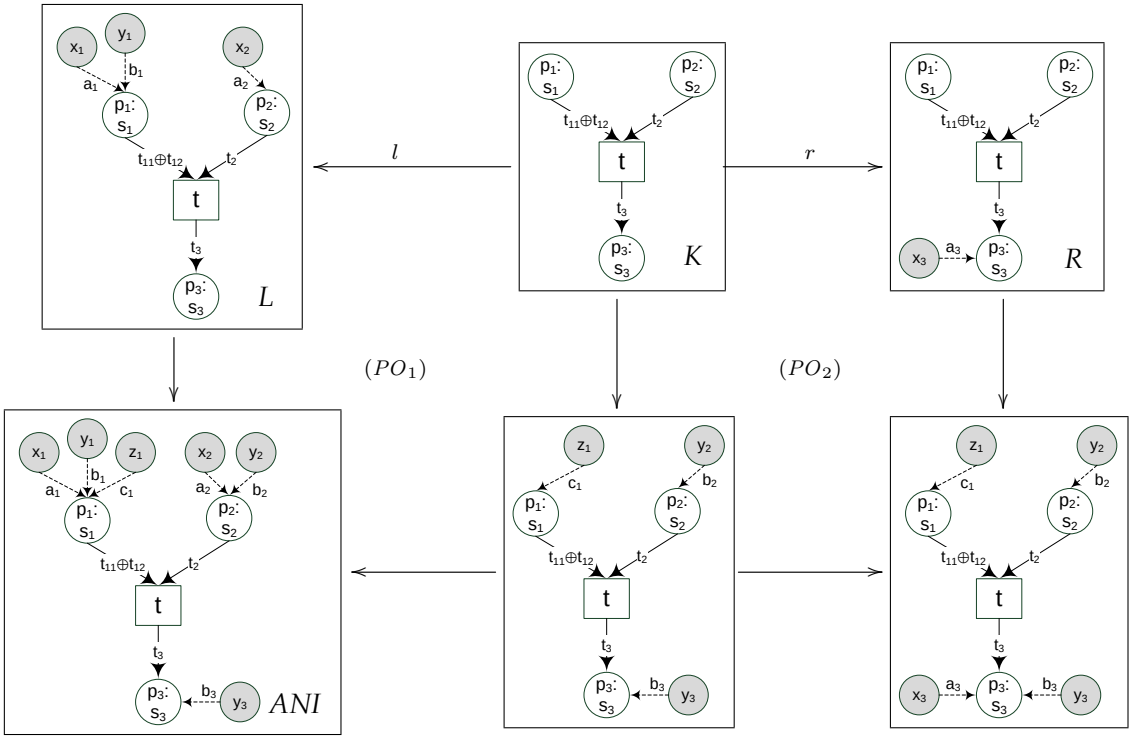


Figure 32: Canonical direct DPO-transformation in **AHLINets** simulating a firing step

Theorem 4.1 (Equivalence of Canonical Transformations and Firing of AHLI Nets)**.**

1. Each firing step $ANI \overset{t,asg}{\longmapsto} ANI'$ *via token selection* $S = (M, m, N, n)$ *corresponds to a canonical direct transformation* $ANI \overset{\varrho(t,S,asg),f}{\Longrightarrow} ANI'$ *via the transition rule* $\varrho(t, S, asg)$ *matched by the inclusion match* $f : L \to ANI$.

[15] This modification of the rule is passable since changing the tokens in R does not affect the firing behaviour of NI_2 (up to the token selections) nor the applicability of the rule.

2. Each canonical direct transformation $ANI \overset{\varrho(t,S,asg),f}{\Longrightarrow} ANI_1$*, via some transition rule* $\varrho(t,S,asg) = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ *with* $t \in T_{ANI}$ *and token selection* $S = (M,m,N,n)$*, and token-injective match* $f : L \to ANI$*, implies the consistent transition assignment* $(f_T(t), asg_f)$ *being enabled in ANI under*

$$\left(f_I(M), \left((f_A \otimes f_P) \circ m \circ f_I^{-1}\right), N, \left((f_A^* \otimes f_P^*) \circ m_1 \circ f_I^{*-1}\right)\right)$$

with firing step $ANI \overset{f_T(t),asg_f}{\rightarrowtail} ANI_1$*. With* asg_f *we denote the translation of asg along* f*, i.e.* $asg_f = f_A \circ asg \circ f^{-1}_{\Sigma|Var(t)}$*.*

Proof.

1. Given the firing step $ANI \overset{t,asg}{\rightarrowtail} ANI'$ via $(M,m,N,n) = S$, the canonical direct transformation of the transition rule $\varrho(t,S,asg)$ is $ANI \overset{\varrho(t,asg),f}{\Longrightarrow} ANI_1$ as in the DPO diagram in Fig. 33. The match f, d, and d' are inclusions.

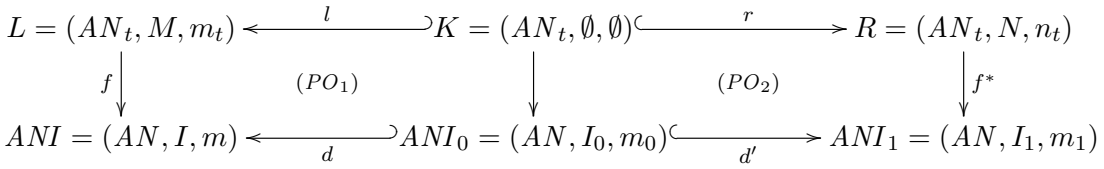


Figure 33: DPO diagram for canonical direct transformation of ANI with $\varrho(t,S,asg)$ in **AHLINets**

According to Fact 4.4 on page 45, we have $I_0 = I \setminus M$ and $I_1 = I_0 \cup (N \setminus \emptyset)$ as in the DPO diagram of the **Sets** components in Fig. 34. The firing step condition $(I \setminus M) \cap N = \emptyset$ grants us the last condition for canonical transformations $I_0 \cup (N \setminus r(\emptyset)) = I_1$.

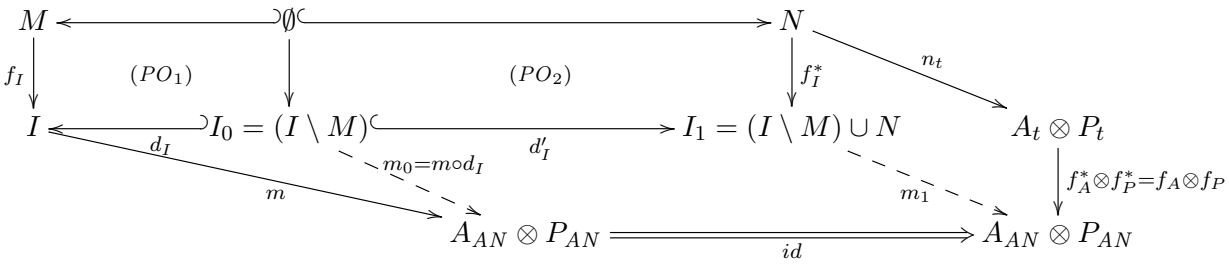


Figure 34: DPO diagram in **Sets** for the token components of the transformation in Fig. 33

For m_1 as induced morphism for the pushout object I_1 follows that

$$m_1(x) = \begin{cases} m_0(x) = m(x) \text{ for } x \in I \setminus M \\ n_t(x) = n(x) \text{ for } x \in N \end{cases}$$

hence $I_1 = I', m_1 = m'$ and $ANI_1 = ANI'$, according to Def. 4.2 on page 43.

Remark. f_I being injective is not only sufficient for the existence of (PO_1), but also necessary, because $I_K = \emptyset$ (see Fact 4.12 on page 50).

2. Consider a canonical direct transformation $ANI \overset{\varrho(t,S,asg),f}{\Longrightarrow} ANI_1$ with

$$ANI = (\Sigma, P, T, pre, post, cond, type, A, I, m)$$

as in the DPO diagrams in Figs. 33 and 34 on the preceding page. $(f_T(t), asg_f) \in CT_{ANI}$ is enabled under

$$\left(f_I(M), \left((f_A \otimes f_P) \circ m \circ f_I^{-1}\right), N, \left((f_A^* \otimes f_P^*) \circ m_1 \circ f_I^{*-1}\right)\right)$$

if

1. $f_I(M) \subseteq I$,
2. $(f_A \otimes f_P) \circ m \circ f_I^{-1} : N \to A \otimes P$,
3. $(I \setminus f_I(M)) \cap N = \emptyset$,
4. $\sum\limits_{i \in f_I(M)} \left((f_A \otimes f_P) \circ m \circ f_I^{-1}(i)\right) = pre_A(f_T(t), asg_f)$
5. $\sum\limits_{i \in N} \left((f_A^* \otimes f_P^*) \circ m_1 \circ f_I^{*-1}(i)\right) = post_A(f_T(t), asg_f)$
6. leading to the follower marking $I' = (I \setminus M) \cup N$
   with $m'(x) = \begin{cases} m_0(x) = m(x) \text{ for } x \in I \setminus M \\ m_1(x) = n(x) \text{ for } x \in N \end{cases}$

By construction of the transition rule $\varrho(t, S, asg)$ and its application to ANI we have 1., 2., and for 6. that $I' = I_1, m' = m_1$. The canonical transformation property grants us 3. It remains to show 4. and 5., i.e. that $f_I(M)$ and N represent the correct numbers of value/place pairs in the environment of $f_T(t)$ to enable it when evaluated with asg_f:

$$\begin{aligned}
&\sum_{i \in f_I(M)} (f_A \otimes f_P) \circ m \circ f_I^{-1}(i) \\
&= \sum_{i \in M} (f_A \otimes f_P) \circ m(i) && (f_I \text{ inj.}) \\
&= (f_A \otimes f_P)^{\oplus} \sum_{i \in M} m_t(i) && (\forall i \in M : m_t(i) = m(i), \text{due to def. } p(t, S, asg)) \\
&= (f_A \otimes f_P)^{\oplus} \circ (\overline{asg} \times id_P)^{\oplus} \circ pre_{AN_t}(t) && ((t, asg) \text{ enabled in } L) \\
&= (\overline{asg_f} \times id_P)^{\oplus} \circ (f_\Sigma^{\#} \otimes f_P)^{\oplus} \circ pre_{AN_t}(t) && (\text{Lemma 4.2}) \\
&= (\overline{asg_f} \times id_P)^{\oplus} \circ pre \circ f_T(t) && (f \text{ AHLI morph.}) \\
&= pre_A(f_T(t), asg_f) && (\text{def. } pre_A)
\end{aligned}$$

and analogously,

$$
\begin{aligned}
&\sum_{i\in N}(f_A^* \otimes f_P^*) \circ m_1 \circ f_I^{*-1}(i) \\
&= \sum_{i\in N}(f_A^* \otimes f_P^*) \circ m_1(i) && (f_I^*(N) = N) \\
&= (f_A^* \otimes f_P^*)^{\oplus} \sum_{i\in M} n_t(i) && (\forall i \in N : n_t(i) = m_1(i), \text{due to constr. } m_1) \\
&= (f_A^* \otimes f_P^*)^{\oplus} \circ (\overline{asg} \times id_P)^{\oplus} \circ pre_{AN_t}(t) && ((t, asg) \text{ enabled in } L) \\
&= (\overline{asg_f} \times id_P)^{\oplus} \circ (f_\Sigma^{\#} \otimes f_P)^{\oplus} \circ post_{AN_t}(t) && (\text{Lemma 4.2}) \\
&= (\overline{asg_f} \times id_P)^{\oplus} \circ post \circ f_T(t) && (f \text{ AHLI morph.}) \\
&= post_A(f_T(t), asg_f) && (\text{def. } post_A)
\end{aligned}
$$

□

Lemma 4.2 (Translated Assignments)**.**
Given two AHLI nets

$$ANI_i = (\Sigma_i = (S_i, OP_i, X_i), P_i, T_i, pre_i, post_i, cond_i, type_i, A_i, I_i, m_i), i \in \{1, 2\}$$

a transition assignment $(t, asg : X_1 \to A_1) \in CT_{ANI_1}$*, and an AHLI net morphism* $f = (f_\Sigma, f_P, f_T, f_A, f_I) : ANI_1 \to ANI_2$ *with* f_Σ *injective on the variables of* $Var(t)$*, it holds for all terms* $term \in T_{OP_1}(Var(t))$ *that*

$$\overline{asg_f} \circ f_\Sigma^{\#}(term) = f_A \circ \overline{asg}(term), \quad \text{where } asg_f = f_A \circ asg \circ f^{-1}_{\Sigma|Var(t)} : Var(f_T(t)) \to A_2$$

Proof by structural induction over all Σ_1 *terms over variables of* $Var(t)$*.* [16] First, note that because (f_Σ, f_A) is a generalized algebra homomorphism we have for all constants c and operations op in OP_1

$$f_A(c_{A_1}) \overset{f_A \text{ homomorph.}}{=} c_{V_{f_\Sigma}(A_2)} \overset{\text{def. } V_{f_\Sigma}}{=} (f_\Sigma(c))_{A_2} \tag{1}$$

$$f_A \circ op_{A_1} \overset{f_A \text{ homomorph.}}{=} op_{V_{f_\Sigma}(A_2)} \circ f_A \overset{\text{def. } V_{f_\Sigma}}{=} (f_\Sigma(op))_{A_2} \circ f_A \tag{2}$$

Case 1: $t = x \in Var(t)$
$f_A \circ \overline{asg}(x) = \overline{f_A \circ asg}(x) = \overline{f_A \circ asg \circ f^{-1}_{\Sigma|Var(t)}} \circ f_\Sigma^{\#}(x)$
because $f_{\Sigma|Var(t)} : Var(t) \to Var(f_T(t))$ is surjective due to f being a net morphism and hence f_Σ bijective on variables in $Var(t)$.

Case 2: $t = (c :\to s) \in T_{OP_1}$
$f_A \circ \overline{asg}(c) = f_A(c_{A_1}) \overset{(1)}{=} (f_\Sigma(c))_{A_2} = (f_\Sigma^{\#}(c))_{A_2} = \overline{asg_f} \circ f_\Sigma^{\#}(c)$
The last equality holds because a constant c would be evaluated by the extension of just any variable assignment to c_{A_2}.

[16] A categorical proof using free constructions can be found in [EP97].

Case 3: $t = op(t_1, \ldots, t_n) \in T_{OP_1}(Var(t))$ with $t_1, \ldots, t_n$ satisfying the property to be proven.

$$\begin{aligned}
& f_A \circ \overline{asg}(op(t_1, \ldots, t_n)) = f_A \circ op_{A_1}(\overline{asg}(t_1), \ldots, \overline{asg}(t_n)) \\
\overset{(2)}{=} & (f_\Sigma(op))_{V_{f_\Sigma}(A_2)}(f_A \circ \overline{asg}(t_1), \ldots, f_A \circ \overline{asg}(t_n)) \\
\overset{ind.assumpt.}{=} & (f_\Sigma(op))_{V_{f_\Sigma}(A_2)}(\overline{asg_f} \circ f_\Sigma^\#(t_1), \ldots, \overline{asg_f} \circ f_\Sigma^\#(t_n)) \\
= & \overline{asg_f}\left((f_\Sigma(op))\,(f_\Sigma^\#(t_1), \ldots, f_\Sigma^\#(t_n))\right) \\
= & \overline{asg_f} \circ f_\Sigma^\#\,(op(t_1, \ldots, t_n))
\end{aligned}$$

□

The second item of the previous theorem covers all possible canonical direct transformations by any transition rule $\varrho(t, S, asg)$ in an arbitrary net ANI, without aasuming that $t \in ANI$ or $M \subseteq I_{NI}$. For the special case that a transition rule $\varrho(t, S, asg)$ is applied on an inclusion match, the second item reduces to the following corollary, which is more similar to the theorem's first item.

Corollary 4.3 (Equivalence of Canonical Transformations and Firing of AHLI Nets)**.**
Given a canonical transformation $ANI \overset{\varrho(t,S,asg),f}{\Longrightarrow} ANI_1$ such that the match $f : L \to ANI$ is an inclusion, then the consistent token assignment (t, asg) is enabled in ANI under S with firing step $ANI \overset{t,asg}{\rightarrowtail} ANI_1$.

This follows directly from 2. of Theorem 4.1.

Theorem 4.4 (Token-injective AHLI Net Morphisms preserve Firing Steps)**.**
For each AHLI net morphism $f : ANI_1 \to ANI_2$, such that f_I is injective and f_Σ is injective on all sets $Var(t)$ for all $t \in T_1$[17], and each firing step $ANI_1 \overset{t,asg}{\rightarrowtail} ANI_1'$ there exists a firing step $ANI_2 \overset{f_T(t),asg_f}{\rightarrowtail} ANI_2'$ and a AHLI net morphism $f' : ANI_1' \to ANI_2'$ (depicted in diagram (1) below).

$$\begin{array}{ccc}
ANI_1 & \overset{t,asg}{\rightarrowtail} & ANI_1' \\
{\scriptstyle f}\downarrow & (1) & \downarrow{\scriptstyle f'} \\
ANI_2 & \underset{f_T(t),asg_f}{\rightarrowtail} & ANI_2'
\end{array}$$

Proof. Given $f : ANI_1 \to ANI_2$ and $ANI_1 \overset{t,asg}{\rightarrowtail} ANI_1'$ via some S as above, we have by the first part of Theorem 4.1 on page 52 the canonical direct transformation with $\varrho(t, S, asg) = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ given by the pushouts (1) and (2) in Fig. 35 on the facing page.

Note that $f : ANI_1 \to ANI_2$ satisfies the gluing condition w.r.t l', because $l'_P = id_{P_1}, l'_T = id_{T_1}$ and $IP_I = \emptyset$ due to injectivity of f_I. This allows to construct the

[17] See the "Context Condition" in [EP97].

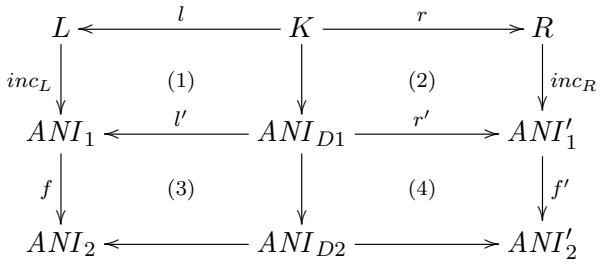


Figure 35: Double pushout diagrams for canonical direct transformation of ANI_1 and ANI_2 with $\varrho(t, S, asg)$

canonical transformation with extended rule $ANI_1 \overset{l'}{\leftarrow} ANI_{D1} \overset{r'}{\rightarrow} ANI'_1$ along pushouts (3) and (4). Hence, also $(1+3)$ and $(2+4)$ are pushouts, defining a canonical direct transformation via $\varrho(t, S, asg)$. The second part of Theorem 4.1 on page 52 implies $ANI_2 \overset{f_T(t), asg_f}{\rightarrowtail} ANI'_2$. □

Remark (Firing-preserving diagrams for (t, asg) correspond to an extension diagram for $\varrho(t, asg)$). The diagram in Fig. 35 corresponds to an extension diagram for rule $\varrho(t, asg)$ and f (as depicted below) because $(1) - (4)$ are pushouts.

$$\begin{array}{ccc} ANI_1 & \overset{\varrho(t,asg),inc}{\Longrightarrow} & ANI'_1 \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f'} \\ ANI_2 & \overset{\varrho(t,asg),f\circ inc}{\Longrightarrow} & ANI'_2 \end{array}$$

4.4 Algebraic Higher-Order Nets with Individual Tokens ($AHOI_{PTI}$)

In this section we treat the Algebraic Higher-Order nets with Individual Tokens ($AHOI_{PTI}$). Actually the $AHOI_{PTI}$ nets are Algebraic High-Level nets with Individual Tokens (AHLI) (see Sect. 4.1.1 on page 40) with a well-defined signature Σ_{PTI} and Σ_{PTI}-Algebra A.

After giving the definition of $AHOI_{PTI}$, we give the definition of the signature Σ_{PTI} by giving the set of the sorts and the operation symbols [HME05]. After giving a short motivation of our signature Σ_{PTI}, we define the Σ_{PTI}-Algebra A [HME05] that builds the semantic of the core of $AHOI_{PTI}$ nets.

4.4.1 Definition Algebraic Higher-Order Nets with individual Tokens

Definition 4.15 (Algebraic Higher-Order Nets with individual Tokens)**.**
An algebraic higher-order net with individual tokens is an algebraic high-level Net according to Def. 4.1 on page 40, as

$$AHOI_{PTI} = (\Sigma, P, T, pre, post, cond, type, A, I, m),$$

with

- $\Sigma = (\Sigma_{PTI})$,
- $A = \Sigma_{PTI}$-Algebra,

$AHOI_{PTI}$ nets have all the properties and definitions in **AHLINets**.

4.4.2 Definition(Signature Σ_{PTI})

We define the signature for the algebraic higher-order with individual tokens ($AHOI_{PTI}$) below [HME05].

$\Sigma_{PTI} =$	
sorts :	Transition, Bool, Mor, Rule, ObjectNet, ONMSet, Selection
opns :	$tt, ff :\rightarrow Bool$ $enabled : ObjectNet\ Transition\ Selection \rightarrow Bool$ $fire : ObjectNet\ Transition\ Selection \rightarrow ObjectNet$ $applicable : Rule\ Mor \rightarrow Bool$ $transform : Rule\ Mor \rightarrow ObjectNet$ $empty : ONMSet \rightarrow Bool$ $coproduct : ObjectNet\ ObjectNet \rightarrow ObjectNet$ $isomorphic : ObjectNet\ ObjectNet \rightarrow Bool$ $left : Rule \rightarrow ObjectNet$ $insert : ObjectNet\ ONMSet \rightarrow ONMSet$ $make : ObjectNet \rightarrow ONMSet$ $splitable : ObjectNet \rightarrow Bool$ $dom : Mor \rightarrow ObjectNet$ $cod : Mor \rightarrow ObjectNet$ $split : ObjectNet \rightarrow ONMSet$ $X := (X_s)_{s \in \{Transition, Bool, Mor, Rule, ObjectNet, ONMSet, Selection\}}$

The signature Σ_{PTI} has the following sorts: Transition represents the transitions of the object nets, Bool represents the boolean data type, Mor represents the set of PTI Morphisms, Rule denotes **PTINet** rules, ObjectNet is the set of the object nets (PTI Nets), ONMSet denotes the monoid over the set of the object nets and Selection represents the set of the token selections. And the operations: *tt* and *ff*, *enabled* for checking whether the transition of an object net is enabled under a token selection to fire. The operation *fire* fires an enabled transition of an object net, the operation *applicable* checks whether a rule is applicable to a given match morphism, the operation *transform*, which applies a rule to a given match morphism and realizes the rule-based transformation, the operation *empty* checks whether a given set of object nets is empty, the operation

coproduct builds the disjoint union of two given object nets, the *isomorphic* operation checks whether two given object nets are isomorphic to each other, the operation *left* gives the object net on the left-hand side of a given rule, *insert* is the operation that adds an object net to a sum of object nets, the operation *make* makes a set of object nets containing exactly one given object net, *splitable* checks whether an object net is the disjoint union of more than one components of object nets, the operation *dom* gives the domain of a given morphism and the operation *cod* gives the codomain of a given morphism, and finally the *split* operation splits an object net that consists of n components into n object nets.

4.4.3 Definition(Σ_{PTI}-Algebra A)

Σ_{PTI}-Algebra A

$A_{Transition} := T_0,\ A_{Bool} := \{true, false\}$,
$A_{Mor} := \{f |\ f :\ NI_1 \to NI_2\ PTI\ Morphism\ with\ NI_1,\ NI_2 \in PTIObject\}$
$PTIObject := \{NI | NI = (P, T, pre, post, I, m), P \subseteq P_0, T \subseteq T_0, I \subseteq I_0\}$
$A_{Rule} := \{r | r = ((L \xleftarrow{i_1} I \xrightarrow{i_2} R), NAC)\ rule\ with\ injections\ i_1, i_2$
$\quad and\ NAC \subseteq \{c | (c : L \to Nac)\} | c\ is\ a\ PTI\ morphism\}$,
$A_{ObjectNet} := PTIObject \cup \{undef\}$,
$A_{ONMSet} := A^{\oplus}_{ObjectNet}$,
$A_{Selection} := \{(M, m, N, n) | M, N \subseteq I_0, m : M \to P_0, n : N \to P_0\}$

$tt_A := true \in A_{Bool}$

$ff_A := false \in A_{Bool}$

$enabled_A : A_{ObjectNet} \times A_{Transition} \times A_{Selection} \to A_{Bool}$

$$(on, t, s) \mapsto \begin{cases} tt_A & \\ \quad if: & \\ \qquad \text{t is enabled under s} & \\ \qquad \text{and } t \in T & \\ \qquad \text{and } on = (P, T, pre, post, I, m) & \\ \qquad \text{and } s = (M, m, N, n) & \\ ff_A & \text{, else.} \end{cases}$$

$fire_A : A_{ObjectNet} \times A_{Transition} \times A_{Selection} \to A_{ObjectNet}$

$$(on, t, s) \mapsto \begin{cases} (P, T, pre, post, I', m') & \text{, if } enabled_A(on, t, s) = tt_A \\ undef & \text{, else.} \end{cases}$$

for: $on = (P, T, pre, post, I, m), I' = (I \backslash M) \cup N$,

$$m' = I' \to P\ with :\ m'(x) = \begin{cases} m(x) & \text{, if } x \in I \backslash M \\ n(x) & \text{, else.} \end{cases}$$

$applicable_A : A_{Rule} \times A_{Mor} \to A_{Bool}$

$$(r, f) \mapsto \begin{cases} tt_A & \text{, if } r \text{ is applicable at match } f \\ ff_A & \text{, else.} \end{cases}$$

$transform_A : A_{Rule} \times A_{Mor} \to A_{ObjectNet}$

$$(r, f) \mapsto \begin{cases} NI_2 & \text{, if } applicable_A(r, f) = tt_A \\ undef & \text{, else.} \end{cases}$$

with the direct transformation is written as:

$NI_1 \overset{r,f}{\Longrightarrow} NI_2$

$empty_A : A_{ONMSet} \to A_{Bool}$

$$(m) \mapsto \begin{cases} tt_A & \text{, if } m = 0 \\ ff_A & \text{, else.} \end{cases}$$

$coproduct_A : A_{ObjectNet} \times A_{ObjectNet} \to A_{ObjectNet}$

$$(NI_1, NI_2) \mapsto \begin{cases} undef & \text{, if } NI_1 = undef \text{ or } NI_2 = undef \\ NI_3 & \text{, else.} \end{cases}$$

with:

$NI_3 = (P_3, T_3, pre_3, post_3, I_3, m_3)$

$P_3 = P_1 \uplus P_2$

$T_3 = T_1 \uplus T_2$

$$pre_3 : T_3 \to P_3^{\oplus}, \qquad pre_3(t) = \begin{cases} pre_1(t) & \text{, if } t \in T_1 \\ pre_2(t) & \text{, else.} \end{cases}$$

$$post_3 : T_3 \to P_3^{\oplus}, \qquad post_3(t) = \begin{cases} post_1(t) & \text{, if } t \in T_1 \\ post_2(t) & \text{, else.} \end{cases}$$

$isomorphic_A : A_{ObjectNet} \times A_{ObjectNet} \to A_{Bool}$

$$(on_1, on_2) \mapsto \begin{cases} tt_A & \text{, if } on_1 \cong on_2 \\ ff_A & \text{, else.} \end{cases}$$

$on_1 \cong on_2 :$ *there is a* $PTI - Morphism\ f\ : on_1 \to on_2$

with: $f = (f_P, f_T, f_I)$, and f_P, f_T, f_I are bijection functions.

$left_A : A_{Rule} \to A_{ObjectNet}$

$left_A(r : ((L \overset{i_1}{\leftarrow} I \overset{i_2}{\rightarrow} R), NAC)) = L$

$insert_A : A_{ObjectNet} \times A_{ONMSet} \to A_{ONMSet}$

$$insert(n, m) = m \oplus n$$

$$make_A : A_{ObjectNet} \to A_{ONMSet}$$
$$(n) \mapsto n$$

$$splitable_A : A_{ObjectNet} \to A_{Bool}$$
$$(n) \mapsto \begin{cases} tt_A & \text{, if } \exists\ n_1, n_2 \in A_{ObjectNet}, \\ \quad with: & \\ \qquad coproduct(n_1, n_2) = n & \\ \qquad and\ n_1 \neq \emptyset & \\ \qquad and\ n_2 \neq \emptyset & \\ ff_A & \text{, else.} \end{cases}$$

$$dom_A : A_{Mor} \to A_{ObjectNet}$$
$$dom_A(f : on_1 \to on_2) = on_1$$

$$cod_A : A_{Mor} \to A_{ObjectNet}$$
$$cod_A(f : on_1 \to on_2) = on_2$$

$$split_A : A_{ObjectNet} \to A_{ONMSet}$$
$$(n) \mapsto \begin{cases} make_A(n) & \text{, if } splitable(n) = ff_A \text{ or } n = undef \\ insert_A(n_1, split_A(n_2)) & \text{, else.} \\ \quad with: & \\ \qquad splitable(n_1) = ff_A & \\ \qquad and\ coproduct(n_1, n_2) = n & \\ \qquad and\ n_1 \neq \emptyset & \\ \qquad and\ n_2 \neq \emptyset & \end{cases}$$

The Σ_{PTI}-Algebra A is the Algebra for the $AHOI_{PTI}$ nets. We explain some of the functions in our Algebra below:

The function $enabled_A$ takes an object net (PTI net) (see Sect. 3.1 on page 27) of set $A_{ObjectNet}$ its transition of set $A_{Transition}$ and a token selection (see Sect. 3.2 on page 28) of set $A_{Selection}$ and returns *true* if the transition is enabled to be fired under this token selection, otherwise *false*. If a given transition of an object net is enabled to be fired so the function $fire_A$ fires this transition and returns an object net with its follower marking. $applicable_A$ is responsible for the checking, whether a given rule is applicable to transform the object net of the match of a morphism. If a given rule is

applicable at a given morphism match so the function $transform_A$ applies this rule on the match of this morphism (see Sect. 3.4 on page 31). The function $coproduct_A$ builds the disjoint union of two object nets, but if at least one of its inputs is *undef*, it returns an error *undef*. With the function $isomorphic_A$ it is possible to check the existence of an isomorphism between two given object nets (PTI nets). The function $insert_A$ adds an object net to the sum of object nets. We can check whether an object net consists of the disjoint union of more than one object nets by using the function $splitable_A$. This function returns *true* if there is a possibility to split an object net into two non empty object nets, otherwise it returns *false*. The function $split_A$ is the function that completes the hitherto existing functions towards the formal modelling of RONs. It enables us to make a sum of object nets from a given object net. If the object net consists of at most one component of object net then we get a sum of object net containing this object net that may also be empty, otherwise we split the object net recursively as long as possible until we get all of its components as single object nets in a sum.

5 The Formal Modelling of RONs as $AHOI_{PTI}$ Nets

In this section we realize the main issue of formal modelling of the Reconfigurable Object Nets (RONs) as Algebraic High-Level Nets with Individual tokens (AHLIs). First we give the formal modelling of the low-level nets (object nets), by giving their definition as low-level PTI nets. After giving the firing behaviour of their transitions we give the definition of the formal modelling of the high-level nets and the firing behaviour of the high-level transitions of RONs. Finally we demonstrate the formalization with the example Producer/Consumer (see Sect. 5.4 on page 72).

5.1 The Formal Modelling of Low-Level Nets (Object Nets)

We formalize the object nets of the reconfigurable object nets as PTI nets (see Sect. 3.1 on page 27). The places and transitions remain the same, we add the identification function, thus we identify every token by the place which contains it. Furthermore we define the set of these identifiers. In our example in Sect. 5.4 on page 72 we have two producers and two consumers, the definition of them is as below:

$$prod_1 = (P_{prod_1},\ T_{prod_1},\ pre_{prod_1},\ post_{prod_1},\ I_{prod_1},\ m_{prod_1}),$$

with:

$$P_{prod_1} = \{produceReady_1,\ deliverReady_1,\ prodBuffer_1\}$$

$$T_{prod_1} = \{produce_1,\ deliver_1\}$$

$$pre_{prod_1}(produce_1) = produceReady_1$$

$$pre_{prod_1}(deliver_1) = deliverReady_1$$

$$post_{prod_1}(produce_1) = deliverReady_1$$

$post_{prod_1}(deliver_1) = produceReady_1 \oplus prodBuffer_1$

$I_{prod_1} = \{p_1\}$

$m_{prod_1}(p_1) = \{produceReady_1\}$

$prod_2 = (P_{prod_2},\ T_{prod_2},\ pre_{prod_2},\ post_{prod_2},\ I_{prod_2},\ m_{prod_2}),$

with:

$P_{prod_2} = \{produceReady_2,\ deliverReady_2,\ prodBuffer_2\}$

$T_{prod_2} = \{produce_2,\ deliver_2\}$

$pre_{prod_2}(produce_2) = produceReady_2$

$pre_{prod_2}(deliver_2) = deliverReady_2$

$post_{prod_2}(produce_2) = deliverReady_2$

$post_{prod_2}(deliver_2) = produceReady_2 \oplus prodBuffer_2$

$I_{prod_2} = \{p_2\}$

$m_{prod_2}(p_2) = \{produceReady_2\}$

$cons_1 = (P_{cons_1},\ T_{cons_1},\ pre_{cons_1},\ post_{cons_1},\ I_{cons_1},\ m_{cons_1})$

with:

$P_{cons_1} = \{consumeReady_1,\ removeReady_1,\ consBuffer_1\}$

$T_{cons_1} = \{remove_1,\ consume_1\}$

$pre_{cons_1}(remove_1) = consBuffer_1 \oplus removeReady_1$

$pre_{cons_1}(consume_1) = consumeReady_1$

$post_{cons_1}(remove_1) = consumeReady_1$

$post_{cons_1}(cosume_1) = removeReady_1$

$I_{cons_1} = \{c_1\}$

$m_{cons_1}(c_1) = \{removeReady_1\}$

$$cons_2 = (P_{cons_2},\ T_{cons_2},\ pre_{cons_2},\ post_{cons_2},\ I_{cons_2},\ m_{cons_2})$$

with:

$$P_{cons_2} = \{consumeReady_2,\ removeReady_2,\ consBuffer_2\}$$

$$T_{cons_2} = \{remove_2,\ consume_2\}$$

$$pre_{cons_2}(remove_2) = consBuffer_2 \oplus removeReady_2$$

$$pre_{cons_2}(consume_2) = consumeReady_2$$

$$post_{cons_2}(remove_2) = consumeReady_2$$

$$post_{cons_2}(cosume_2) = removeReady_2$$

$$I_{cons_2} = \{c_2\}$$

$$m_{cons_2}(c_2) = \{removeReady_1\}$$

Example. Figure 36 visualizes this example.

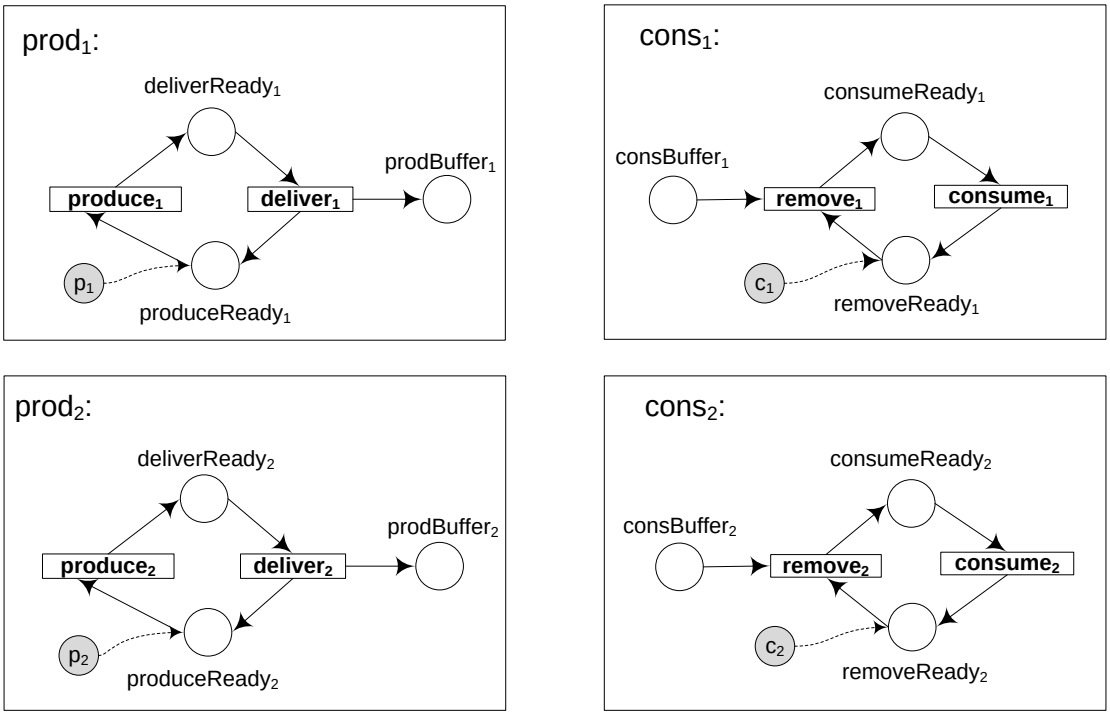


Figure 36: Object Nets in $AHOI_{PTI}$

5.2 Formalization of Firing Behaviour of Low-Level Transitions

The low-level transitions (object nets transitions) are now PTI net transitions, so they have the firing behaviour as defined in Sect. 3.2 on page 28.

In our example in sec. 5.4 on page 72 the transition $produce_1$ is *enabled* under the *token selection*

$(M_{prod_1}, m_{prod_1}, N_{prod_1}, n_{prod_1})$, where

- $M_{prod_1} = \{p_1\}$
- $N_{prod_1} = \{p_1'\}$
- $n_{prod_1}(p_1') = \{deliverReady_1\}$.

After the enabled transition $produce_1$ fires, we get the following marking $(I'_{prod_1}, m'_{prod_1})$, where

$$I'_{prod_1} = (I_{prod_1} \backslash M_{prod_1}) \cup N_{prod_1} = (\{p_1\} \backslash \{p_1\}) \cup \{p_1'\} = \{p_1'\}$$

$$m'_{prod_1} : I'_{prod_1} \to P_{prod_1} \text{ with } m'_{prod_1}(p_1') = \{deliverReady_1\}$$

and the new PTI net

$$prod_1' = (P_{prod_1},\ T_{prod_1},\ pre_{prod_1},\ post_{prod_1},\ I'_{prod_1},\ m'_{prod_1})\ .$$

5.3 The Formal Modelling of High-Level Net and of the Firing Behaviour of RONs

RONs have two kinds of high-level places, i.e. places of the type ObjectNet for object nets (P/T nets) and places of the type Rule for rules (see Sect. 2 on page 9). In the Σ_{PTI}-Algebra A of $AHOI_{PTI}$ (see Def. 4.4.3 on page 59) we defined the corresponding sort $A_{ObjectNet}$ and A_{Rule}. $A_{ObjectNet}$ is the set of the PTI nets with an additional error element for the occurrence of an undefined PTI net. A_{Rule} for the set of rules. Moreover, we defined another sort A_{ONMSet} that denotes the monoid over the set of object nets (PTI nets) (see Def. 3.1 on page 27). Thus the formal modelling of the places is the same of the $AHOI_{PTI}$ place, which are actually AHLI places (see Def. 4.1 on page 40).

In order to give the correct formal definition of the firing behaviour of RON transitions we specify for every kind of their transitions the minimum set of firing conditions to be defined in the corresponding transitions of $AHOI_{PTI}$.

Transitions of kind STANDARD: We can realize the functions of the transitions of this kind in a very trivial way, i.e. for *Copy Object Net, Copy Object Net and Move Rule, Move Object Net, Delete Object Net* (see Sect. 2.3.1 on page 11). Up to the transitions including the function *Union of Object Nets*, i.e. *Union of Object Nets, Union and Copy of Object Nets, Union and Copy of Object Nets and Copy of Rule*, the firing conditions set for this function is:

$$\{on' = coproduct(on_1, on_2)\}$$

where on_1 and on_2 are the nets from the pre-domain of the transition and on' is the net built of the disjoint union of on_1 and on_2 in its post-domain. For building the disjoint union of more than two Object Nets the function *coproduct* can be used nested, e.g. if we want to build the disjoint union of the object nets on_1, on_2, on_3 and on_4, the firing condition set for this case is as folow:

$$\{on' = coproduct(on_1, coproduct(on_2, coproduct(n_3, n_4)))\}$$

Example. Figure 37 shows the visualization of the formal modelling of the transition *Union and Copy of Object Nets* of kind STANDARD in $AHOI_{PTI}$. As we know this transition takes two object nets in its pre-domain, builds the disjoint union of them and puts a copy of the resulting object net on every place of type (kind)[18] *ObjectNet* in its post-domain.

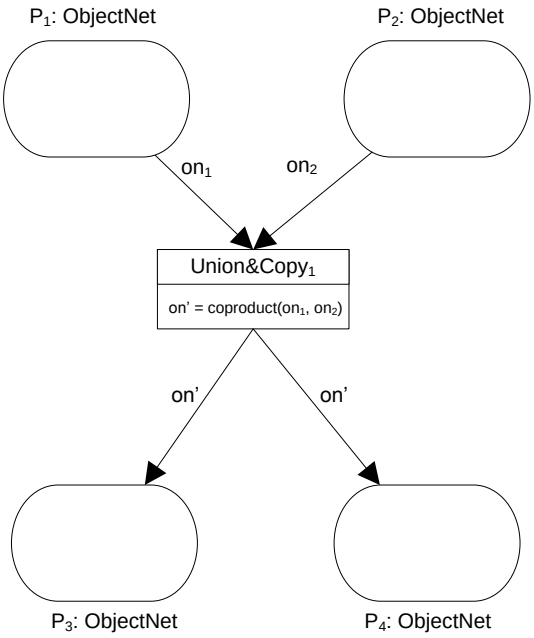


Figure 37: The formal modelling of the transition Union and Copy of Object Nets of kind STANDARD

Example. Figure 38 on the facing page shows the visualization of the formal modelling of the transition *Union and Copy of Object Nets and Copy of Rule* of kind STANDARD in $AHOI_{PTI}$, which takes two object nets and a rule from its pre-domain, builds the disjoint union of these object nets, duplicates the resulting object net and puts the copies in its post-domain. Furthermore it duplicates the rule and puts the copies in every place of type *Rule* in its post-domain. The transition $Union\&Copy_2$ in this example takes the rule r from the place P_1 , the object

[18] We use the words *kind* in RON and *type* in $AHOI_{PTI}$ as synonymous

nets on_1 from the place P_2 and on_2 from the place P_3. It builds the *coproduct* (disjoint union) of on_1 and on_2 and puts a copy of the resulting object net on' on each place of type *ObjectNet* in the post-domain, i.e. on places P_6 and P_7. Moreover, it puts a copy of the rule r, which is now marked with r'[19] on each place of type *Rule*, i.e. on places P_4 and P_5

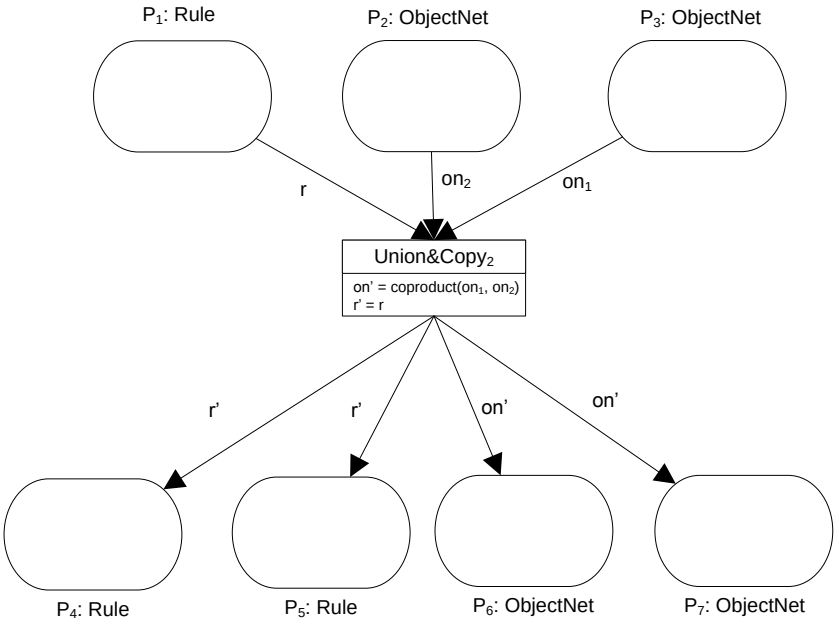


Figure 38: The formal modelling of the transition Union and Copy of Object Nets and Copy of Rule of kind STANDARD

Transitions of kind FIRE: The firing conditions set for the transitions of kind FIRE is:

$$\{enabled(on, t) = tt,\ on' = fire(on, t)\}$$

where on is the PTI net from the pre-domain, t is the transition of on that has to be fired, on' the transition with the follower marking that has to be put in post-domain.

Example. Figure 39 on the following page shows the formal modelling of a transition of kind FIRE in $AHOI_{PTI}$. In this variant the transition fires an enabled transition of an object net in its pre-domain and puts a copy of the object net with the follower marking in every place of type *ObjectNet* in its post-domain.

[19] See the firing behaviour of AHLI nets Def. 4.2 on page 43

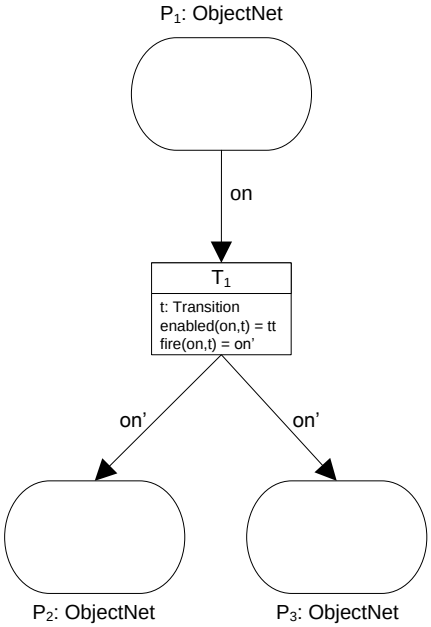


Figure 39: Formal modelling of a transition of kind FIRE

Transitions of kind APPLYRULE: The firing conditions set for the transitions of kind APPLYRULE is:

$\{dom(m) = left(r),\ cod(m) = on,\ applicable(r, m) = tt,\ on' = transform(r, m)\}$

where r is the rule that has to be applied to a net n in the codomain of a match morphism m, n' is the resulting net after applying the rule r if it is applicable to the match morphism m.

Example. Figure 40 on the next page shows the formal modelling of the transitions of kind APPLYRULE in $AHOI_{PTI}$. The transition *ApplyRule* applies the rule r to the object net *on*. After firing this transition we get the new object net on' in post-domain and the same rule in post-domain but marked with a new variable r'. The other transition *Union&Apply* applies the rule r from the place P_1 of type *Rule* to the resulting *coproduct* (disjoint union) of the two object nets n_1 from the place P_2 and n_2 from the place P_3 of type *ObjectNet* and puts the new object net n' after the successful rule application on place P_5 of type *ObjectNet* in its post-domain. Furthermore this transition puts the rule r' on place P_4 of type *Rule* in its post-domain, which is the same rule r but on different place.

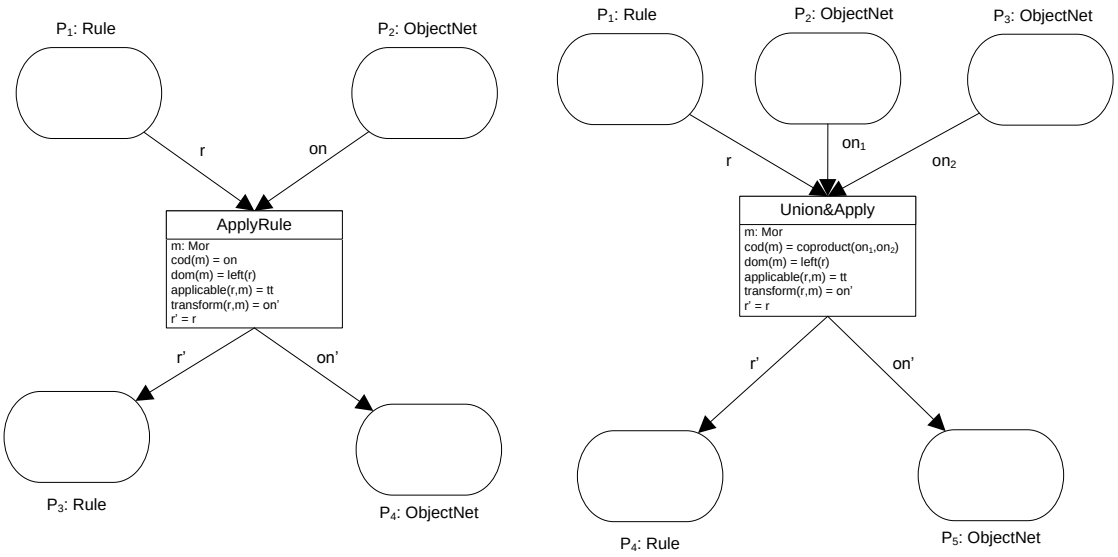


Figure 40: Formal modelling of a transition of kind APPLYRULE

Transitions of kind SPLIT: This kind of transition is the most complicated transition to be modelled in $AHOI_{PTI}$. The main problem is how to split one object net (PTI net) consisting of n components into n independent single object nets. In RON it is implemented recursively, so we defined a recursive function $split_A$ that uses the functions $insert_A$, $make_A$ and $splitable_A$. The function $make_A$ makes a set containing a net that can not be split, otherwise the function $insert_A$ puts the nets stepwise recursively in a set. The split function in $AHOI_{PTI}$ is well-defined, because it puts only the nets that consist of exactly one component into the set of the split net and the termination is also ensured. That is realized by using the boolean function $splitable_A$ that checks, whether an given object net consist at least of two connected components. The function $split_A$ splits only splittable nets. The function $make_A$ takes an empty object net, a single object net or an undefined object net and put it on the post-domain.

In order to simulate the function of a transition of kind SPLIT in $AHOI_{PTI}$ we need two more transitions and one more place. For better intuition, we show the visualization of this simulation below.

Example. Figure 41 on the following page shows the formal modelling of a transition of kind SPLIT in $AHOI_{PTI}$ nets.

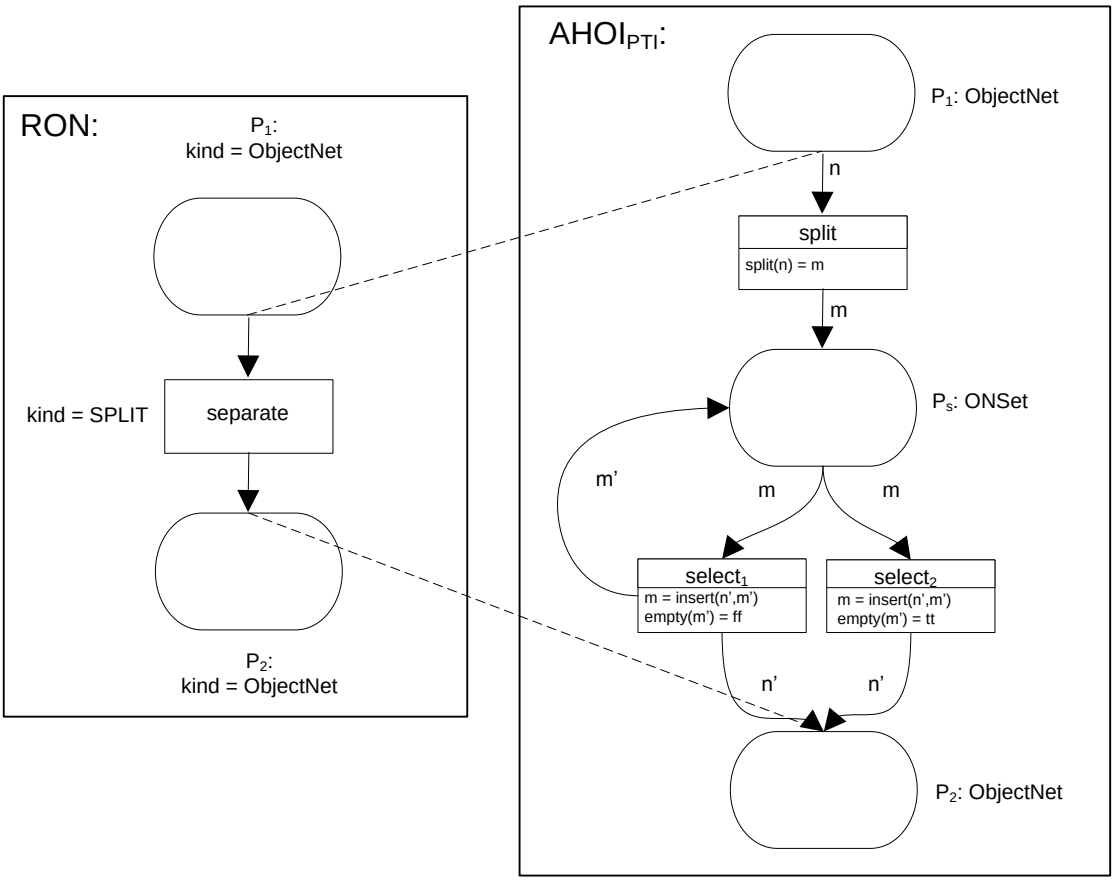


Figure 41: Formal modelling of a transition of kind SPLIT in $AHOI_{PTI}$ nets

5.4 Example: Producer/Consumer as $AHOI_{PTI}$ Net

In this example we give the formal modelling of Producer/Consumer (Sect. 2.4 on page 18) $AHOI_{PTI}$ as follows:

$$ANI_{PC} = (\Sigma_{PC}, P_{PC}, T_{PC}, pre_{PC}, post_{PC}, cond_{PC}, type_{PC}, A_{PC}, I_{PC}, m_{PC}),$$

where

$\Sigma_{PC} = \Sigma_{PTI}$,

$A_{PC} = \Sigma_{PTI}$-Algebra A

$P_{PC} = \{Producers, Consumers, ProdCons, Prod/Cons_1, Prod/Cons_2, ProdAndCons, Merger, Separator, CheckProd, CheckCons\}$,

$T_{PC} = \{prodAction, consAction, prodMeetsCons, deal, disconnect, split, select_1, select_2, prodLeaves, consLeaves\}$,

$pre_{PC}(prodAction) = (p', Producers)$

$pre_{PC}(consAction) = (c', Consumers)$

$pre_{PC}(deal) = (on, ProdCons)$

$pre_{PC}(prodMeetsCons) = (p, Producers) \oplus (c, Consumers) \oplus (r, Merger)$

$pre_{PC}(disconnect) = (on', ProdCons) \oplus (r, Separator)$

$pre_{PC}(split) = (n, Prod/Cons_1)$

$pre_{PC}(select_1) = (m, Prod/Cons_2)$

$pre_{PC}(select_2) = (m, Prod/Cons_2)$

$pre_{PC}(prodLeaves) = (n', ProdAndCons) \oplus (r, CheckProd)$

$pre_{PC}(consLeaves) = (n', ProdAndCons) \oplus (r, CheckCons)$

$post_{PC}(prodAction) = (p'', Producers)$ with $p'' = fire(p', t)$

$post_{PC}(consAction) = (c'', Consumers)$ with $c'' = fire(c', t)$

$post_{PC}(deal) = (on'', ProdCons)$ with $on'' = fire(on, t)$

$post_{PC}(prodMeetsCons) = (on, ProdCons) \oplus (r, Merger)$

$post_{PC}(disconnect) = (n, Prod/Cons_1) \oplus (r, Separator)$

$post_{PC}(split) = (m, Prod/Cons_2)$

$post_{PC}(select_1) = (m', Prod/Cons_2) \oplus (n', ProdAndCons)$

$post_{PC}(select_2) = (n', ProdAndCons)$

$post_{PC}(prodLeaves) = (p', Producers) \oplus (r, CheckProd)$

$post_{PC}(consLeaves) = (c', Consumers) \oplus (r, CheckCons)$

$cond_{PC}(prodAction) = \{t \in X_{Transition},\ enabled(p', t) = tt,\ p'' = fire(p', t)\}$

$cond_{PC}(consAction) = \{t \in X_{Transition},\ enabled(c', t) = tt,\ c'' = fire(c', t)\}$

$cond_{PC}(deal) = \{t \in X_{Transition},\ enabled(on, t) = tt,\ on'' = fire(on, t)\}$

$cond_{PC}(prodMeetsCons) = \{m \in X_{Mor},\ dom(m) = left(r),\ cod(m) = coproduct(p, c),$
$applicable(r, m) = tt,\ on = transform(r, m)\}$

$cond_{PC}(disconnect) = \{m \in X_{Mor},\ dom(m) = left(r),\ cod(m) = on',\ applicable(r, m) =$
$tt,\ n = transform(r, m)\}$

$cond_{PC}(split) = \{split(n) = m\}$

$cond_{PC}(select_1) = \{m = insert(n', m'),\ empty(m') = ff\}$

$cond_{PC}(select_2) = \{m = insert(n', m'),\ empty(m') = tt\}$

$cond_{PC}(prodLeaves) = \{m \in X_{Mor},\ dom(m) = left(r),\ cod(m) = n',\ applicable(r, m) = tt,\ p' = transform(r, m)\}$

$cond_{PC}(consLeaves) = \{m \in X_{Mor},\ dom(m) = left(r),\ cod(m) = n',\ applicable(r, m) = tt,\ c' = transform(r, m)\}$

$type_{PC}(Producers) = ObjectNet$

$type_{PC}(Consumers) = ObjectNet$

$type_{PC}(ProdCons) = ObjectNet$

$type_{PC}(Prod/Cons_1) = ObjectNet$

$type_{PC}(Prod/Cons_2) = ObjectNet$

$type_{PC}(ProdAndCons) = ObjectNet$

$type_{PC}(Merger) = Rule$

$type_{PC}(Separator) = Rule$

$type_{PC}(CheckProd) = Rule$

$type_{PC}(CheckCons) = Rule$

$I_{PC} = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$

$m_{PC}(x_1) = (prod_1, Producers)$

$m_{PC}(x_2) = (prod_2, Producers)$

$m_{PC}(x_3) = (cons_1, Consumers)$

$m_{PC}(x_4) = (cons_2, Consumers)$

$m_{PC}(x_5) = (merge, Merger)$

$m_{PC}(x_6) = (separate, Separator)$

$m_{PC}(x_7) = (checkProd, CeckProd)$

$m_{PC}(x_8) = (checkCons, CheckCons)$

Example. Figure 42 on the facing page shows the *Producer/Consumer* as $AHOI_{PTI}$ net.

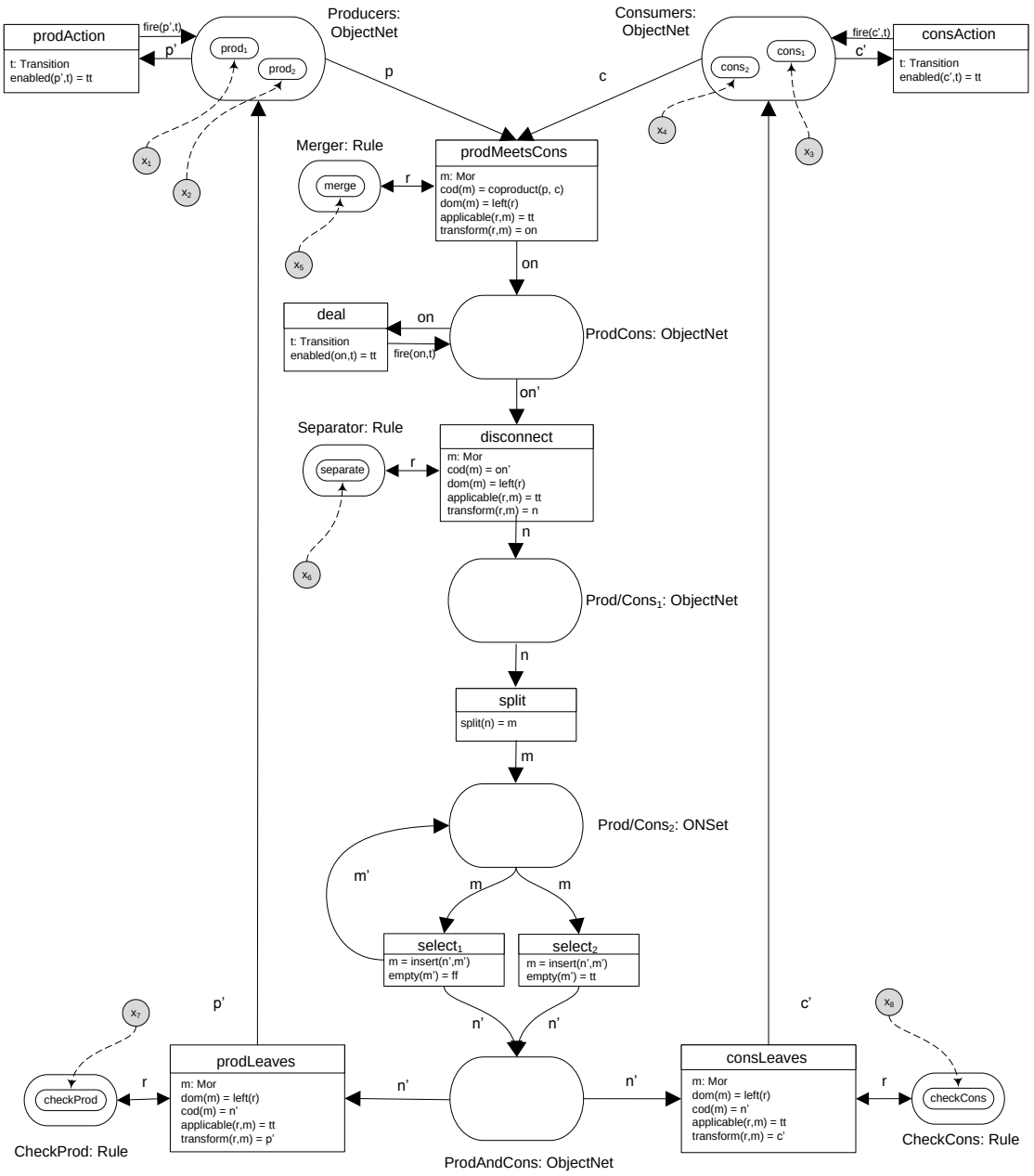


Figure 42: Producer Consumer in initial state

Example. Figure 43 on the next page shows the *Producer/Consumer* as $AHOI_{PTI}$ net together with two object nets $prod_1$ and $cons_1$ in initial state.

Example. Figure 44 on page 75 shows the enabled transition *connect* fire between a producer and a consumer via firing the high-level transition *deal*

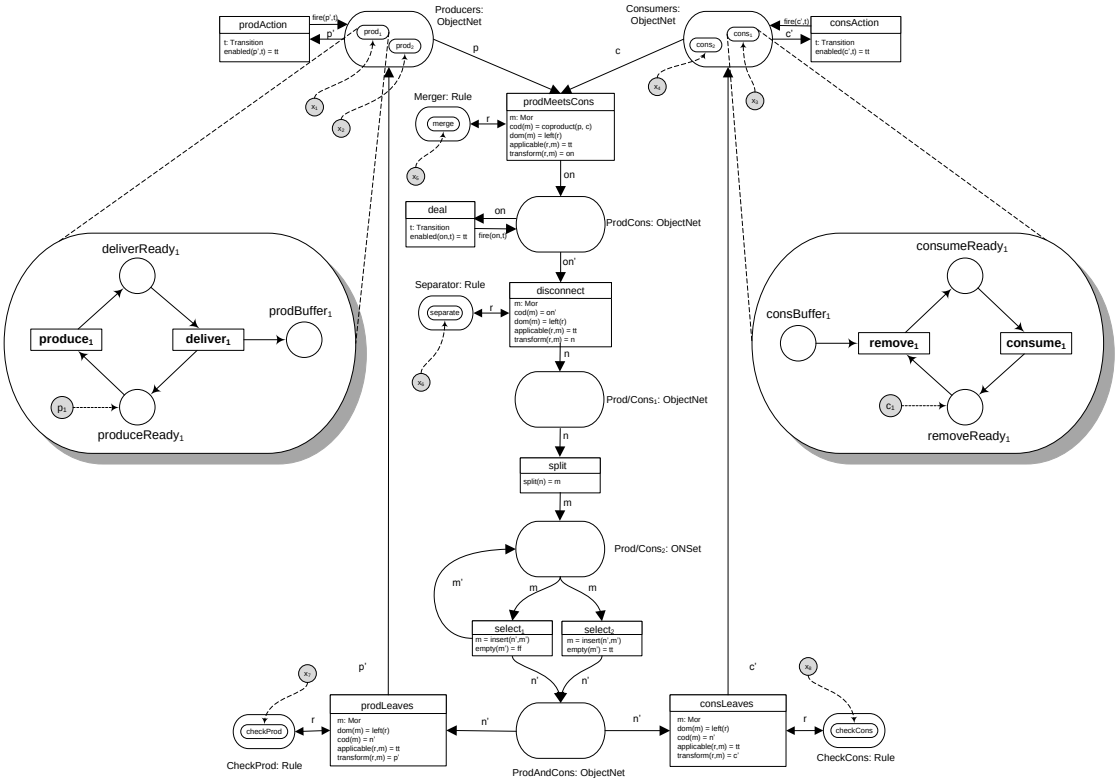


Figure 43: Producer/Consumer with two object nets $prod_1$ and $cons_1$ in initial state

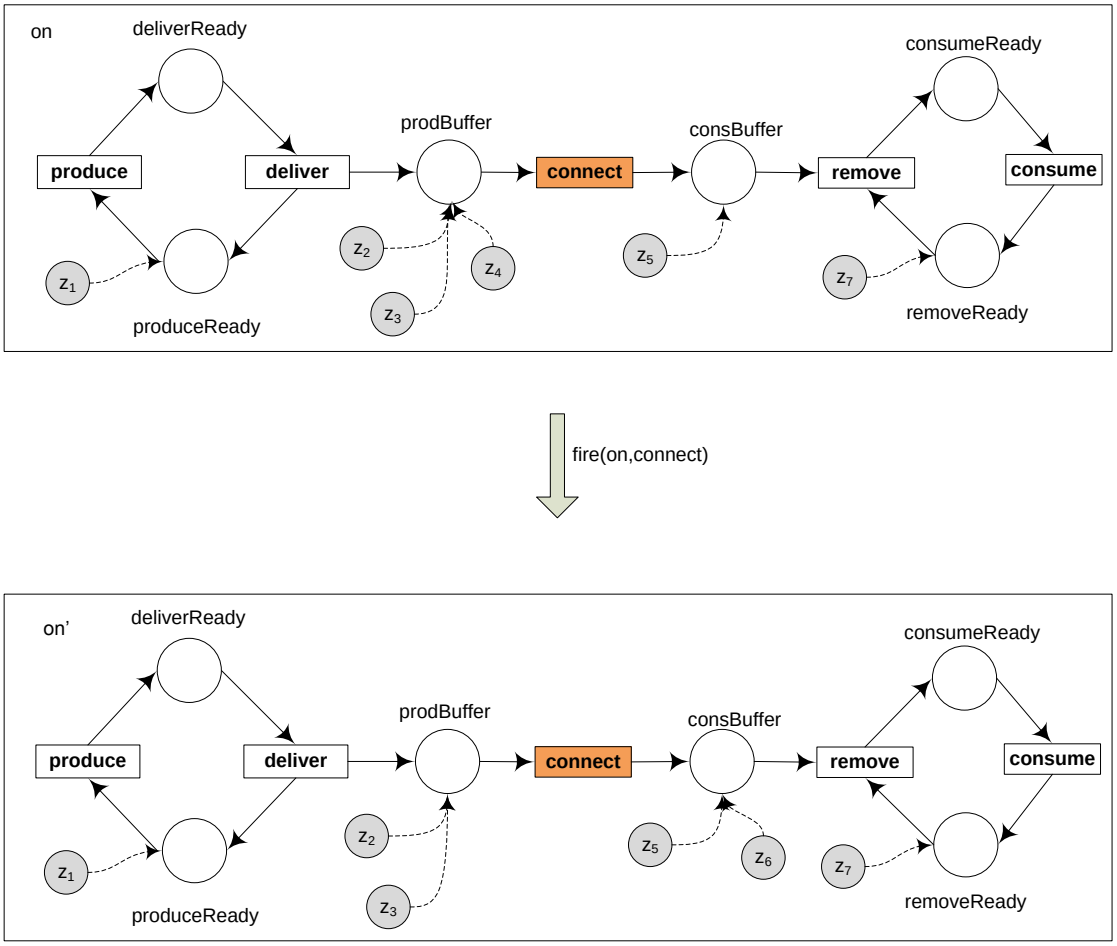


Figure 44: Firing the transition *deal*

6 Realization of the analysis based on permutation equivalence

In this section we give an efficient method to analyze whether two given firing sequences in an AHLI net are permutation equivalent or not [HCEK10]. Since $AHOI_{PTI}$ nets are special AHLI nets, the results formulated for AHLI nets below, are also valid for $AHOI_{PTI}$ nets. First we show in Theorem 6.1 the existence of effective unions in AHLI nets. The existence of effective union is an important condition for the decomposition of the morphisms, which allow us to build a process skeleton on the components with $\mathcal{M}$-morphisms (see Sect. 3 in [Her08]). To analyze such permutation equivalences we benefit from the property of Theorem 4.1 on page 52 to simulate a given firing sequence in an AHLI net as transformation sequence. Moreover we use Theorem 6.2 on page 79 to analyze the possible permutation equivalence of this firing sequence. Finally we demonstrate these techniques on example (Sect. 6.3 on page 79).

6.1 Characterization of AHLI Net Firing Sequence as Transformation Sequence

In order to analyze a firing sequence of $AHOI_{PTI}$ nets we can use the analysis based on techniques of permutation equivalence in [HCEK10]. Before using these techniques we have to show that AHLI nets have effective unions, i.e. the union of two subobjects exists in the $\mathcal{M}$-adhesive category **AHLINets** and can be constructed within **AHLINets** as the pushout over the intersection in the category of subobjects. "A suboject of an object T in a category **C** is an equivalence class of monomorphisms $a : A \to B$ "[Her09]. The category of these subobjects of T is $\mathbf{Sub}_{\mathcal{M}}(T)$ [Her09]. Thus we have to show that the constructed AHLI net is a subnet of a given net T [Her08].

Theorem 6.1 (Union in $\mathbf{Sub}_{\mathcal{M}}(T)$ for $(\mathbf{AHLINets}, \mathcal{M})$)**.**
Let T be an AHLI net in **AHLINets** *and $\mathcal{M}$ be the class of monomorphisms which are isomorphisms on the data part. The union $A \cup B$ of two AHLI nets A and B in $\mathbf{Sub}_{\mathcal{M}}(T)$ exists and is given by the pushout* (1) *over the intersection $A \cap B$ with the induced $\mathcal{M}$-morphism $u : A \cup B \to T$ of the pushout* (1)*.*

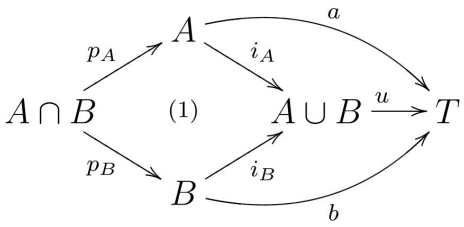


Figure 45: Theorem 6.1

Proof. The intersection exists by Lemma 1 in [Her08] and the pushout exists (see Theorem 5.3 in [MGE+10]), because $p_A \in \mathcal{M}$. The induced morphism u is monomorphic using

Theorem 4.7 in [LS04] and the existence of pullbacks along $\mathcal{M}$-morphisms. It remains to show that u is also an $\mathcal{M}$-morphism. This means that u is additionally an isomorphism on the data part and identity on the signature part. Since a and i_A are $\mathcal{M}$-morphisms they are isomorphisms on the data part and identities on the signature part. Using the inverse isomorphism on the data part of i_A we have that u is an isomorphism on the data part and using the identity on the signature part we have that u is an identity on the signature part by the composition of isomorphisms and identities. □

By **Theorem** 6.1 on the preceding page we are now able to apply the analysis techniques based on permutation equivalence.

6.2 Analysis based on Permutation Equivalence

A firing sequence of an AHLI net can be simulated as canonical AHLI transformation sequence (see 4.1 on page 52). In our example in Sect. 5.4 on page 72 we can translate every firing step as a transformation step, by defining the corresponding rule for rule-based transformation (see figure 46 on the following page). We can use this property to analyze the possible permutation equivalence in a given firing sequence for an $AHOI_{PTI}$ net by using the Theorem 6.2 on page 79.

$$\begin{array}{ccccc} L & \xleftarrow{l} & K & \xrightarrow{r} & R \\ {\scriptstyle f}\downarrow & (PO_1) & \downarrow & (PO_2) & \downarrow{\scriptstyle f^*} \\ ANI_1 & \longleftarrow & ANI_1' & \longrightarrow & ANI_2 \end{array}$$

Example. Figure 46 on the following page shows a transformation step as firing step.

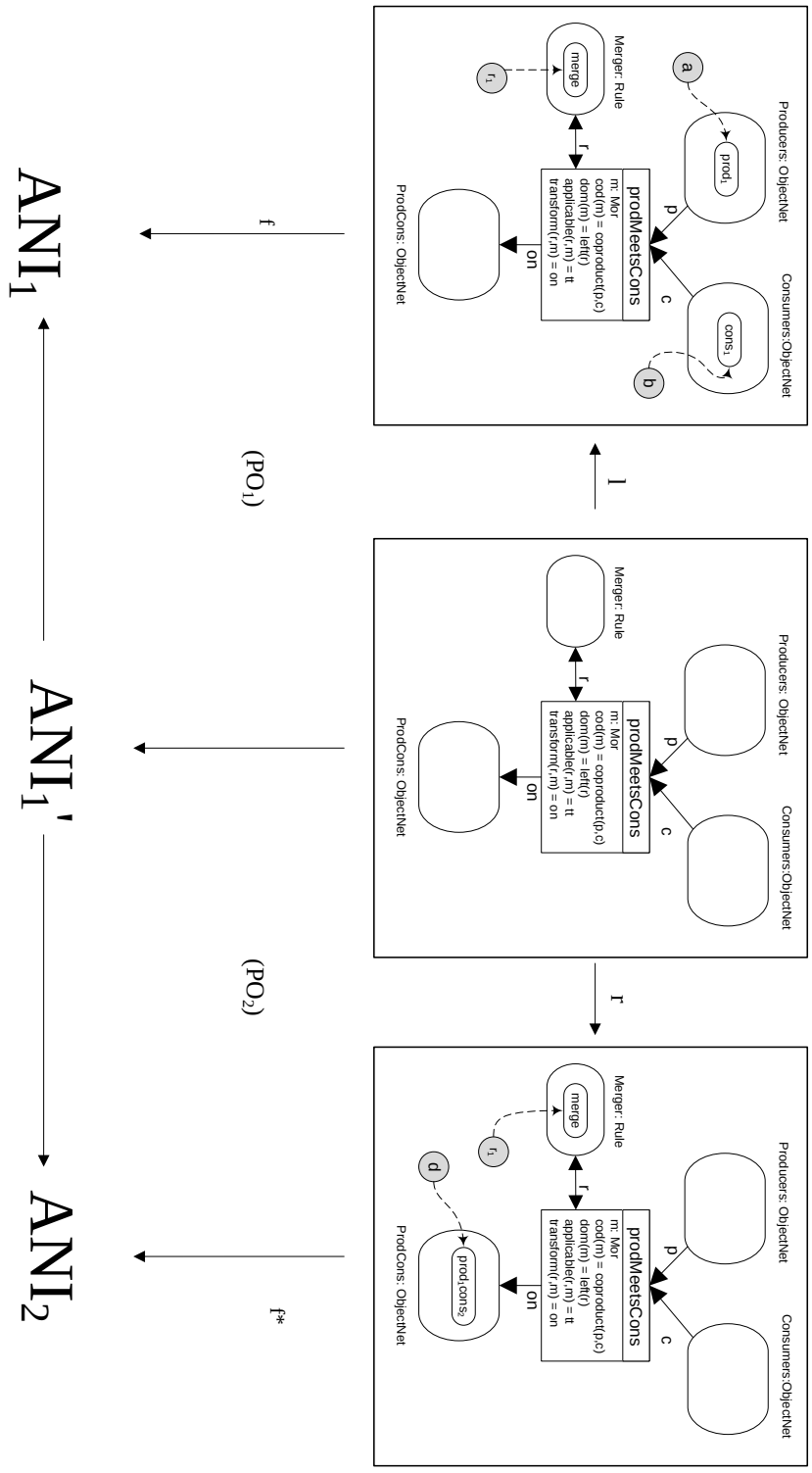


Figure 46: Transformation step as firing step

As introduced in [HCEK10] we define the notion *permutation equivalence of transformation sequences* as follows:

Definition 6.1 (Permutation Equivalence of Transformation Sequences)**.**
Two transformation sequences d and d' respecting NACs are permutation equivalent, written $d' \overset{\pi}{\approx} d$ if, disregarding the NACs, they are switch equivalent as for Def. 3 in [HCEK10].

The analysis of permutation equivalence for transformation sequences is based on **Theorem** 6.2 below, which is proven by Corollary 1 in [HCEK10].

Theorem 6.2 (Analysis of Permutation Equivalence of Derivations)**.**
Let d be a transformation sequence respecting NACs of an $\mathcal{M}$-adhesive transformation system with NACs over **C**, *and let* $\underline{Prc}(d)$ *be its process skeleton. Then, a derivation d' is permutation equivalent to d (d' $\overset{\pi}{\approx}$ d) if and only if the sequence of names $s_{d'}$, which contains all the direct derivations of d in the order they are actually fired in d', is a transition complete firing sequence of the marked P/T Petri net* $\underline{Prc}(d)$.

It is very efficient to use the above technique based on Petri Nets in order to verify whether two permutations of a given firing sequence are equivalent. A process skeleton must be built where each transition of this process skeleton represents a transformation step of the transformation sequence that simulates the given firing sequence.

6.3 Example

In this example we show how useful and efficient it is to use the technique of the permutation equivalence [20] to analyze whether the two given permutations d and d' of transformation sequences are equivalent (d' $\overset{\pi}{\approx}$ d) or not.

Let d be the first transformation sequence of a firing sequence beginning from the initial state in our example Producer/Consumer in Sect. 5.4 on page 72 with:

d_1: $fire(prod_1, produce_1)$ via firing high-level transition *prodAction*, i.e. the enabled transition $produce_1$ of $prod_1$ fires by firing the transition *prodAction* and the object net with the follower marking is $prod_1^1$ with $fire(prod_1, produce_1) = prod_1^1$.

d_2: $fire(prod_1^1, deliver_1)$ via firing high-level transition *prodAction*, i.e. the enabled transition $deliver_1$ of $prod_1^1$ fires by firing the transition *prodAction* and the object net with the follower marking is $prod_1^2$ with $fire(prod_1^1, produce_1) = prod_1^2$. There is an item produced after these two steps above.

d_3: $fire(prod_1^2, produce_1)$ via firing high-level transition *prodAction*, i.e. the enabled transition $produce_1$ of $prod_1^2$ fires by firing the transition *prodAction* and the object net with the follower marking is $prod_1^3$ with $fire(prod_1^2, produce_1) = prod_1^3$.

[20] Note that is this case we can consider the permutation equivalence as a switched equivalence, because in this example there are no NACs that have to be considered [Her08, Her09, HCEK10]

d_4: $fire(prod_1^3, deliver_1)$ via firing high-level transition *prodAction*, i.e. the enabled transition $deliver_1$ of $prod_1^3$ fires by firing the transition *prodAction* and the object net with the follower marking is $prod_1^4$ with $fire(prod_1^3, produce_1) = prod_1^4$. After these two steps the object nets produced an item and put it on $prodBuffer_1$. There are two items produced after these four steps above.

d_5: firing high-level transition *prodMeetsCons* and merging the two object nets $prod_1^4$ and $cons_1$ into one object net $prod_1cons_1$.

d_6: $fire(prod_1cons_1, connect_1)$ via firing high-level transition *deal* and the object net with the follower marking is $prod_1cons_1^1$ with $fire(prod_1cons_1, connect_1) = prod_1cons_1^1$.

d_7: $fire(prod_1cons_1^1, connect_1)$ via firing high-level transition *deal* and the object net with the follower marking is $prod_1cons_1^2$ with $fire(prod_1cons_1^1, connect_1) = prod_1cons_1^2$.

d_8: firing high-level transition *disconnect* which deletes the transition $connect_1$ and puts the object net $prod_1cons_1^3$ on $Prod/Cons_1$.

d_9: firing high-level transition *split* and making a set s_1 of object nets containing $prod_1^5$ and $cons_1^1$.

d_{10}: $fire(prod_2, produce_2)$ via firing high-level transition *prodAction*, i.e. the enabled transition $produce_2$ of $prod_2$ fires by firing the transition *prodAction* and the object net with the follower marking is $prod_2^1$ with $fire(prod_2, produce_2) = prod_2^1$.

d_{11}: $fire(prod_2^1, deliver_2)$ via firing high-level transition *prodAction*, i.e. the enabled transition $deliver_2$ of $prod_2^1$ fires by firing the transition *prodAction* and the object net with the follower marking is $prod_2^2$ with $fire(prod_2^1, produce_2) = prod_2^2$. There is now an item produced through these two steps above.

d_{12}: $fire(prod_2^2, produce_2)$ via firing high-level transition *prodAction*, i.e. the enabled transition $produce_2$ of $prod_2^2$ fires by firing the transition *prodAction* and the object net with the follower marking is $prod_2^3$ with $fire(prod_2^2, produce_2) = prod_2^3$.

d_{13}: $fire(prod_2^3, deliver_2)$ via firing high-level transition *prodAction*, i.e. the enabled transition $deliver_2$ of $prod_2^3$ fires by firing the transition *prodAction* and the object net with the follower marking is $prod_2^4$ with $fire(prod_2^3, produce_2) = prod_2^4$. After these two steps the object nets produced an item and put it on $prodBuffer_2$. There are two items produced after these four steps above.

d_{14}: firing high-level transition *prodMeetsCons* and merging the two object nets $prod_2^4$ and $cons_2$ into one object net $prod_2cons_2$.

d_{15}: $fire(prod_2cons_2, connect_2)$ via firing high-level transition *deal* and the object net with the follower marking is $prod_2cons_2^1$ with $fire(prod_2cons_2, connect_2) = prod_2cons_2^1$.

d_{16}: $fire(prod_2cons_2^1, connect_2)$ via firing high-level transition *deal* and the object net with the follower marking is $prod_2cons_2^2$ with $fire(prod_2cons_2^1, connect_2) = prod_2cons_2^2$.

d_{17}: firing high-level transition *disconnect* which deletes the transition $connect_2$ and puts the object net $prod_2cons_2^3$ on $Prod/Cons_1$.

d_{18}: firing high-level transition *split* and making a set s_2 of object nets containing $prod_2^5$ and $cons_2^1$.

d_{19}: firing high-level transition $select_1$ and putting the new object net $prod_2^6$ on $ProdAndCons$ and $cons_2^2$ in the set s_2^1 on $Prod/Cons_2$.

d_{20}: firing high-level transition $select_2$ and putting the new object net $cons_2^3$ on $ProdAndCons$.

d_{21}: firing high-level transition *prodLeaves* and putting the new object net $prod_2^7$ on $Producers$.

d_{22}: firing high-level transition *consLeaves* and putting the new object net $cons_2^4$ on $Consumers$.

d_{23}: $fire(cons_2^4, remove_2)$ via firing the high-level transition *consAction* with $cons_2^5 = fire(cons_2^4, remove_2)$.

d_{24}: $fire(cons_2^5, consume_2)$ via firing the high-level transition *consAction* with $cons_2^6 = fire(cons_2^5, consume_2)$.

d_{25}: $fire(cons_2^6, remove_2)$ via firing the high-level transition *consAction* with $cons_2^7 = fire(cons_2^6, remove_2)$.

d_{26}: $fire(cons_2^7, consume_2)$ via firing the high-level transition *consAction* with $cons_2^8 = fire(cons_2^7, consume_2)$.

d_{27}: firing high-level transition $select_1$ and putting the new object net $prod_1^6$ on $ProdAndCons$ and $cons_1^2$ in the set s_1^1 on $Prod/Cons_2$.

d_{28}: firing high-level transition $select_2$ and putting the new object net $cons_1^3$ on $ProdAndCons$.

d_{29}: firing high-level transition *prodLeaves* and putting the new object net $prod_1^7$ on $Producers$.

d_{30}: firing high-level transition *consLeaves* and putting the new object net $cons_1^4$ on $Consumers$.

d_{31}: $fire(cons_1^4, remove_1)$ via firing the high-level transition *consAction* with $cons_1^5 = fire(cons_1^4, remove_1)$.

d_{32}: $fire(cons_1^5, consume_1)$ via firing the high-level transition *consAction* with $cons_1^6 = fire(cons_1^5, consume_1)$.

d_{33}: $fire(cons_1^6, remove_1)$ via firing the high-level transition *consAction* with $cons_1^7 = fire(cons_1^6, remove_1)$.

d_{34}: $fire(cons_1^7, consume_1)$ via firing the high-level transition *consAction* with $cons_1^8 = fire(cons_1^7, consume_1)$.

We construct the process skeleton (figure 47 on the next page) according to *Def.12.* in [HCEK10] of the transformation sequence $\underline{Prc}(d)$ as a marked Petri net with:

$d = \langle d_1; d_2; d_3; d_4; d_5; d_6; d_7; d_8; d_9; d_{10}; d_{11}; d_{12}; d_{13}; d_{14}; d_{15}; d_{16}; d_{17}; d_{18}; d_{19}; d_{20}; d_{21}; d_{22}; d_{23}; d_{24}; d_{25}; d_{26}; d_{27}; d_{28}; d_{29}; d_{30}; d_{31}; d_{32}; d_{33}; d_{34}\rangle$

From the process skeleton (figure 47 on the facing page)[21] it is now easy to find permutation equivalent sequences of the transformation sequence above by recognizing the dependencies between the transitions, each one of them representing a transformation step in our transformation sequence. In our example we have a process skeleton which consists of two independent sub nets, therefore we can take an arbitrary permutation built by changing the order between the transitions of these two sub nets of the process skeleton. Furthermore we can also change the order of the transitions which are independent respecting the read causality [Her08]. Thus, e.g. we can change the order between the transitions $d1$ and d_{18}, because they belong to different sub nets of our process skeleton $\underline{Prc}(d)$ and between the transitions d_{11} and d_{12}, because they are independent respecting the read causality [Her08].

Thus we can construct another transformation sequence d' which is permutation equivalent to d with:

$d' = \langle d'_{10}; d'_1; d'_{11}; d'_2; d'_{12}; d'_3; d'_{13}; d'_4; d'_{14}; d'_5; d'_{15}; d'_6; d'_{16}; d'_7; d'_{17}; d'_8; d'_{18}; d'_9; d'_{19}; d'_{27}; d'_{21}; d'_{29}; d'_{20}; d'_{28}; d'_{22}; d'_{30}; d'_{23}; d'_{31}; d'_{24}; d'_{32}; d'_{25}; d'_{33}; d'_{34}; d'_{26}\rangle$

d'_i is equivalent to d_i for $i \in \{1, 2, ..., 34\}$, i.e. the applied rule is the same and the new matches are equivalent and derived by stepwise applying the Local Church-Rosser Theorem [EEPT06].

By Theorem 6.2 on page 79 we know that $d' \overset{\pi}{\approx} d$, i.e. the transformation sequence d is permutation equivalent to d'.

The details of a transformation step can also be reconstructed from the process skeleton together with the generated subobject transformation system according to [Her09].

Example. Figure 47 on the next page shows the process skeleton of d.

[21] Note that the names of the transitions are $(d_1, d_2, ..., d_{34})$ in figure 47 on the next page for better intuition, while the formally defined transition names in [HCEK10] are the pure numbers $(1, 2, ..., 34)$

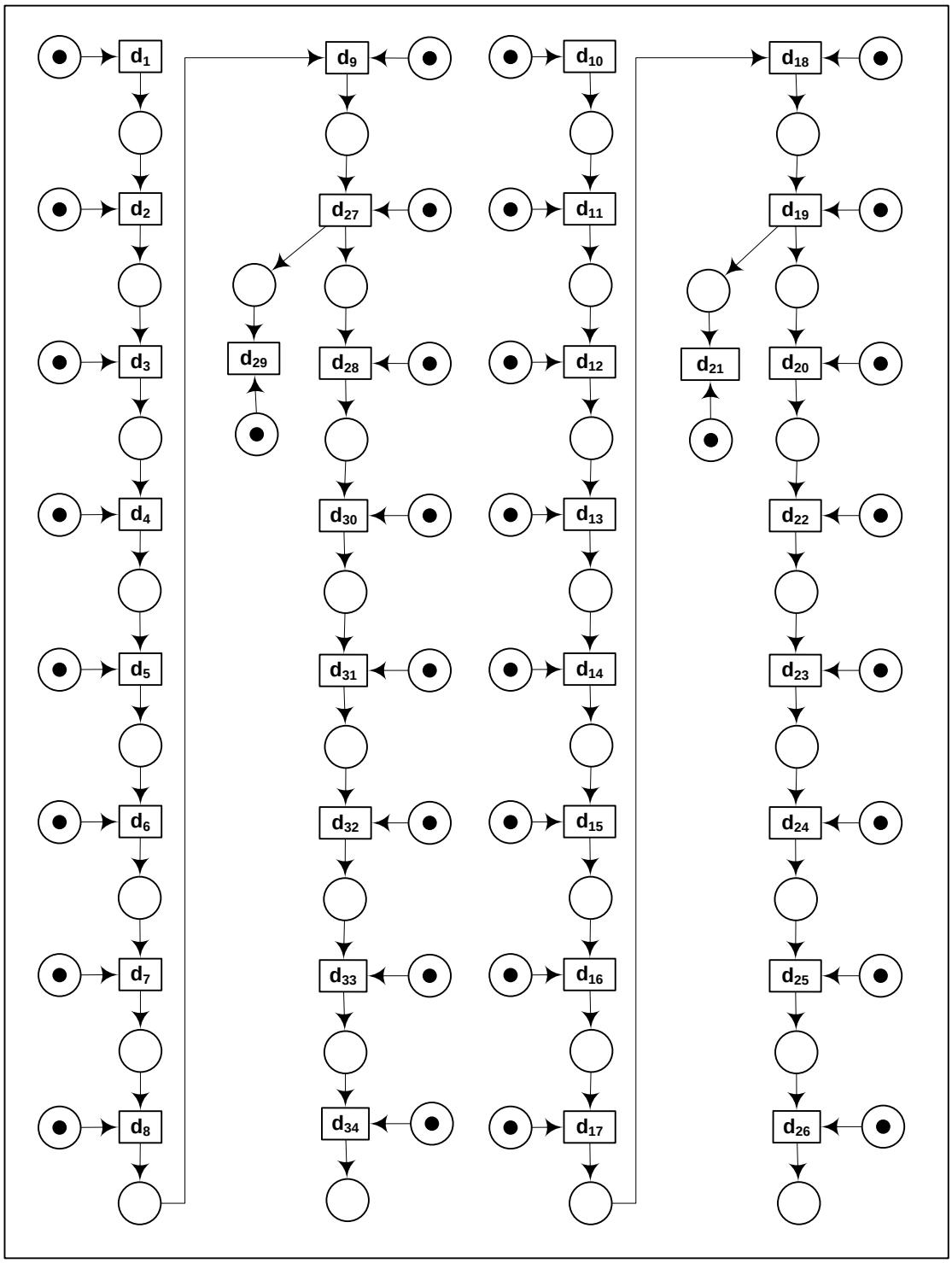


Figure 47: Process Skeleton $\underline{Prc}(d)$

7 Conclusion: Summary of Results and Future Work

The formal definition and formal modelling of Reconfigurable Object Nets (RONs) has been our first aim in this thesis. This has been achieved by giving the formal definition of the low-level nets (object nets) as place/transition nets with individual tokens and the formal definition of high-level nets as Algebraic Higher-Order Nets ($AHOI_{PTI}$) by defining the signature Σ_{PTI} and the Σ_{PTI}-Algebra A, which is an AHLI net. Furthermore the formal modelling of the firing behaviour of RON transitions has been also realized by giving the minimum set of equations that has to occur in transition (see Section 5)

The second main aim of our thesis has been to realize the analysis techniques based on permutation equivalence to verify the existence of possibly permutation equivalent firing sequences. This aim has also been achieved by:

- Proving the existence of effective unions in category **AHLINets**.
- Translating the firing sequence in **AHLINets** into the corresponding transformation sequence using the property *Equivalence of Canonical Transformations and Firing of AHLI Nets*.
- Building and analyzing the corresponding process skeleton of the transformation sequence as (P/T net) (see Section 6).

In future works the implementation of the analysis of permutation equivalence for RON editor hast to be considered, which allows the efficient analysis whether a alternative firing sequence exists at runtime. These analysis techniques are to be lifted to high-level nets, i.e. to be able to analyze the existence of permutation equivalences directly (omitting the process skeleton). Another aspect to be considered in RON editor is the translation of the firing behaviour of the transition of kind APPLYRULE into a modified transition of a new kind APPLY that will be able to apply a sequence of rules on a given object net at once respecting the application conditions of the rule applications.

References

[AGG10] TFS-Group, TU Berlin. *AGG*, 2010. `http://tfs.cs.tu-berlin.de/agg`.

[BEHM07] Enrico Biermann, Claudia Ermel, Frank Hermann, and Tony Modica. A Visual Editor for Reconfigurable Object Nets based on the ECLIPSE Graphical Editor Framework. In G. Juhas and J. Desel, editors, *Proc. 14th Workshop on Algorithms and Tools for Petri Nets (AWPN'07)*, Universität Koblenz-Landau, Germany, October 2007. GI Special Interest Group on Petri Nets and Related System Models.

[BEMS08] E. Biermann, C. Ermel, T. Modica, and P. Sylopp. Implementing Petri Net Transformations using Graph Transformation Tools. *ECEASST*, 14, 2008.

[BM08] E. Biermann and T. Modica. Independence Analysis of Firing and Rule-based Net Transformations in Reconfigurable Object Nets. In J. de Lara C. Ermel and R. Heckel, editors, *Proc. Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'08)*, volume 10. EC-EASST, 2008.

[EBO92] H. Ehrig, M. Baldamus, and F. Orejas. New concepts for amalgamation and extension in the framework of specification logics. In *Proc. WADT-COMPASS-Workshop Dourdan, 1991*, pages 199–221. Springer LNCS 655, 1992.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in Theoretical Computer Science. Springer Verlag, 2006.

[EHP+08] H. Ehrig, K. Hoffmann, J. Padberg, C. Ermel, U. Prange, E. Biermann, and T. Modica. Petri Net Transformations. In *Petri Net Theory and Applications*, pages 1–16. I-Tech Education and Publication, 2008.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. Springer, Berlin, 1985.

[EMF10] Eclipse Consortium. *Eclipse Modeling Framework (EMF) – Version 2.6.1*, 2010. `http://www.eclipse.org/emf`.

[EOP06] H. Ehrig, F. Orejas, and U. Prange. Categorical Foundations of Distributed Graph Transformation. In A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, and G. Rozenberg, editors, *Proc. Third International Conference on Graph Transformation (ICGT'06)*, volume 4178 of *LNCS*, pages 215 – 229, Natal, Brazil, September 2006. Springer.

[EP97] C. Ermel and J. Padberg. Formalization of Variables in Algebraic High-Level Nets. Technical Report 97-19, Technical University Berlin, 1997.

[GEF10] Eclipse Consortium. *Eclipse Graphical Editing Framework (GEF) – Version 3.6.1*, 2010. `http://www.eclipse.org/gef`.

[HCEK10] Frank Hermann, Andrea Corradini, Hartmut Ehrig, and Barbara König. Efficient Process Analysis of Transformation Systems Based on Petri nets. Technical Report TR 2010-3, TU Berlin, 2010.

[Her08] Frank Hermann. Process Definition of Adhesive HLR Systems (Long Version) . Technical Report 2008/09, TU Berlin, Fak. IV, 2008.

[Her09] Frank Hermann. Permutation Equivalence of DPO Derivations with Negative Application Conditions based on Subobject Transformation Systems: Long Version. Technical Report 2009/10, TU Berlin, Fak. IV, 2009.

[HME05] K. Hoffmann, T. Mossakowski, and H. Ehrig. High-Level Nets with Nets and Rules as Tokens. In *Proc. of 26th Intern. Conf. on Application and Theory of Petri Nets and other Models of Concurrency*, volume 3536 of *LNCS*, pages 268–288. Springer, 2005.

[LS04] S. Lack and P. Sobociński. Adhesive Categories. In *Proc. FOSSACS 2004*, volume 2987 of *LNCS*, pages 273–288. Springer, 2004.

[MGE+10] Tony Modica, Karsten Gabriel, Hartmut Ehrig, Kathrin Hoffmann, Sarkaft Shareef, Claudia Ermel, Ulrike Golas, Frank Hermann, and Enrico Biermann. Low- and High-Level Petri Nets with Individual Tokens. Technical Report 2009/13, Technische Universität Berlin, 2010. `http://www.eecs.tu-berlin.de/menue/forschung/forschungsberichte/2009`.

[MM90] J. Meseguer and U. Montanari. Petri Nets are Monoids. *Information and Computation*, 88(2):105–155, 1990.

[PER95] J. Padberg, H. Ehrig, and L. Ribeiro. Algebraic high-level net transformation systems. *Mathematical Structures in Computer Science*, 5:217–256, 1995.

[RON10] TFS-Group, TU Berlin. *RON-Editor*, 2010. `http://tfs.cs.tu-berlin.de/roneditor`.

[TBG91] Andrzej Tarlecki, Rod M. Burstall, and Joseph A. Goguen. Some fundamental algebraic tools for the semantics of computation: Part 3: Indexed categories. *Theoretical Computer Science*, 91(2):239–264, 1991.

[Val98] Rüdiger Valk. Petri Nets as Token Objects: An Introduction to Elementary Object Nets. *Proc. of the International Conference on Application and Theory of Petri Nets*, 1998.