Electronic Communications of the EASST
Volume 68 (2014)

Proceedings of the 8th International Workshop on Graph-Based Tools (GraBaTs 2014)

Non-Deterministic Matching Algorithm for Net Transformations

Mathias Blumreiter and Julia Padberg

14 pages

Guest Editors: Matthias Tichy, Bernhard Westfechtel
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Non-Deterministic Matching Algorithm for Net Transformations

Mathias Blumreiter[1] **and Julia Padberg**[2]

[1] Technical University Hamburg-Harburg, Germany

[2] Hamburg University of Applied Sciences, Germany
julia.padberg@haw-hamburg.de

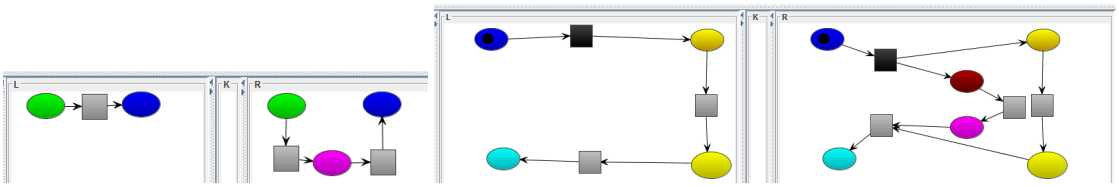
Abstract: Modeling and simulating dynamic systems require to represent their processes and the system changes within one model. To that effect, reconfigurable Petri nets consist of a place/transition net and a set of rules that can modify the Petri net. The application of a rule is based on finding a suitable match of the rule in the given net. This match is an isomorphic subnet that has to be located meeting requirements of the rule application as well as the simulation. In this paper a non-deterministic algorithm is presented for the matching in reconfigurable Petri nets. It is an extension of the VF2 algorithm for graph (sub-)isomorphisms. We show that this extension is correct and complete. Non-determinism ensures that during simulation different matches can be found for each transformation step and is hence crucial for the simulation. But non-determinism has not been present in the VF2 algorithm. For the matching algorithm non-determinism is proven.

Keywords: Reconfigurable Petri nets, matching algorithm, non-determinism, simulation, net transformation

1 Introduction

Motivation for reconfigurable Petri nets, a family of formal modeling techniques (e.g., in [EP03, LO04, KCD10]) is the observation that in increasingly many application areas the underlying system has to be dynamic in a structural sense. Complex coordination and structural adaptation at run-time (e.g., mobile ad-hoc networks, communication spaces, ubiquitous computing) are main features that need to be modeled adequately. The distinction between the net behavior and the dynamic change of its net structure is the characteristic feature that makes reconfigurable Petri nets so suitable for systems with dynamic structures.

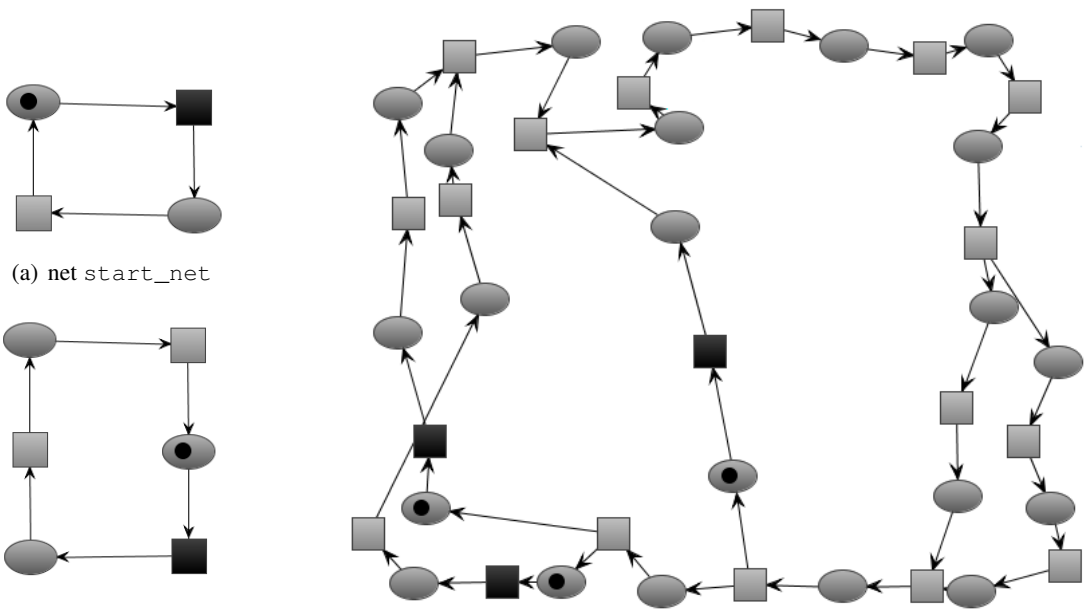
Reconfigurable Petri nets consist of marked Petri nets, i.e., a net with a marking, and a set of rules whose application modifies the net's structure at runtime. For the sake of the main focus we subsequently consider only a small and abstract example. More complex nets and rules can be found in case studies for the applications of reconfigurable Petri nets (see [Gab14, Mod12, Rei12]). As an example of a dynamic system we use a cyclic process that can either be executed or modified. These modifications change the process by inserting additional sequential steps using rule `sequential_ext` in Fig. 1(a) or by forking into parallel steps rule `parallel_ext` in Fig. 1(b). The colors of the places and transitions indicate the mappings within the rule. The net in Fig. 2(a) describes a cyclic process that can execute one step and then returns to the start. The left-hand side of the rule is the net L and shows the places



(a) rule `sequential_ext` (b) rule `parallel_ext`

Figure 1: Rules

that need to be in the context and the transition that is deleted. In the right hand side of the rule is the net R and shows the added place and transitions as well as the context. For reasons of space we have omitted the intermediate net K that denotes the context explicitly. The rule `sequential_ext` is the first rule that can be applied by matching L in net `start_net` in Fig. 2(a). Reconfigurable Petri nets allow the application of these rules together with the firing of the transitions. Let the application of rule `sequential_ext_s` be the first step, followed by a firing step. This results in the net in Fig. 2(b). The resulting net has an additional place and an additional transition, denoting the process to have been modified by inserting a sequential step. Moreover, the next step has already been executed by firing the transition in the post-domain of the first place. These steps are chosen non-deterministically, so the start net in Fig. 2(a) may evolve in 20 steps to the net in Fig. 2(c) by firing transitions or applying rules.



(a) net `start_net`

(b) net after 2 steps

(c) net after 20 steps

Figure 2: Start and intermediate nets

The simulation of reconfigurable Petri nets has to cover the full behavior. If this is not the case, not all possible runs can be simulated and the simulation is not coherent with the semantics of reconfigurable Petri nets. Hence, it is crucial that the search of the match is non-deterministic. The match is the occurrence of a left hand side L of a rule to a target net N and is given by an injective net morphism. So, the main contribution of this paper is the presentation and implementation of the non-deterministic algorithm PNVF2. This algorithm is an adaption of the VF2 algorithm [CFSV04b] for subgraph isomorphism, because a match, that is the injective occurrence morphism, can be considered to be equivalent to a net L being isomorphic to subnet of N. The PNVF2 algorithm is correct and complete and - in contrast to the VF2 algorithm - it is non-deterministic.

The paper is organized as follows: First we define reconfigurable Petri nets based on place/transition nets. In the next section we outline the algorithm, called PNVF2, that finds non-deterministically a match of the left hand side of a given rule in a given net. We explain its main function and the data structures for the representation of the state space. In Section 4 we show that the algorithm is correct and complete, state its non-determinism and discuss its complexity. n Section 5 we then discuss related and ongoing work.

2 Reconfigurable Petri Nets

We use the algebraic approach to Petri nets, so a marked place/transition net is given by $N = (P, T, pre, post, m)$ with pre- and post-domain functions $pre, post : T \to P^{\oplus}$ and a marking $m \in P^{\oplus}$, where $P^{\oplus}$ is the free commutative monoid over the set P of places. Markings $m_1, m_2 \in P^{\oplus}$ are given by multisets or linear sums of places, defined by the free commutative monoid over the set P of places. Accordingly, we can extend relations ($\leq$), addition ($\oplus$) and substraction ($\ominus$) to markings. E.g., we have $m_1 \leq m_2$ if $m_1(p) \leq m_2(p)$ for all $p \in P$. A transition $t \in T$ is m-enabled for a marking $m \in P^{\oplus}$ if we have $pre(t) \leq m$, and in this case the follower marking m' is given by $m' = m \ominus pre(t) \oplus post(t)$ and $m[t\rangle m'$ is called firing step. To obtain the weight of an arc from a place to a transition t the pre-domain function is restricted to that place, i.e. $pre(t)_{|p} = \lambda_p \in \mathbb{N}$ for $pre(t) = \sum_{p \in P} \lambda_p p$; analogously the weight of an arc from a transition to a place is given by the restriction of the post-domain function. Note, that in [Blu13] as well as ReConNet[1] decorated nets are considered. For the sake of brevity we here merely consider place/transition nets.

Place/transition nets yield an $\mathcal{M}$-adhesive HLR category. $\mathcal{M}$-adhesive HLR systems can be considered as a unifying framework for graph and Petri net transformations providing enough structure that most notions and results from algebraic graph transformation systems are available: results on parallelism and concurrency of rules and transformations, results on negative application conditions and constraints, and so on (e.g., in [EEPT06]). Net morphisms are given as a pair of mappings for the places and the transitions, so that the structure and the decoration and the marking are preserved. So, a net morphism $f : N_1 \to N_2$ between two place/transition nets $N_i = (P_i, T_i, pre_i, post_i, m_i,)$ for $i \in \{1, 2\}$ is given by $f = (f_P : P_1 \to P_2, f_T : T_1 \to T_2)$, so that $pre_2 \circ f_T = f_P^{\oplus} \circ pre_1$ and $post_2 \circ f_T = f_P^{\oplus} \circ post_1$ and $m_1(p) \leq m_2(f_P(p))$ for all $p \in P_1$. Moreover, the morphism f is called strict if both f_P and f_T are injective and $m_1(p) = m_2(f_P(p))$ holds for all $p \in P_1$.

[1] The tool ReConNet has been developed at the HAW Hamburg in various students projects (see [EHOP12]).

A rule in the DPO-approach given by three nets called left hand side L, interface K and right hand side R, respectively, and a span[2] of two strict net morphisms $K \to L$ and $K \to R$. Then a match (or occurrence) morphism $o : L \to N$ is required that identifies the relevant parts of the left hand side in the given net N. Then a transformation step $N \overset{(r,o)}{\Longrightarrow} N'$ via rule r can be constructed in two steps. Given a rule with a match $o : L \to N$ the gluing conditions have to be satisfied in order to apply a rule at a given match. These conditions ensure the result is again a well-defined net. In this case, we obtain a net N' leading to a direct transformation $N \overset{(r,o)}{\Longrightarrow} N'$ consisting of the pushouts (1) and (2) in the category of place/transition nets (see Fig. 3). So, we combine one place/transition net N together with a set of net transformation rules leading to reconfigurable place/-transition nets.

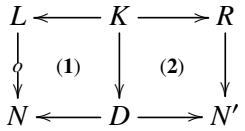


Figure 3: Net transformation

Definition 1 (Reconfigurable Nets) A reconfigurable place/transition net $RN = (N, \mathcal{R})$ is given by a place/transition net N and a set of net transformation rules $\mathcal{R}$.

3 Matching Algorithm PNVF2

In this section we outline an algorithm that finds a match of a left hand side L of a rule in a given net N. This match being an injective net morphism implies specific requirements the algorithm has to satisfy the preservation of the net structure, that is preservation of the transitions pre- and post-domain as well as the preservation of the marking. It has to discover only adequate matches with respect to gluing conditions and it has to deliver correct matches, including the possibility of delivering all matches and ensuring the non-determinism of the simulation. For graphs one of the efficient algorithms for subgraph isomorphism is the VF2 that allows the satisfaction of the above mentioned requirements.

3.1 Outline of the Algorithm

The adaptation of the VF2 algorithm [CFSV04b] to Petri nets, called PNVF2 algorithm [Blu13], uses of an intricate data structure. In contrast to the VF2 algorithm the PNVF2 algorithm needs to consider the pre- and post-domain of transitions. Given two nets $N_j = (P_j, T_j, pre_j, post_j, m_j)$ for $j \in \{1,2\}$ with pre- and post-domain functions $pre_j, post_j : T \to P^{\oplus}$ we are looking for an injective morphism, the match $M = (M_P : P_1 \to P_2, M_T : T_1 \to T_2)$. For convenience we use M as a relation $M = \{(v,w) \mid v \mapsto w\}$ as well, especially the two sets $M_1 = \{v \mid (v,w) \in M\}$ and $M_2 = \{w \mid (v,w) \in M\}$.

As a small example (in Subsect. 3.2) we consider the nets given in Fig. 4 on page 9 and search a match of $N1$ in $N2$, where the indices of places and transitions represent an fixed, but arbitrary order.

The match is computed by searching recursively the possible mappings of places and transitions. In the algorithm the match M describes an injective partial morphism defined by the

[2] Actually, ReConNet implements the co-span DPO approach, but for the matching algorithm this is irrelevant .

current match. For the computation of the match M the PNVF2 performs a recursive depth-first search. The possible mappings of the nodes constitute the search space. At each level of recursion, the algorithm tries to map a source node from net N_1 to different target nodes in N_2. For the computation of the match we do not differentiate between places and transitions, as the search space is a combination of the place and the transition mappings. At each recursion level only one type of nodes, either places or transitions, is investigated. The node type is chosen non-deterministically, yielding candidate pairs of places or transitions. These are possible mappings and they are checked for feasibility. This feasibility comprises several conditions that ensure the preservation of the net structure, that is preservation of the transitions pre- and post-domain and the preservation of the marking. For each state s of the matching process there is a corresponding partial match $M(s) = (M_P(s) : P_1 \to P_2, M_T(s) : T_1 \to T_2)$, which contains only a subset of the complete M.

Algorithm 1 PNVF2

establish arbitrary order for places and for transitions in both nets
initialize $M(s_0) = \emptyset$
initialize arrays *core*, *in* and *out*
function MATCH(intermediate state s_i)
  if $(P_1 \cup T_1) = M_1(s_i)$ **then**
    match $M(s_i)$ is complete **return** ▹ termination with a match
  else
    compute terminal and residual node sets
    compute set of candidate pairs $P(s_i)$
    for all $(v, w) \in P(s_i)$ **do**
      if $feasible(s_i, v, w) = true$ **then**
        compute s_{i+1} by
        $M(s_{i+1}) = M(s_i) \cup \{(v, w)\}$
        update arrays *core*, *in* and *out*
        MATCH(s_{i+1})
        restore arrays *core*, *in* and *out*
      end if
    end for
    if $i > 0$ **then**
      backtrack to s_{i-1}
    else
      no match found **return** ▹ termination without a match
    end if
  end if
end function

After the successful mapping of a candidate the algorithm examines a new pair at the next recursion level. In this way increasing parts of the match are computed. At each recursion level

the terminal and the residual node sets are computed. For $j \in \{1,2\}$ the sets $T^{in|out}_{P_j|T_j}(s)$[3] are based on the in- and outgoing arcs (*in* or *out*) of the current match to places and transitions for the view from net N_1 or net N_2. E.g., $T^{out}_{P_1}(s) = \{p \in (P_1 - M_1) \mid p \leq post_1(t) \text{ for } t \in M_1(s)\}$ is the set of places, that are reached from the nodes (i.e., transitions) of net N_1, that already belong to the current partial match $M(s)$. The corresponding terminal sets $T^{term}_j(s) = T^{in}_{P_j}(s) \cup T^{out}_{P_j}(s) \cup T^{in}_{T_j}(s) \cup T^{out}_{T_j}(s)$ unite the in- and outgoing places and transitions. The residual node sets $\widetilde{P_j|T_j}$ contain those nodes of N_j for $j \in \{1,2\}$ that are neither in the current match nor connected to any node of the current match, e.g., $\widetilde{P_2}(s) = P_2 - M_2(s) - T^{term}_2(s)$. Based on these sets the set of candidate pairs is computed. According to the pre-defined order, that has been fixed during initialization, the smallest place and the smallest transition of net N_1 is examined next. The possible candidate pairs of terminal nodes are given by $P_{T^{in|out}_X} = \{\min T^{in|out}_{X_1}(s)\} \times T^{in|out}_{X_2}(s)$ for $X \in \{P,T\}$. Analogously, the possible candidate pairs of residual nodes are given for $X \in \{P,T\}$ by $P_{\widetilde{X}} = \{\min \widetilde{X_1}(s)\} \times \widetilde{X_2}(s)$.

For the example (in Subsect. 3.2) the empty match of the first state s_0 is given by $M(s_0) = \emptyset$ and $P(s_0) = P_{\widetilde{P}}(s_0) = \{(p_1,p_a),(p_1,p_b),(p_1,p_c)\}$ and the first candidate is the pair (p_1,p_a), as p_1 and p_a are the first places of the corresponding order.

The PNVF2 examines the terminal out sets $P_{T^{out}_P}$ and $P_{T^{out}_T}$ first. If they are empty, the candidates given in the terminal in sets $P_{T^{in}_P}$ and $P_{T^{in}_T}$ are used. Lastly, the candidates of the residual node sets are considered for $X \in \{P,T\}$:

$$P(s) = \begin{cases} P_{T^{out}_X}(s) & \text{if } P_{T^{out}_P}(s) \cup P_{T^{out}_T}(s) \neq \emptyset \\ P_{T^{in}_X}(s) & \text{if } P_{T^{out}_P}(s) \cup P_{T^{out}_T}(s) = \emptyset \wedge P_{T^{in}_P}(s) \cup P_{T^{in}_T}(s) \neq \emptyset \\ P_{\widetilde{X}}(s) & \text{if } P_{T^{out}_P}(s) \cup P_{T^{out}_T}(s) = \emptyset \wedge P_{T^{in}_P}(s) \cup P_{T^{in}_T}(s) = \emptyset \end{cases} \tag{1}$$

For the candidate pair (p_1,p_a) from the example (in Subsect. 3.2) we have $P(s_1) = P_{T^{out}_T}(s_1) = \{t1\}$, $P_{T^{in}_T}(s_1) = \{t2\}$ and $P_{\widetilde{T}}(s_1) = \{t3\}$ in state s_1. Since Petri nets are bipartite, the corresponding sets for places are empty.

These candidate pairs are not necessarily matching pairs. Similarly to VF2 [CFSV04b], the feasibility function preserves the net structure and prunes the search space. $feasible(s,v,w)$ depends on the state s and a candidate pair $(v,w) \in P(s)$, that is either a pair of places or a pair of transitions, so for $X \in \{P,T\}$ we have $feasible(s,v,w) = feasible_X(s,v,w)$ with $(v,w) \in X_1 \times X_2$. The conditions for the feasibility are presented in six rules (see [Blu13]), which may differ for places and transitions defined below. The first rules are concerned with the preservation of the net structure, so that the found match is an injective net morphism.

- $rule_{sem,P}$ states that the marking needs to preserved.
- $rule_{sem,T}$ states that the number of in- and outgoing arcs of both transitions must be the same.

$$\begin{aligned} rule_{sem,P} &\equiv m_1(p_1) \leq m_2(p_2) \\ rule_{sem,T} &\equiv card_{pre_1}(t_1) = card_{pre_2}(t_2) \wedge card_{post_1}(t_1) = card_{post_2}(t_2) \end{aligned} \tag{2}$$

[3] $T^{in|out}_{P_j|T_j}$ is a short notation for $T^{in}_{P_j}$, $T^{out}_{P_j}$, $T^{in}_{T_j}$ or $T^{out}_{T_j}$.

- $rule_{pred,P}$ and $rule_{pred,T}$ examine the predecessors of the candidate pair. $rule_{pred,P}(s,p_1,p_2)$ ensures that for each transition t_1 of N_1 in the current match there is a transition t_2 of N_2 in the current match so that $post_1(t_1)_{|p_1} = post_2(t_2)_{|p_2}$. So, $rule_{pred,T}(s,t_1,t_2)$ ensures that for each place p_1 of N_1 in the match there is a place $p_2 = M(s)(p_1)$ of N_2 in the match so that $pre_1(t_1)_{|p_1} = pre_2(t_2)_{|p_2}$.

- Analogously, $rule_{succ,P}$ and $rule_{succ,T}$ examine the successors of the candidate pair.

$$\begin{aligned} rule_{pred,P}(s,p_1,p_2) &\equiv \forall t_1 \in M_1(s) \cap T_1 : post_1(t_1)_{|p_1} = post_2(M(s)(t_1))_{|p_2} \\ rule_{pred,T}(s,t_1,t_2) &\equiv \forall p_1 \in M_1(s) \cap P_1 : pre_1(t_1)_{|p_1} = pre_2(t_2)_{|M(s)(p_1)} \\ rule_{succ,P}(s,p_1,p_2) &\equiv \forall t_1 \in M_1(s) \cap T_1 : pre_1(t_1)_{|p_1} = pre_2(M(s)(t_1))_{|p_2} \\ rule_{succ,T}(s,t_1,t_2) &\equiv \forall p_1 \in M_1(s) \cap P_1 : post_1(t_1)_{|p_1} = post_2(t_2)_{|M(s)(p_1)} \end{aligned} \tag{3}$$

Each pair that satisfies the rules (2) and (3) leads to a correct partial match if added to the current state. However, the pursuit of all following states does not necessarily pay off. The following rules check the possibility of mapping the neighborhood of the candidates and hence prune the search space.

- The rules $rule_{in,T|P}$ and $rule_{out,T|P}$ perform a lookahead of size one for places and for transitions in the searching process by checking that there are at least as many adjacent nodes in the corresponding terminal sets of net N_2 as there are in the sets of net N_1. Since net morphisms preserve the pre- and post domain of the transitions the number of adjacent places must match.

- The rules $rule_{new,T|P}$ perform a lookahead of size two as it checks the amount of adjacent nodes in the residual node sets.

$$\begin{aligned} &rule_{in|out,P}(s_n,p_1,p_2) \\ &\quad\equiv |\{t_1 \mid t_1 \in T_{T_1}^{in|out} \wedge post_1(t_1)_{|p_1} > 0\}| \leq |\{t_2 \mid t_2 \in T_{T_2}^{in|out} \wedge post_2(t_2)_{|p_2} > 0\}| \\ &\quad\wedge |\{t_1 \mid t_1 \in T_{T_1}^{in|out} \wedge pre_1(t_1)_{|p_1} > 0\}| \leq |\{t_2 \mid t_2 \in T_{T_2}^{in|out} \wedge pre_2(t_2)_{|p_2} > 0\}| \\ &rule_{in|out,T}(s_n,t_1,t_2) \\ &\quad\equiv |\{p_1 \mid p_1 \in T_{P_1}^{in|out} \wedge pre_1(t_1)_{|p_1} > 0\}| = |\{p_2 \mid p_2 \in T_{P_2}^{in|out} \wedge pre_1(t_2)_{|p_2} > 0\}| \\ &\quad\wedge |\{p_1 \mid p_1 \in T_{P_1}^{in|out} \wedge post_1(t_1)_{|p_1} > 0\}| = |\{p_2 \mid p_2 \in T_{P_2}^{in|out} \wedge post_2(t_2)_{|p_2} > 0\}| \\ &rule_{new,P}(s_n,p_1,p_2) \\ &\quad\equiv |\{t_1 \mid t_1 \in \widetilde{T_1} \wedge post_1(t_1)_{|p_1} > 0\}| \leq |\{t_2 \mid t_2 \in \widetilde{T_2} \wedge post_2(t_2)_{|p_2} > 0\}| \\ &\quad\wedge |\{t_1 \mid t_1 \in \widetilde{T_1} \wedge pre_1(t_1)_{|p_1} > 0\}| \leq |\{t_2 \mid t_2 \in \widetilde{T_2} \wedge pre_2(t_2)_{|p_2} > 0\}| \\ &rule_{new,T}(s_n,t_1,t_2) \\ &\quad\equiv |\{p_1 \mid p_1 \in \widetilde{P_1} \wedge pre_1(t_1)_{|p_1} > 0\}| = |\{p_2 \mid p_2 \in \widetilde{P_2} \wedge pre_1(t_2)_{|p_2} > 0\}| \\ &\quad\wedge |\{p_1 \mid p_1 \in \widetilde{P_1} \wedge post_1(t_1)_{|p_1} > 0\}| = |\{p_2 \mid p_2 \in \widetilde{P_2} \wedge post_2(t_2)_{|p_2} > 0\}| \end{aligned} \tag{4}$$

Accordingly, feasibility of pairs of places is given by

$$feasible_P \equiv rule_{sem,P} \wedge rule_{pred,P} \wedge rule_{succ,P} \wedge rule_{in,P} \wedge rule_{out,P} \wedge rule_{new,P} \tag{5}$$

and feasibility of pairs of transitions is given by

$$feasible_T \equiv rule_{sem,T} \wedge rule_{pred,T} \wedge rule_{succ,T} \wedge rule_{in,T} \wedge rule_{out,T} \wedge rule_{new,T} \quad (6)$$

Looking at the example (in Subsect. 4) the pair (t_1, t_b) cannot be considered for the match since $feasible_T(s_1, t_1, t_b) \equiv false$. This results from $rule_{sem,T}$ because t_1 has less incoming and outgoings arcs than t_b.

The representation of the state uses six integer arrays for each net N_j, namely $core_{j,X}, in_{j,X}, out_{j,X}$ for $j \in \{1,2\}$ and $X \in \{P,T\}$. These arrays are shared among all states, so if the algorithm backtracks, it restores the previous value for these arrays. The elements of these arrays are ordered according to the (arbitrary, but fixed during initialization) order of the nodes, i.e., the second element of $core_{1,P}$ refers to the second place of net N_1 and the third element of $in_{2,T}$ refers to the third transition in net N_2.

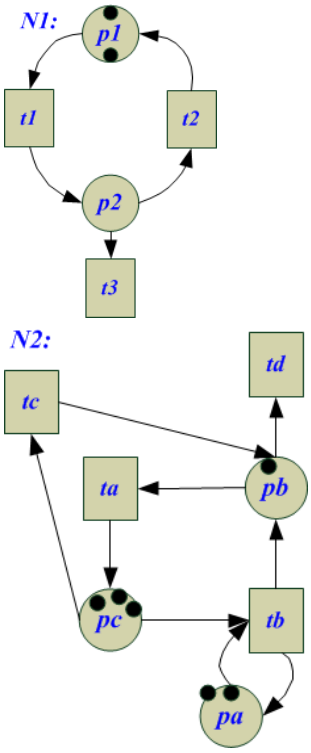
In Subsect. 4 the six core array are presented by tables for different states in Fig. 4. In state s_1 the array $core_1$ and the array $core_2$ are presented in the second row. The order of the nodes in both nets is given by the implicit order of their indices. So, p_2 of net N_1 is represented in the third column, and t_d of net N_2 in the last column.

The core arrays are used for the mappings from the perspective of each net. The elements position in the core array refers to the position of the node in the corresponding net, whereas the value refers to the mapped node in the other net. The elements of $core_1$ and $core_2$ point mutually to each other. The *in* and *out* arrays manage the terminal and residual node sets. Adding a pair of nodes to the current state changes the values of *in* and *out* for the neighboring nodes. The value coincides with the recursion depth, in which a node enters the terminal node set. If a pair of nodes is added to the state only those fields are set to the current depth that do not have a value. The combination of *core*, *in* and *out* arrays, the terminal and residual node sets can be computed.

In state $s2$ (see Subsect. 3.2) the mapping of p_1 in net N_1 to the place p_c in net N_2 leads to $core_{1,P}[1] = 3$ and $core_{2,P}[3] = 1$ and the mapping of t_1 to t_c leads to $core_{1,T}[1] = 3$ and $core_{2,T}[3] = 1$.

3.2 Example

For the nets given in Fig. 4, the empty match of the first state s_0 is given by $M(s_0) = \emptyset$ and $P(s_0) = P_{\widetilde{P}}(s_0) = \{(p_1,p_a),(p_1,p_b),(p_1,p_c)\}$. Hence, the arrays are empty as well. According to the predefined order (given by the obvious order of the indices) the candidate pair (p_1, p_a) is chosen. As $feasible_P(s_0, p_1, p_a)$ holds, $M(s_1) = \{(p_1, p_a)\}$ with the corresponding core, in and out arrays for state s_1 in Fig. 4. Then $P(s_1) = P_T(s_1) = \{(t_1, t_b)\}$ is computed, but $feasible_T(s_1, t_1, t_b) \equiv false \wedge \cdots \equiv false$. So, the algorithm backtracks and computes a new state s_1 based on $P(s_0) = P_P(s_0) = \{(p_1,p_b),(p_1,p_c)\}$ leading to $M(s_1) = \{(p_1,p_c)\}$ and the arrays of state s_1 in Fig. 4. Next we obtain $P(s_1) = P_T(s_1) = \{(t_1,t_b),(t_1,t_c)\}$ with $feasible_T(s_1,t_1,t_b) \equiv false$ and $feasible_T(s_1,t_1,t_c) \equiv true$. The next recursion step yields $M(s_2) = \{(p_1,p_c),(t_1,t_c)\}$ and the arrays of state s_2. Then $P(s_2) = P_P(s_2) = \{(p_2,p_b)\}$ and $feasible_P(s_2,p_2,p_b) \equiv true$. This leads to $M(s_3) = \{(p_1,p_c),(t_1,t_c),(p_2,p_b)\}$ and the candidate pairs $P(s_3) = \{(t_2,t_a),(t_2,t_b),(t_2,t_d)\}$ and $feasible_T(s_3,t_2,t_a)$ holds. There is no more backtracking necessary and the last state is s_5 with $M(s_5) = \{(p_1,p_c),(t_1,t_c),(p_2,p_b),(t_2,t_a),(t_3,t_d)\}$ being an injective net morphism. The corresponding arrays for state s_5 are given in Fig. 4.



s_1

	$N1$					$N2$						
	P		T				P		T			
core1	1					core2	1					
in1	1			1		in2	1				1	
out1	1		1			out2	1				1	

new s_1

	$N1$					$N2$						
	P		T				P		T			
core1	3					core2			1			
in1	1			1		in2			1	1		
out1	1		1			out2			1		1	1

s_2

	$N1$					$N2$						
	P		T				P		T			
core1	3		3			core2			1			1
in1	1		2	1		in2			1	1		2
out1	1	2	1			out2		2	1		1	1

s_5

	$N1$					$N2$						
	P		T				P		T			
core1	3	2	3	1	4	core2		2	1	2		1	3
in1	1	3	2	1	5	in2		3	1	1	3	2	5
out1	1	2	1	3	3	out2		2	1	4	1	1	3

Figure 4: Nets $N1$ and $N2$ and some states

4 Evaluation of PNVF2

4.1 Correctness

Both the VF2 and the PNVF2 guarantee the correctness of the found matches. A match is considered to be correct if the structure of the source net and its annotations are preserved. And the match has to be an injective morphism. To satisfy these conditions, the algorithms examine the candidates before they add them to the respective part of the match. The candidate selection during the recursive descent makes sure that the mapping is injective. And the function $feasible$ ensures structural and semantic compatibility. The correctness of PNVF2 is stated explicitly in Theorem 1. For Lemma 1 we need at each state the corresponding partial nets[4]. In these partial nets the sets of places and the sets of transitions are restricted to the matched places and transitions. Given $j \in \{1,2\}$ and $0 \leq i \leq |P_1| + |T_1|$ we define $Q_{j,i}$ with places $P_j \cap M_j(s_i)$ and transitions $T_j \cap M_j(s_i)$, and the pre- and post-domain are given by $pre_{j|T_j \cap M_j(s_i)}$ and $post_{j|T_j \cap M_j(s_i)}$ and the other functions accordingly. Obviously, the partial nets $Q_{j,i}$ are well-defined. Then each state computed by PNVF2 leads to an injective net morphism between the corresponding partial nets.

[4] Note, that these nets are subgraphs but there are not necessarily net morphisms form $Q_{j,i}$ to N_j.

Lemma 1 (Correctness of $feasible$) *Let $1 \leq i \leq n = |P_1| + |T_1|$ be given.*

1. *If $M(s_{i-1}) : Q_{1,i-1} \to Q_{2,i-1}$ is an injective net morphism and $feasible(s_{i-1}, v, w)$ holds for $(v,w) \in P(s_{i-1})$ and $M(s_i) = M(s_{i-1}) \cup \{(v,w)\}$, then $M(s_i) : Q_{1,i} \to Q_{2,i}$ is an injective net morphism as well.*
2. *If $feasible(s_i, v, w) = false$, there is no injective net morphism $f : N_1 \to N_2$ with $f_Q := f_{|Q_{1,i+1}} : Q_{1,i+1} \to Q_{2,i+1}$, where $Q_{j,i+1}$ are the partial nets induced by the match $M(s_{i+1}) = M(s_i) \cup \{(v,w)\}$.*

Proof sketch

1. *Since $M(s_{i-1}) : Q_{1,i-1} \to Q_{2,i-1}$ is an injective net morphism, it is an injective mapping of places and an injective mapping of transitions. The addition of a pair of places (or transitions) that are not in the current sets of places (or transitions), yields again an injective mapping. For $(v,w) \in P_1 \times P_2$ $feasible_P(s_{i-1}, v, w)$ holds (equation 5), hence the marking is preserved (rule 2). Moreover, the net structure is preserved (rule 3). Similarily, for $(v,w) \in T_1 \times T_2$ $feasible_T(s_{i-1}, v, w)$ holds (equation 6), hence the number of in- and outgoing arcs of both transitions are the same. Moreover, the net structure is preserved (rule 3). Hence, $M(s_i) : Q_{1,i} \to Q_{2,i}$ is an injective net morphism.*
2. *Failure of $feasible(s_i, v, w)$ (equation 5) implies that there is no injective morphisms with $f_Q : Q_{1,i+1} \to Q_{2,i+1}$, and hence no $f : N_1 \to N_2$ with $f_{|Q_{1,i+1}} = f_Q$.*

This lemma allows us to deduce that the result of PNVF2 is an injective net morphism.

Theorem 1 (PNVF2 is correct and complete.) *Let be given two place/transition nets N_j with $j \in \{1,2\}$ and $n = |P_1| + |T_1|$. PNVF2 yields $M(s_n)$ if and only if there exists an injective net morphism from $f : N_1 \to N_2$.*

Proof sketch

if: *If the algorithm yields $M(s_n)$ then $f = M(s_n) : N_1 \to N_2$ is an injective net morphism is shown by induction over $n = |P_1| + |T_1|$ the size of N_1.*

Base *$M(s_0) = \emptyset$ is an injective net morphism for N_1 being the empty net.*

Hyp. *If PNVF2 yields $M(s_{n-1})$ for a net N_1 of size $n-1 = |P_1| + |T_1|$, then each $M(s_i) : Q_{1,i} \to Q_{2,i}$ is an injective net morphism for $i \leq n-1$.*

Step *For for a net N_1 of size $n = |P_1'| + |T_1'|$ we have two cases*

$|P_1'| = |P_1| + 1$ and $|T_1'| = |T_1|$. Since PNVF2 yields $M(s_n)$ and by induction hypothesis there is $M(s_{n-1}) : Q_{1,n-1} \to Q_{2,n-1}$ an injective net morphism. Since PNVF2 yields $M(s_n)$, there is some $p_1', p_2' \in P(s_{n-1})$ so that $feasible_P(s_{n-1}, p_1', p_2')$ holds. We have then $M(s_n) : Q_{1,n} \to Q_{2,n} = M(s_n) : N_1 \to N_2$ and due to Lemma 1 this is an injective net morphism.

$|T_1'| = |T_1| + 1$ and $|P_1'| = |P_1|$ analogously.

only if: *If PNVF2 does not yield $M(s_n)$, then there is a step in each recursion path, where $feasible(s_i, v, w)$ fails and due to Lemma 1 there is no injective net morphism $f : N_1 \to N_2$.*

4.2 Non-Determinism

The non-determinism of the matches is paramount for the simulation of reconfigurable nets, since the full potential behavior should be captured. In a situation where several different matches are possible, the algorithm has to deliver different matches for different runs. After sufficiently many runs the algorithm must have delivered each of the possible matches. A very inefficient strategy would be to compute first all possible mappings and to choose then one non-deterministically. Instead we have realized non-deterministic choice at each possible step of the algorithm. Implementing non-determinism at every level results in mapping source nodes randomly to target nodes. However, this strategy has also the disadvantage that it is very inefficient, as choosing random pairs the algorithm ignores the structure of the net to a large extent. This causes the matching to fail more often in larger depths of the recursion. The VF2 follows the strategy to map adjacent components to the destination net. For this purpose nodes of the terminal sets are preferred. For the sake of reasonable runtime of the PNVF2, this strategy is maintained. The VF2 algorithm requires a fixed order on the the source nodes and an order on the target nodes. This order determines the sequence in which nodes of the corresponding sets are processed and it ensures that the algorithm does not generate the same states via different execution paths. Any possible order of the nodes is sufficient, as long as it remains stable during the run. We use this fact as the main realization of non-determinism by permuting all places and transitions of both nets at each start of the algorithm. Another, but less influential, implementation of non-determinism is the random choice of the node type (i.e., choosing places or transitions) at each level of the recursion.

In order to prove that the algorithm is non-deterministic, i.e., it can compute any injective net morphism between two given nets, we need to state an order of the places and transitions that leads to a given injective net morphism. Then this order is compatible with the given morphism.

Lemma 2 (Compatible order) *Let an injective net morphism $f : N_1 \to N_2$ be given. For orders $\rho = (\rho_{X_j}) : X_j \to \{1,...,|X_j|\}$ with $X \in \{P,T\}$ and $j \in \{1,2\}$ with $\rho_{P_1}(p) = \rho_{P_2}(f_P(p))$ and $\rho_{T_1}(t) = \rho_{T_2}(f_T(t))$ we have for all $0 \leq i \leq n = |P_1| + |T_1|$ and all $(v,w) \in M(s_i)$:*

There exists a match $M(s_i)$, so that we have $\rho_{X_1}(v) = \rho_{X_2}(w)$

If we need not to differentiate between places and transitions we use ρ_i instead of ρ_{X_i}.

Proof sketch *Induction over i for $0 \leq i \leq n = |P_1| + |T_1|$:*

Base *At state s_0 we have $M(s_0) = \emptyset$.*

Step *By induction hypothesis we have $M(s_i)$ and $\rho_1(v) = \rho_2(w)$ for all $(v,w) \in M(s_i)$.*

Then there is minimal $\rho_1(x)$ with $\rho_1(x) = \min X_1$ and $X_1 \in \{T^{in|out}_{P_1|T_1}, \widetilde{T_1}, \widetilde{P_1}\}$.

Since $f(x) = \rho_2^{-1}(\rho_1(x))$ and since the terminal and residual nodes sets are constructed similarly, we have that $x \in X_1$ implies $\rho_2^{-1}(\rho_1(x)) \in X_2$ for $X_j \in \{T^{in|out}_{P_j|T_j}, \widetilde{T_j}, \widetilde{P_j} \mid j \in \{1,2\}\}$.

So, there is one candidate pair $(x, \rho_2^{-1}(\rho_1(x))) \in P(s_i)$.

As f is an injective net morphism, the rules $rule_{sem}$, $rule_{pred}$ and $rule_{succ}$ hold (see rules 2 and 3). $rule_{in}$, $rule_{out}$ and $rule_{new}$ (see rule 4) hold, because the neighboring nodes can be mapped, as f is an injective net morphism. Hence $(feasible(s_i, x, \rho_2^{-1}(\rho_1(x))))$ holds.

Last, let $M(s_{i+1}) = M(s_i) \cup (x, \rho_2^{-1}(\rho_1(x)))$.

Theorem 2 (PNVF2 is non-deterministic.) *Given* $f : N_1 \to N_2$ *an injective net morphism, then PNVF2 can yield* $M(s_n) : N_1 \to N_2$*, so that* $M(s_n) = f$.

From Lemma 2 we directly obtain a match $M(s_n) = f$.

4.3 Complexity

One of the most important properties of an algorithm is its complexity. In [CFSV04b] the space and time complexity of VF2 has been given for graph isomorphisms by $\Theta(N)$ and $\Theta(N^2)$ for the best case and by $\Theta(N)$ and $\Theta(N!N)$ for the worst case on the number of nodes N. In principle we have the same complexity measures. Nevertheless, for a more precise investigation of the complexity differences of both algorithms the complexity measures have been computed explicitly for the subgraph isomorphisms in [Blu13]. In the following we assume that the execution of an elementary operation, as the reading and setting of the arrays, is done in $\Theta(1)$. Time complexity of the VF2 is based on the number of nodes of the source graph n and of the target graph m, and time complexity of the PNVF2 is based on the number of places p_1 and transitions t_1 of the source net and of the target net p_2 and t_2. For the PNVF2 the best case occurs if two equally sized nets without any arcs are considered. Due to the lack of arcs any mapping of places (resp. transitions) is a valid match. The worst case occurs if there are two almost complete nets whose search area needs to be investigated completely. In contrast to the best case, the amount of nodes in the target can be significantly higher than in the source.

The analysis of the space complexity refers only to the data structures of the algorithms. As a representation of the current matching state, both algorithms use the same instance at each recursion level, namely the arrays core, in and out. Therefore the memory is allocated only once. For each graphs three arrays of length n and m are managed. And the three arrays for each of the nets have the length $p_1 + t_1$ and $p_2 + t_2$. Due to the reduced search space, as nets are bipartite graphs, we can expect the complexity results to be slightly better. Comparing the complexity measures, we consider a net as a graph, and hence have $n = p_1 + t_1$ and $m = p_2 + t_2$. Then we can rate

$$p_2 t_2 \cdot \binom{p_2}{p_1} \cdot p_1! \cdot \binom{t_2}{t_1} \cdot t_1! \leq (p_2+t_2)^2 \cdot p_1! \cdot t_1! \cdot \binom{p_2}{p_1} \cdot \binom{t_2}{t_1} \leq (p_2+t_2)^2 \cdot \binom{p_2}{p_1} \cdot \binom{t_2}{t_1} \cdot (p_1+t_1)! \leq m^2 \cdot \binom{m}{n} n!$$

So, we obtain $T_{PNVF2}(p1+t1, p2+t2) \in O(N! \cdot N)$, since we have by induction $nm^2 \cdot \binom{m}{n} n! \leq NN!$, and hence $p_2 t_2 \cdot \binom{p_2}{p_1} \cdot p_1! \cdot \binom{t_2}{t_1} \cdot t_1! \leq m^2 \cdot \binom{m}{n} \cdot n! \leq N \cdot N!$ for $n = p1 + t1$, $m = p2 + t2$ and for $N = n + m$.

5 Related Work and Conclusion

Obviously, reconfigurable Petri nets are closely related to graph transformations (see [MEE10]). AGG [AGG13] and its derivations as the RON-editor [BEMS08] and CPEditor [Mod12], translate Petri nets into attributed graphs and net rules to attributed graph rules. But they do not consider directly Petri nets, and hence do not support the separation of dynamics. An extensive

discussion of various matching algorithms has been presented in [CFSV04a] comprising exact as well as inexact algorithms. Due to the nature of the requirements we have investigated merely exact algorithms [MB00, Ull76, DZ09]. For more details see [Blu13]. The extension the VF2 algorithm by domain-specific information has been in order to speed up the pattern matching process. These search plan driven graph pattern matching techniques have been investigated in [GHS09, GBG+06, GSR05]. The Ullmann algorithm has been the basis of the ReConNet's previous implementation, that was faulty with respect to the non-determinism and moreover did not provide means for the realization of negative application conditions, whereas, the PNVF2 has been adapted accordingly.

Ongoing work concerns model checking by translating nets and rules into rewrite logic Maude and an explicit representation of an abstract reachability graph based on [Pad12]. Future work includes the investigation of the runtime behavior and memory consumption of the implemented algorithm and an experimental comparison with the VF2 algorithm to evaluate the implementation. Another fruitful extension of ReConNet are control structures, as the extension of rules with negative application conditions or the introduction of transformation units.

Acknowledgements: We are grateful to the referees for their valuable remarks.

References

[AGG13] AGG. The Attributed Graph Grammar System. 2013. Revision: 02/10/2013 15:25:28. http://user.cs.tu-berlin.de/~gragra/agg/

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

[Blu13] M. Blumreiter. Algorithmus zum nichtdeterministischen Matching in rekonfigurierbaren Petrinetzen. Bachelor's thesis, Hamburg University of Applied Sciences, Germany, 2013.

[CFSV04a] D. Conte, P. Foggia, C. Sansone, M. Vento. Thirty Years Of Graph Matching In Pattern Recognition. *International Journal of Pattern Recognition and Artificial Intelligence* 18(3):265–298, 2004.

[CFSV04b] L. P. Cordella, P. Foggia, C. Sansone, M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 26(10):1367 – 1372, oct. 2004.

[DZ09] Y.-T. Dai, S.-Y. Zhang. A fast labeled graph matching algorithm based on edge matching and guided by search route. In *International Conference on Wavelet Analysis and Pattern Recognition*. Pp. 1–7. 2009.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in TCS. Springer, 2006.

[EHOP12] M. Ede, K. Hoffmann, G. Oelker, J. Padberg. ReConNet: A Tool for Modeling and Simulating with Reconfigurable Place/Transition Nets. In Krause and Westfechtel (eds.), *Proc. of the Seventh International Workshop on Graph-Based Tools*. Volume 54. Electronic Communications of the EASST, 2012.

[EP03] H. Ehrig, J. Padberg. Graph Grammars and Petri Net Transformations. In Desel et al. (eds.), *Lectures on Concurrency and Petri Nets*. Lecture Notes in Computer Science 3098, pp. 496–536. Springer, 2003.

[Gab14] K. Gabriel. *Interaction on Human-Centric Communication Platforms: Modelling and Analysis using Algebraic High-Level Nets and Processes*. PhD thesis, Technische Universität Berlin, 2014.

[GBG+06] R. Geiß, G. V. Batz, D. Grund, S. Hack, A. Szalkowski. GrGen: A Fast SPO-Based Graph Rewriting Tool. In Corradini et al. (eds.), *Third International Conference on Graph Transformations*. Lecture Notes in Computer Science 4178, pp. 383–397. Springer, 2006.

[GHS09] H. Giese, S. Hildebrandt, A. Seibel. Improved Flexibility and Scalability by Interpreting Story Diagrams. *ECEASST* 18, 2009.

[GSR05] L. Geiger, C. Schneider, C. Reckord. Template- and model-based code generation for MDA-tools. In Giese and Zündorf (eds.), *Proceedings of the 3rd International Fujaba Days*, pp. 57–62. 2005.

[KCD10] L. Kahloul, A. Chaoui, K. Djouani. Modeling and Analysis of Reconfigurable Systems Using Flexible Petri Nets. In *4th IEEE International Symposium on Theoretical Aspects of Software Engineering*. Pp. 107–116. 2010.

[LO04] M. Llorens, J. Oliver. Structural and Dynamic Changes in Concurrent Systems: Reconfigurable Petri Nets. *IEEE Trans. Computers* 53(9):1147–1158, 2004.

[MB00] B. T. Messmer, H. Bunke. Efficient subgraph isomorphism detection: a decomposition approach. *Knowledge and Data Engineering, IEEE Transactions on* 12(2):307–323, 2000.

[MEE10] M. Maximova, H. Ehrig, C. Ermel. Formal Relationship between Petri Net and Graph Transformation Systems based on Functors between M-adhesive Categories. *ECEASST* 40, 2010.

[Mod12] T. Modica. *Formal Modeling, Simulation, and Validation of Communication Platforms*. PhD thesis, Technical University of Berlin, 2012.

[Pad12] J. Padberg. Abstract Interleaving Semantics for Reconfigurable Petri Nets. *ECEASST* 51, 2012.

[Rei12] F. Reiter. Modellierung und Analyse von Szenarien des Living Place mit rekonfigurierbaren Petrinetzen. Bachelor Thesis, Hochschule für Angewandte Wissenschaften Hamburg, 2012.

[Ull76] J. R. Ullmann. An Algorithm for Subgraph Isomorphism. *J. ACM* 23(1):31–42, Jan. 1976.