Martin Schwarick, Monika Heiner (Hrsg.)

Algorithmen und Werkzeuge für Petrinetze

17ter Workshop, AWPN 2010
Cottbus, 07.-08.Oktober 2010

Proceedings

Preprints CS-02-10

Herausgeber:

Martin Schwarick, Monika Heiner
Brandenburgische Technische Universität Cottbus, Institut für Informatik,
03013 Cottbus, Germany
{ms, monika.heiner}@informatik.tu-cottbus.de

ISSN 1437-7969 (Preprint CS-02-10 BTU Cottbus)
ISSN 1613-0073 (CEUR Workshop Proceedings)

Online-Proceedings verfügbar unter http://CEUR-WS.org/Vol-643/

BibTeX-Eintrag für Online-Proceedings:

```
@proceedings{awpn2010,
  editor    = {Martin Schwarick and Monika Heiner},
  title     = {Proceedings of the 17th German Workshop on
               Algorithms and Tools for Petri Nets, AWPN 2010,
               Cottbus, Germany, October 07--08, 2010},
  booktitle = {Algorithmen und Werkzeuge f\"ur Petrinetze},
  publisher = {CEUR-WS.org},
  series    = {CEUR Workshop Proceedings},
  volume    = {643},
  year      = {2010},
  url       = {http://CEUR-WS.org/Vol-643/}
}
```

Vorwort

Der Workshop *Algorithmen und Werkzeuge für Petrinetze* bietet seit 1994 ein gemeinsames Forum für Entwickler und Anwender petrinetz-basierter Technologien. Er bietet insbesondere für Nachwuchswissenschaftler die Möglichkeit, Erfahrungen bei einer wissenschaftlichen Veranstaltung zu sammeln. Das wird unterstützt durch den traditionell geringen finanziellen Aufwands für die Teilnahme und die deutschsprachige Ausrichtung, wobei auch englischsprachige Beiträge willkommen sind.

Im Jahr 2010 fand der Workshop in seiner 17ten Ausgabe erstmals an der Brandenburgischen Technischen Universität in Cottbus statt. Veranstalter war wie immer die Fachgruppe *Petrinetze und verwandte Systemmodelle* der Gesellschaft für Informatik.

Es gab 19 eingereichte Beiträge, die alle nach einer kurzen Prüfung auf sachliche Angemessenheit durch die Organisatoren in das Programm aufgenommen wurden. Ein ausführlicher Begutachtungsprozess fand dagegen, wie auch in den vergangenen Jahren, nicht statt.

Oktober 2010

Martin Schwarick
Monika Heiner

Steering Committee

Jörg Desel (Stellvertreter)	FernUniversität Hagen
Ekkart Kindler	Technical University of Denmark
Kurt Lautenbach	Universität Koblenz-Landau
Robert Lorenz	Universität Augsburg
Daniel Moldt	Universität Hamburg
Rüdiger Valk	Universität Hamburg
Karsten Wolf (Sprecher)	Universität Rostock

Bisherige AWPN-Workshops

1. Berlin 1994
2. Oldenburg 1995
3. Karlsruhe 1996
4. Berlin 1997
5. Dortmund 1998
6. Frankfurt 1999
7. Koblenz 2000
8. Eichstätt 2001
9. Potsdam 2002
10. Eichstätt 2003
11. Paderborn 2004
12. Berlin 2005
13. Hamburg 2006
14. Koblenz 2007
15. Rostock 2008
16. Karlsruhe 2009
17. Cottbus 2010

Inhaltsverzeichnis

Universal Inhibitor Petri Net 1
Dmitry Zaitsev

On Optimizing the Sweep-Line Method 16
Robert Prüfer

Embedding the free-choice semantics of AND/XOR-EPCs into the Boolean semantics 22
Christoph Schneider and Joachim Wehler

Modular and Hierarchical Modelling Concept for Large Biological Petri Nets Applied to Nociception 42
Mary Ann Blätke and Wolfgang Marvan

Computation of enabled transition instances for colored Petri nets 51
Fei Liu and Monika Heiner

Hybrid Petri Nets for Modelling of Hybrid Biochemical Interactions 66
Mostafa Herajy and Monika Heiner

IDD-MC - a model checker for bounded Stochastic Petri nets 80
Martin Schwarick

Simulative CSL model checking of Stochastic Petri nets in IDD-MC 88
Christian Rohr

Re-Thinking Process Mining with Agents in Mind 94
Nils Erik Flick, Lawrence Cabac, Nicolas Denz, and Daniel Moldt

Helper Agents as a Means of Structuring Multi-Agent Applications 100
Kolja Markwardt and Daniel Moldt

PyTri, a Visual Agent Programming Language 106
Jochen Simon and Daniel Moldt

Optimised Calculation of Symmetries for State Space Reduction 112
Harro Wimmel

Reachability Analysis via Net Structure 118
Harro Wimmel and Karsten Wolf

Decidability Issues for Decentralized Controllability of Open Nets 124
Karsten Wolf

On the notion of deadlocks in open nets 130
Richard Müller

A graphical user interface for service adaptation 136
Christian Gierds and Niels Lohmann

Managing test suites for services 142
Kathrin Kaschner

The Petri Net API A collection of Petri net-related functions 148
Niels Lohmann, Stephan Mennicke, and Christian Sura

Partner datenverarbeitender Services 154
Christoph Wagner

Autorenverzeichnis 160

Universal Inhibitor Petri Net

Dmitry Zaitsev

International Humanitarian University
Department of Information Technology
Fontanskaya Doroga st., 33, Odessa 65009, Ukraine
http://member.acm.org/~daze

Abstract. The universal inhibitor Petri net was constructed that executes an arbitrary given inhibitor Petri net. The inhibitor Petri net graph, its marking and the transitions firing sequence were encoded as 10 scalar nonnegative integer numbers and represented by corresponding places of universal net. The algorithm of inhibitor net executing that uses scalar variables only was constructed on its state equation and encoded by universal inhibitor Petri net. Subnets which implement arithmetic, comparison and copying operations were employed.

Keywords: universal inhibitor Petri net, universal Turing machine, encoding, algorithm

1 Introduction

It is known, that inhibitor, synchronous, priority and other extended Petri net classes constitute a universal algorithmic system [1,2]. For such universal algorithmic systems as Turing machines, there are known examples of universal Turing machine construction [3]. In this connection it is of a definite interest the construction of a universal Petri net which executes an arbitrary given Petri net that is the goal of the present paper.

2 The Concept of a Universal Petri Net

The universal net is constructed in the class of inhibitor Petri nets [1,2], the corresponding universal inhibitor Petri net is denoted as UIPN. Considering nondeterministic character of Petri net dynamics the most close analog is nondeterministic Turing machine [3].

As it is of interest the constructing of the universal Petri net with a fixed structure, the only way of input and output information representation is the marking of a fixed number of definite UIPN places. Therefore, it is necessary to give rules of a biunique encoding of Petri net graph and its marking by a fixed quantity of nonnegative integer numbers. Let there are given the corresponding encoding rules and sXIPN is the code of Petri net XIPN graph and sQXIPN is the code of the marking QXIPN.

The concept of reachable marking in Petri net implies the existence of the corresponding enabled sequence of transitions firing [1,2]. But the usage of only marking QXIPN in the definition of UIPN does not guarantee the obtaining of all the enabled sequences of transitions firing of the net XIPN. Let definite rules of the transitions firing sequences encoding are given and sZQXIPN is a code of the enabled transitions firing sequence ZQXIPN which moves Petri net XIPN from marking Q0XIPN to marking QXIPN. Then the functioning of UIPN can be represented as the scheme shown in fig. 1.

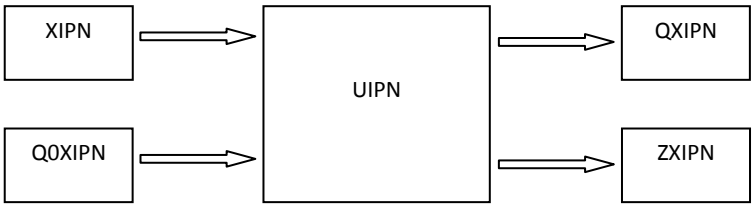


Fig. 1. The scheme of universal inhibitor Petri net UIPN functioning.

Definition 1. Petri net UIPN is a universal inhibitor Petri net if and only if for an arbitrary given inhibitor Petri net XIPN and its initial marking Q0XIPN the net UIPN stops in the marking (sQXIPN,sZQXIPN), where marking QXIPN is reachable in XIPN with the transitions firing sequence ZQXIPN and any marking (sQXIPN,sZQXIPN) which UIPN stops in is a code of a marking QXIPN reachable in XIPN from initial marking Q0XIPN with the transition firing sequence ZQXIPN.

The requirement of the UIPN stopping possibility even in case of a nondead marking QXIPN of the net XIPN is connected with the provisioning the checkpoint (observance) of any reachable marking (and transitions firing sequence) and abstracting from the implementation of UIPN; otherwise it is necessary to add some extra restrictions for the exclusion out of the observance the intermediate markings of UIPN.

3 Formal Representation of Inhibitor Petri Net

Graph of inhibitor Petri net [1,2] is a four-tuple $G = (P, T, B, D)$, where $P = \{p_1, .., p_m\}$ is a finite number of nodes named places, $T = \{t_1, .., t_n\}$ is a finite number of nodes named transitions and the mappings $B: P \times T \to \mathbb{N} \cup \{-1\}$ and $D: T \times P \to \mathbb{N}$ define the input and output arcs of transitions correspondingly together with their multiplicity, $\mathbb{N}$ is the set of nonnegative integer numbers; zero value of mappings B, D denote the absence of the arc, nonzero – the arc multiplicity, the special value -1 denotes the inhibitor arc. The mappings can be represented by the corresponding matrices: $B = \|b_{i,j}\|, b_{i,j} = B(p_j, t_i)$ и $B = \|d_{i,j}\|, d_{i,j} = D(t_i, p_j)$.

The state of net is named a marking and represented by the mapping $Q: P \to \mathbb{N}$, that gives the number of dynamic elements – tokens within places of net. Inhibitor Petri net [1,2] is a couple $N = (G, Q_0)$, where G is the net graph and Q_0 – its initial marking. The marking can be represented by the corresponding vector: $Q =$

$\|q_j\|, q_j = Q(p_j)$. Thus, the inhibitor Petri net is given by the pair of numbers, pair of matrices and a vector: N=(m, n, B, D, Q_0).

The dynamics of inhibitor net constitutes a step-by-step process of its marking transformation as a result of transitions firing [1,2] and can be formally represented by the following system:

$$\begin{cases} q_j^k = q_j^{k-1} - x(b_{l,j}) + d_{l,j}, j = \overline{1,m} \\ u(t_i) = \bigwedge\limits_{j=1,m} ((y(b_{i,j}) \wedge (q_j^{k-1} = 0)) \vee (\bar{y}(b_{i,j}) \wedge (q_j^{k-1} \geq b_{i,j})), i = \overline{1,n} \\ u(t_l) = 1, l \in \overline{1,n} \\ k = 1,2,\ldots \\ x(b) = \begin{cases} b, b \geq 0 \\ 0, b = -1 \end{cases}, y(b) = \begin{cases} 0, b \geq 0 \\ 1, b = -1 \end{cases}. \end{cases} \quad (1)$$

The first line of the system (1) describes the marking transformation at the transition t_l firing; the function $u(t_i)$ in the second line defines the transition t_i enabling condition at the current step k, the third line defines nondeterministic choice of the firing transition t_l out of the set of enabled transitions, the fourth line gives the order of steps sequence; auxiliary mappings x and y serve for defining the marking decrement and inhibitor arc recognition respectively.

4 Encoding of Inhibitor Petri Net

In the present section a representation of encoding of inhibitor Petri net, its current marking and corresponding transitions firing sequence is obtained in the form of the marking of 10 special places of universal net UIPN (fig. 2). The examples of nets encoding are shown in Appendix A.

4.1 Encoding of a Vector

Let Q is a vector (line), containing m nonnegative integer elements; suppose that elements indexing is started from zero. Let also the following value is calculated

$$r = \max_j q_j + 1.$$

The vector encoding function is defined as

$$s = \varphi(Q) = \sum_{j=0}^{m-1} r^j \cdot q_j.$$

Statement 1. The vector encoding function is injective.

The corresponding decoding function is represented as

$$q_i = \psi(s, j) = (s \; div \; r^j) \; mod \; r.$$

Inherently, the defined encoding is the form of numbers representation in the radix notation with the radix r.

The encoding can be implemented recursively

$$s_j = s_{j-1} \cdot r + q_{m-1-j}, s_0 = q_{m-1}, j = \overline{1, m-1}, \tag{2}$$

where the code of the vector Q equals to $s = s_{m-1}$.

The decoding can be implemented recursively also

$$q_j = s_{m-1-j} \bmod r, s_{m-1-(j+1)} = s_{m-1-j} \ div\ r, s_{m-1} = s, \quad j = \overline{0, m-1}. \tag{3}$$

4.2 Encoding of a Matrix

Let A is a $n \times m$ matrix with nonnegative integer values of elements; suppose that elements indexing is started from zero. Let also the following value is calculated

$$r = \max_{i,j} a_{i,j} + 1.$$

While encoding, let us represent the matrix as a vector with the expansion on lines. Then the matrix A is encoded as

$$s = \varphi(A) = \sum_{i=0}^{n} \sum_{j=0}^{m} r^{(m \cdot i + j)} \cdot a_{i,j.}$$

Statement 1. The matrix encoding function is injective.

The corresponding decoding function is represented as

$$a_{i,j} = \psi(A, i, j) = \left(s \ div\ r^{(m \cdot i + j)}\right) mod\ r.$$

The encoding can be implemented recursively

$$s_v = s_{v-1} \cdot r + a_{i,j}, s_0 = a_{n-1,m-1}, \quad v = \overline{1, n \cdot m - 1}, i = v\ div\ n, j = v\ mod\ n, \tag{4}$$

where the code of the matrix A equals to $s = s_{n \cdot m-1}$.

The decoding can be implemented recursively also

$$a_{i,j} = s_{n \cdot m-1-v} \bmod r, s_{n \cdot m-1-(v+1)} = s_{n \cdot m-1-v} \ div\ r, s_{n \cdot m-1} = s, \quad v = \overline{0, n \cdot m - 1}, i = v\ div\ n, j = v\ mod\ n. \tag{5}$$

4.3 Encoding of Inhibitor Petri Net Graph

The graph is represented by the pair of matrices B and D. Usually the zero value of the matrix element indicates the absence of the corresponding arc, nonzero – its multiplicity. The representation of inhibitor arcs of matrix B require supplementary agreements to avoid negative values. Let k is the multiplicity of arc, then for its representation the value $k + 1$ is used; the value of 1 is reserved for the inhibitor arc representation.

It is reasonable the separate encoding according to (4) and storing in separate places the codes of matrices B and D, as well as the corresponding values of r. For the storing of the encoded Petri net graph, 6 corresponding places with names m, n, sB, rB, sD, rD are used shown in fig. 2 which marking contains the values m, n, sB, rB, sD, rD respectively.

4.4 Encoding of Marking

The marking of a Petri net containing m places is given by the vector Q of size m with the nonnegative integer components $q_i = Q(p_i)$. For the storing of the marking encoded according to (2), 3 places with the names m, sQ, rQ are used shown in fig. 2 which marking contains values m, sQ, rQ respectively.

4.5 Encoding of the Transitions Firing Sequence

The transitions firing sequence Z of length k is represented by the vector $\bar{Z}$ of size k with nonnegative integer components $z_j = i$, where i is the number of transition t_i firing on the step j. For the storing of the encoded according to (2) sequence, 3 places with the names k, sZ, n are used shown in fig. 2 which marking contains values k, sZ, n respectively.

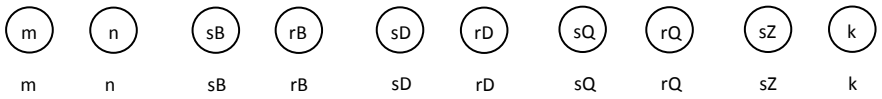


Fig. 2. The representation of the Petri net and transitions firing sequence encoding.

Note that places m, n are used as the parameters for the encoding (decoding) the Petri net graph, marking and transitions firing sequence.

4.6 Encoding of the Enabled Transitions Set

The enabled transitions set of Petri net is auxiliary information for the nondeterministic choice of the firing transition t_l $(u_l = 1)$ on the current step. For the representation of the enabled transitions set, the vector $\bar{u}$ of size n is used which components are the enabling indicators u_i of the corresponding transitions $t_i, i =$

$\overline{0, n-1}$, calculated according to (1). Then for the encoding of $\bar{u}$, the rules of the vector encoding (2) are applied at $r = 2$.

5 Algorithm of Inhibitor Petri Net Executing

On the system (1) according to the chosen way of the encoding of Petri net graph, marking and transitions firing sequence let us construct the algorithm AUIPN of inhibitor Petri net executing using C-like pseudo language:

```
void AUIPN()
{
        uint u, l;

        inputXIPN();
        k=1; sZ=0;
        while(NonDeterministic())
        {
                CheckFire(&u);
                if(u==0) goto out;
                PickFire(u, &l);
                Fire(l);
                mul_add(&sZ,n,l-1);
                k++;
        }
        out:    outputXIPN();
}
```

The following variables are used: u – the code of enabled transitions indicator, l – the number of the firing transition, k – the number of the current step; procedures: CheckFire – checking the transitions enabling conditions, PickFire – the firing transition choice, Fire – the firing of the transition; NonDeterministic – nondeterministic choice of a number belonging to the set $\{0,1\}$. The algorithms of the auxiliary procedures mod_div, mul_add are the following:

```
void mod_div(&m,&x,y)
{
        (*m) = (*x) mod y;
        (*x) = (*x) div y;
}

void mul_add(&x,y,z)
{
        (*x) = (*x) * y + z;
}
```

The algorithm of the procedure CheckFire is the following:

```
void CheckFire(uint *u)
{
        uint i, j, qj, bij, ui, uij;
        uint sB1, sQ1;

        sB1=sB; &u=0;
        for(i=n; i>0; i--)
        {
```

```
                sQ1=sQ;
                ui=1;
                for(j=m; j>0; j--)
                {
                        mod_div(&qj,&sQ1,rQ);
                        mod_div(&bij,&sB1,rB);
                        uij=1;
                        if(bij==0) continue;
                        bij--;
                        if(bij==0) uij=(qj==0);
                        else uij=(qj>=bij);
                        ui=ui && uij;
                }
                mul_add(&u,2,ui);
        }
}
```

Lemma 1. Algorithm CheckFire creates the set of transitions enabled in the current marking.

Proof. The algorithm constitutes the sequential computation of the vector $\bar{u}$ components according to the second line of the system (1) and their simultaneous encoding (2) into the variable u after the calculation of the current component in the variable ui. The loop on the variable i defines the exhaustion of all the transitions, the nested loop on the variable j defines the exhaustion of all the places for the chosen transition. The order of the sequential decoding of matrix B and vector Q elements corresponds to the order of the loops indices modification according to (3) and (5). □

The algorithm of the procedure PickFire is the following:

```
void PickFire(uint u, uint *l)
{
        uint ui, i;

        i=0;
        while(u>0)
        {
                mod_div(&ui,&u,2);
                i++;
                if(ui==0) continue;
                if(NonDeterministic()) goto out;
        }
out:    *l=i;
}
```

Lemma 2. Algorithm PickFire executes the choice of an arbitrary firing transition from the set of enabled transitions.

Proof. The condition of the firing transition choice corresponds to the third line of the system (1) as well as to the order of the vector $\bar{u}$ decoding according to (3). For the nondeterministic choice of the firing transition the function NonDeterministic is used for the exit out of the loop. The condition $u > 0$ provides the loop completion after the last enabled transition processing which is chosen as the firing at least. □

The algorithm of the procedure Fire is the following:

```
void Fire(uint l)
```

```
{
        uint rQ1, maxQ1, shift, qj, bij, dij, j;
        uint sB1, sD1, sQ1;

        sB1=sB; sD1=sD; sQ1=0; rQ1=rQ+rD-1; maxQ1=0;

        shift=(n-l)*m;
        while(shift--)
        {
                mod_div(&b,&sB1,rB);
                mod_div(&d,&sD1,rB);
        }

        for(j=m; j>0; j--)
        {
                mod_div(&qj,&sQ, rQ);
                mod_div(&bij,&sB1, rB);
                if(bij>0) bij--;
                dij=mod_div(&sD1, rD);
                qj=qj-bij+dij;
                maxQ1=max(qj,maxQ1);
                mul_add(&sQ1,rQ1,qj);
        }
        sQ=0; rQ=maxQ1+1;

        for(j=m; j>0; j--)
        {
                mod_div(&qj,&sQ1,rQ1);
                mul_add(&sQ,rQ,qj);
        }
}
```

Lemma 3. Algorithm Fire implements the marking transformation as a result of the specified transition firing.

Proof. The algorithm implements the recalculating of the marking according to the first line of the system (1) and the described way of the matrices B, D and the vector Q decoding according to (5) and (3). The value of the variable shift corresponds to the number of the passing through elements for the positioning to the beginning of the firing transition line with the number l. Then into the first loop on the variable j the preliminary recalculating of the marking code (2) is executed into the variable sQ1; at that the value of rQ1 is used which provides the storing of the maximal possible value of the new marking element rQ+rD-2. For the avoiding the rQ growth, into the second loop on the variable j the final recalculating of the marking code (2) is executed into the variable sQ according to the actual value of the maximal element maxQ1. □

Theorem 1. Algorithm AUIPN implements the dynamics of an arbitrary given inhibitor Petri net.

Proof. Let us show that the algorithm AUIPN recalculates the marking of inhibitor Petri net according to the system (1) and stores the employed transitions firing sequence. The algorithm of the step executing is represented by the loop while of AUIPN and completely corresponds to the system (1). At the beginning, the procedure CheckFire determines the enabled transitions set and forms the code (2) of the corresponding enabled transitions indicator u (Lemma 1). At the absence of the

enabled transitions $u == 0$, the algorithm stops that corresponds to a dead marking. The procedure PickFire implements nondeterministic choice of the firing transition from the set of the enabled transitions; the variable l returns the firing transition number (Lemma 2). The procedure Fire implements the current marking transformation as a result of the transition with the number l firing and its simultaneous encoding (2) (Lemma 3). Then into the code (2) of the transitions firing sequence sZ is added the number l and the value of the current step k is incremented by unit. Nondeterministic exit out of the loop corresponds to the Definition 1. □

Algorithm AUIPN was also encoded in C language using the library MPI for the representation of lengthy integers and tested on a series of Petri nets.

Theorem 2. Algorithm AUIPN can be represented by an inhibitor Petri net.

The Theorem 2 proof is the immediate consequence of the facts that inhibitor Petri net is a universal algorithmic system [1] and the algorithm AUIPN uses nonnegative integer scalar variables only which values can be represented by the marking of the corresponding Petri net places.

For the constructive proof of Theorem 2, the corresponding net is constructed on the algorithm AUIPN in the following sections of the work.

6 Principles of Algorithms Encoding by Inhibitor Petri Net

There are known various approaches to the algorithm encoding by a Petri net based on the principles of combining data flows and control flows [2,4,5]. Let us employ the direct encoding of the basic C language operators for the representing of single control flow. Each of variables is represented by the corresponding place of Petri net; all the variables are static global (fig. 3). The control flow is modeled by the trace of a single token passage from initial place start to the final place finish.

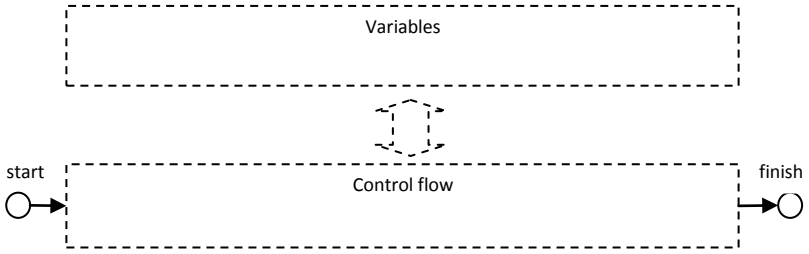


Fig. 3. Overall organization of the net UIPN.

For the unified organization of work with variables let us represent the operators of the programming language in the form shown in fig. 4.

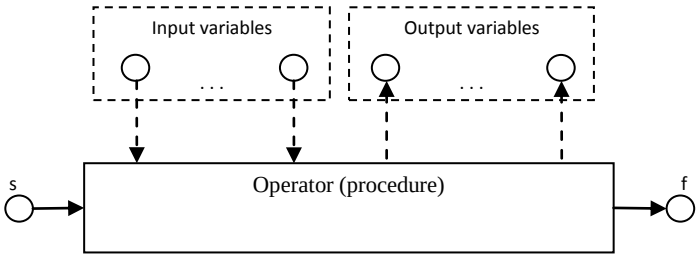


Fig. 4. Representation of the programming language operator.

To provide the reentering the control flow through the operators (procedures) let us adopt the following agreements: all the internal places have zero marking; before the beginning of the work the input variables are copied into the input places of the operator; the work of the operator is launched by a token put into the place start (s); the operator finishes its work at the hitting the place finish (f) by the token; at the completion of work all the places of the operator are empty excepting the output places which contain the result. Dashed arcs denote the following extra rules of the forming the values of the operator input and output variables: at the launch the content of the variable is copied into local input place of the operator; after the completion the variable is cleaned and the value from the local output place of the operator is moved into it (fig. 5).

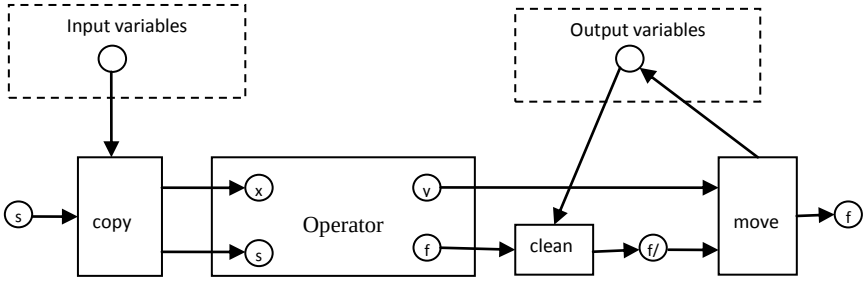


Fig. 5. The forming of input and output variables.

In case of a few variables the chains of copy are created for the sequential copying of input variables and the chains of clean, move for the moving of the output variables values. The sequence of clean, move is denoted as assign. The represented scheme provides the correct work with variables in general case. In some cases the work with variables can be optimized, when they are temporary or input and output at the same time. For the expressions calculating the approach of data flows [2] can be implemented: the executing of operations is ordered according to their priorities; input places of operations are fused with output places of the next operation.

Let us consider the basic control constructions of the programming language: sequence, conditional (unconditional) branch, loop. Let us abstract from the used variables.

Lemma 4. Algorithmic control constructions of the programming language can be encoded by inhibitor Petri net in the following way (fig. 6):

Name	Form	Net
Sequence	`operator1;` `operator2;`	a)
Branch	`if(condition()) then operator1; else operator2;`	b)
Loop while	`while(condition()) operator;`	c)
Loop for	`for(i=n;i>0;i--) operator;`	d)

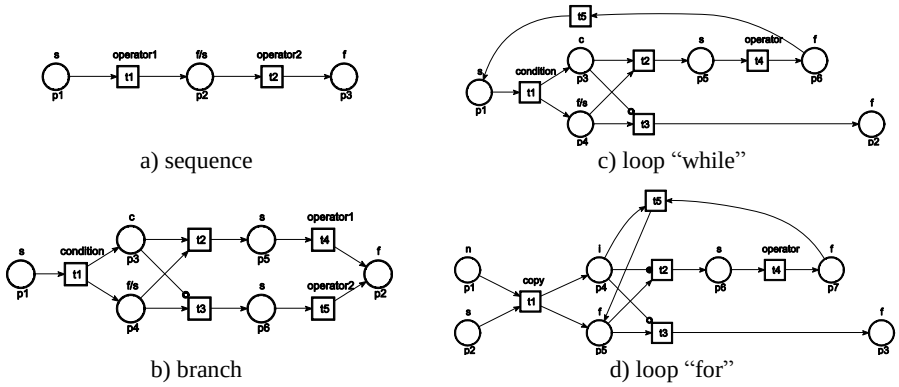


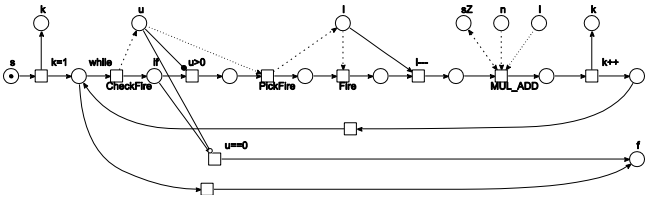
Fig. 6. Encoding of the programming language control constructions.

For each control construction the correctness of its representation can be proven by the way of classifying all the enabled transitions firing sequences and their comparison with the order of operators execution into the constructions of the programming language [2]. Note that according to fig. 6a) the operators superposition at the program encoding is implemented by the merging (fusion) of the output place f of the first operator with the input place s of the second operator.

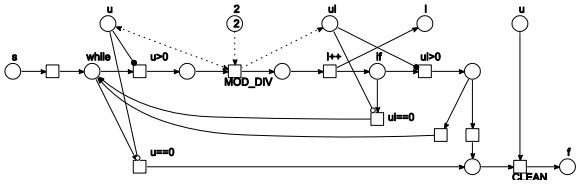
There are known the representations of basic algebraic and logic operations by Petri nets [2,6]. In some cases it is convenient the direct representation of the most used actions such as, for example, mod_div and mul_add for the decoding and encoding of Petri nets. In Appendix B the nets implementing the operations used in the algorithm AUIPN are listed. For the graphical representation of inhibitor arc the hollow circle at the end of arc is used. Arc with the filled circle at its end denotes the couple of arcs with the opposite direction and equal multiplicity; they are used for the checking of a place marking.

Lemma 5. Nets listed in Appendix B implement the specified operations.

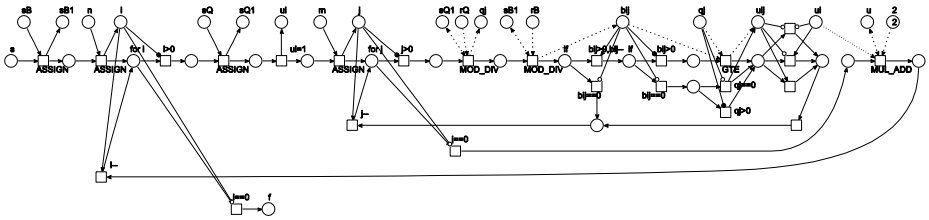
For each of the represented nets it is possible to bring the proof of the correct implementation of the specified operation on the base of all the enabled transitions firing sequences classification [2,6].



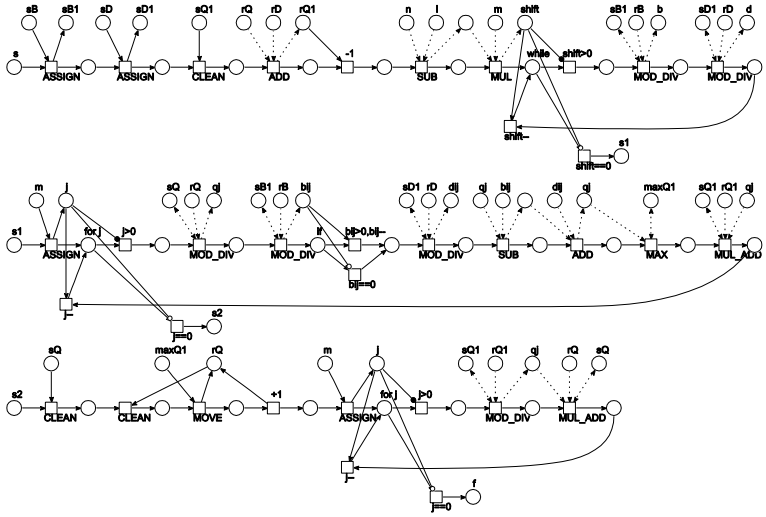
a) UIPN



b) PickFire



c) CheckFire



d) Fire

Fig. 7. Universal inhibitor Petri net UIPN.

7 Composing Universal Inhibitor Petri Net UIPN

Let us encode the algorithm AUIPN of universal inhibitor Petri net work by inhibitor Petri net according to the rules described in Section 6. Note that Lemma 4 and Lemma 5 lists all the control constructions and all the operations employed in the algorithm AUIPN. The net UIPN represented in fig. 7 is obtained. For the representing of the algorithm variables, fused places are used: all the places with the same name are logically the same place; fused places simplifies the graphical representation of the net. Let us suppose that before the net UIPN launch, the code of target (executing) net XIPN is loaded into places shown in fig. 2 and after the stopping of the net UIPN, the code of the marking and the transitions firing sequence of the net XIPN is read from the corresponding places.

Dashed arcs denotes considered in Section 6 agreements on the input and output variables copying. Bidirectional arcs are used for the work with variables which are the both input and output; in this case the copying can be optimized applying twice move without cleaning. In some cases for the copying of an input variable together with its cleaning it is reasonable the usage of move instead of copy; as the corresponding notation the dotted arc is used. The substitution of a transition implies the copying of the corresponding subnet with the merging (fusion) of contact places. In general case the transition substitution requires the indication of input and output places mapping; in the listed nets the places mapping is defined implicitly by the context of the used operations and is not indicated.

Theorem 3. Net UIPN is the universal inhibitor Petri net.

The Theorem 3 proof directly follows from Theorem 1 and the correctness of used rules of sequential algorithm encoding by inhibitor Petri net (Lemma 4) and the correctness of nets implementing the used operations (Lemma 5).

Note that net UIPN is represented in a component-wise way according to the used procedures, operations and the rules of work with variables. There is of a definite interest the binding of UIPN in the form of united inhibitor Petri net and its execution in the environment of a simulating system that simulates the firing of transitions.

8 Conclusions

In the present work the universal inhibitor Petri net was constructed that executes an arbitrary given inhibitor Petri net.

It is possible the constructing of universal nets in other classes of Petri nets which are the universal algorithmic system [2]: priority, synchronous, timed. Moreover, it is possible the combined constructing, for example, of inhibitor net that executes an arbitrary synchronous net.

There are known examples of universal Turing machines constructing with the minimal number of used symbols/states [7,8]. In this connection there is of a definite interest the constructing of universal Petri net with the minimal number of places (transitions), the minimal value of the marking.

References

1. Agerwala T. A Complete Model for Representing the Coordination of Asynchronous Processes, Baltimore, John Hopkins University, Res. Rep. No. 32, July 1974.
2. Kotov V. Petri Nets, Moscow, Nauka, 1984, 160 p. In Russ.
3. The Universal Turing Machine. A Half-Century Survey / Rolf Herken (ed.), Springer-Verlag, Wien New York, 1994, 609 p.
4. Best E, Devillers R., Koutny M. Petri Nets, Process Algebra and Concurrent Programming Languages. Lecture Notes in Computer Science, Vol. 1492: Lectures on Petri Nets II: Applications / Reisig, W.; Rozenberg, G. (eds.), 1998.
5. Goltz U. On Representing CCS Programs by Finite Petri Nets. Proc. MFCS-88, Springer-Verlag Lecture Notes in Computer Science Vol.324, 339-350 (1988).
6. Sleptsov A.I. State equation and equivalent transformations of loaded Petri nets (algebraic approach) // Formal models of parallel computations: Proceedings of all-USSR conference, Novosibirsk (Russia), 1988, p. 151-158. In Russ.
7. Minsky M. Size and structure of universal Turing machines using tag systems. In Recursive Function Theory, Provelence, 1962. AMS, vol. 5, p. 229-238.
8. Rogozhin Y. Small universal Turing machines. TCS, 168(2):215-240, 1996.

Appendix A: Examples of Nets Encoding

1) Petri net graph

Net	m	n	sB	rB	sD	rD
add	6	4	21180169496	3	282946	2
max	8	8	254813592433189871074065241412	3	293862152152879368	2
mul	10	9	646549072061101455668889034663481743952654	3	1935225908529245455975681	2

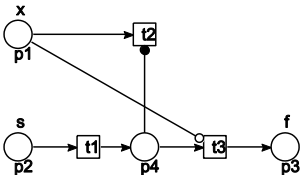
2) Marking

Net	Marking	Q	sQ	rQ
add	addQ0	(2,3,1,0,0,0)	2880	4
add	addQ	(0,0,0,5,1,0)	186	6
max	maxQ0	(2,3,1,0,0,0,0,0)	46080	4
max	maxQ	(0,0,0,3,1,0,0,0)	832	4
mul	mulQ0	(2,3,1,0,0,0,0,0,0,0)	737280	4
mul	mulQ	(0,0,0,6,1,0,0,0,0,0)	722701	7

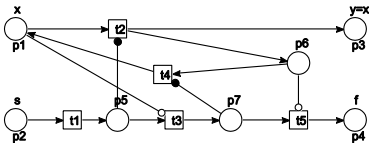
3) Transitions firing sequence

Net	Q0	Q	Z	sZ	k
add	addQ0	addQ	t1,t3,t2,t2,t3,t3,t4	2411	7
max	maxQ0	maxQ	t1,t2,t2,t6,t7,t8	4983	6
mul	mulQ0	mulQ	t1,t2,t4,t4,t5,t6,t6,t7,t2,t4,t4,t5,t6,t6,t7,t2,t4,t4,t5,t6,t6,t7,t3,t9,t9,t8	109815712212339723705298	26

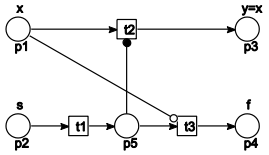
Appendix B: Implementation of Used Operations



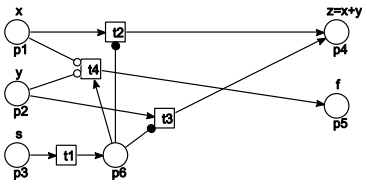
CLEAN ($x = 0$)



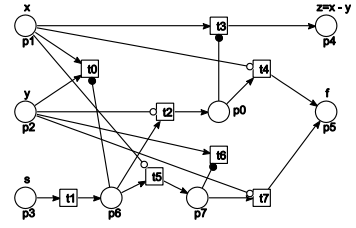
COPY ($y = x$)



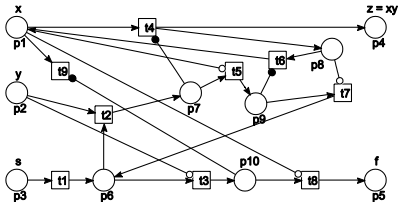
MOVE ($y = x, x = 0$)



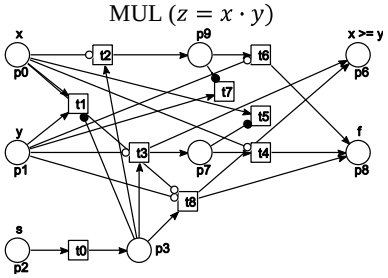
ADD ($z = x + y$)



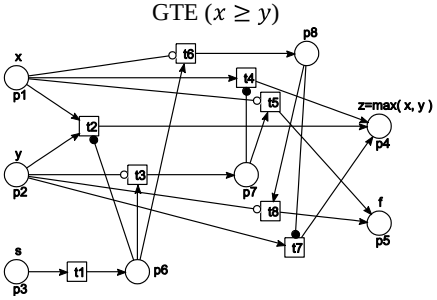
SUB ($z = x \dot{-} y$)



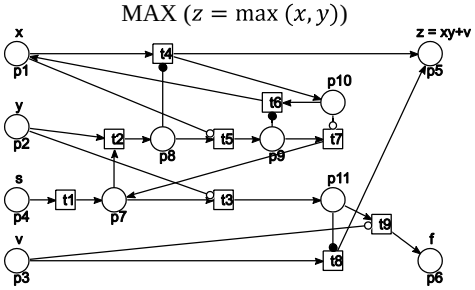
MUL ($z = x \cdot y$)



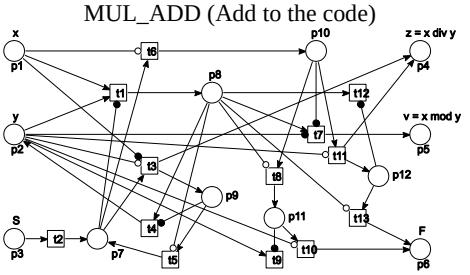
GTE ($x \geq y$)



MAX ($z = \max(x, y)$)



MUL_ADD (Add to the code)



MOD_DIV (Extract from the code)

On Optimizing the Sweep-Line Method

Robert Prüfer

Humboldt-Universität zu Berlin, Institut für Informatik
Unter den Linden 6, 10099 Berlin, Germany
pruefer@informatik.hu-berlin.de

Abstract. For applying the *sweep-line method*, a state space reduction technique, it is necessary to provide an automatic calculation of a *progress measure*. In [1], such a calculation has been proposed for Petri nets. The approach presented there may lead to suboptimal results, therefore the author suggested possible optimizations. In this paper, we check whether the proposed optimization goals indeed lead to an improved performance of the sweep-line method.

1 Introduction

The *sweep-line method* is a state space reduction technique that can be used to verify safety properties (e.g. the existence of deadlocks or reachability of a given state) of the modelled system. The main idea of this technique is that during state space exploration, there is some kind of "progress": States that have been processed and will never be visited again are no longer needed for further exploration. To quantify this progress, a *progress measure* is needed that assigns a *progress value* to every state. To use the sweep-line method fully automatically, the progress measure must be calculated automatically, too. Such an automatic calculation of the progress measure for applying the sweep-line method to state spaces of Petri nets has been proposed in [1]. This approach leaves degrees of freedom and, in some cases, calculates a progress measure that does not lead to an optimal state space reduction. In [1], the author suggests two optimization goals to improve the performance of the sweep-line method: minimizing the number of so-called *regress transitions* and avoiding chains of such regress transitions. In this publication, we develop methods to achieve the proposed optimization goals and apply our modified calculation of the progress measure in an experiment to check whether it leads to an improved state space reduction.

2 Definitions

We write $N = [P,T,F,W,s_0]$ for a place/transition Petri net with arc weights, where P,T and F denote places, transitions and arcs. $W : ((P \times T) \cup (T \times P)) \to \mathbb{N}$ denotes the arc weights, where $W(x,y) = 0$ iff $(x,y) \notin F$. s_0 is the initial marking of the net. For every $t \in T$, we define the vector Δt by $\Delta t(p) = W(t,p) - W(p,t)$ for all $p \in P$. For $X = P \cup T$ and every $x \in X$, we define ${}^\bullet x = \{y \in X \mid (y,x) \in F\}$ and $x^\bullet = \{y \in X \mid (x,y) \in F\}$. A place $p \in P$ is a *shared place* iff $|p^\bullet| \geq 2$. We write $s \xrightarrow{t} s'$ for an edge of the state space of N, where firing t in state s leads to the *target state* s', which is the *successor of* s.

A *directed graph* is written as $G = [V,E]$, where V is the set of vertices and $E \subseteq (V \times V)$ is the set of directed edges.

3 The Sweep-Line Method

There exist two forms of the sweep-line method: the basic form [2], which is only suitable for cycle-free state spaces, and the generalized form [3], which can be applied to all types of state spaces. We first describe the basic method and afterwards extend it to the generalized method.

As we stated above, the *progress measure* assigns a progress value $p(s)$ to every state s of the state space. For the basic sweep-line method, the progress measure is monotonous, i.e., for all states s, s' and transitions t, $s \xrightarrow{t} s'$ implies $p(s) \leq p(s')$. The state space is explored such that states with low progress values are visited first. Therefore, we can divide the set of states into three classes: (1) states that have been explored including all their successors, (2) states that have been explored, but which's successors have not been completely explored (we also call this class *front*) and (3) states that have not been explored yet. As the progress measure is monotonous, all states of class (1) can be deleted; only the states of the front must be stored.

For the generalized sweep-line method, the progress measure does not have to be monotonous, i.e., for an edge $s \xrightarrow{t} s'$ of the state space it may hold $p(s) > p(s')$. Such edges are called *regress edges*. Applying the basic method would now lead to the following problem: Let $s \xrightarrow{t} s'$ be a regress edge with s in class (2) or (3) and s' in class (1). Because all states of class (1) have been deleted, we have no information that we have explored s' yet. The exploration would be continued from this state and s' will eventually be explored again as an unknown state – the method would not terminate. That is why in the generalized sweep-line method target states of regress edges are marked as *persistent* and stored permanently in memory. In the current iteration (*sweep*) of the sweep-line method, successors of persistent states are not explored, and newly marked persistent states are initial states of the next sweep. Therefore, it is guaranteed that the method terminates and explores all states at least once.

It is obvious that a large number of persistent states increases memory consumption and can lead to a large number of sweeps.

The sweep-line method can be combined with stubborn sets, another state space reduction method; the computation method for stubborn sets described in [4] can be used for this purpose.

4 Calculation of Progress Values for Petri nets

The progress measure required for the sweep-line method needs to be generated for each system which's state space should be explored. The automatic calculation of a progress measure for Petri nets described in [1] is divided in two phases. Before state space exploration is started, an *offset function* o that maps every transition of a given Petri Net into an *offset value* $o(t) \in \mathbb{Q}$ is calculated. During state space exploration, offset values are combined to progress values: For the initial state s_0, set $p(s_0) = 0$. Then, for every edge $s \xrightarrow{t} s'$ of the state space, set $p(s') = p(s) + o(t)$. For each transition t with $o(t) < 0$, every edge $s \xrightarrow{t} s'$ of the state space is a regress edge. Therefore, such transitions are called *regress transitions* (*RTs*).

Using this approach for calculating a consistent progress measure, offset values must preserve the linear dependencies of all vectors Δt.

Definition 1 (Linear Dependency Preserving Offset Function). *Let N be a Petri net with transitions T. An offset function o is* linear dependency preserving *if for all $t \in T$, $\{t_0,\ldots,t_n\} \subseteq T \setminus \{t\}$, $\lambda_0,\ldots,\lambda_n \in \mathbb{Q}$ with $\Delta t = \lambda_0 \Delta t_0 + \cdots + \lambda_n \Delta t_n$ it holds $o(t) = \lambda_0 o(t_0) + \cdots + \lambda_n o(t_n)$.*

The method for computing offset values suggested in [1] leaves degrees of freedom and, in some cases, calculates negative offset values for more transitions than necessary. In other words, the calculation of offset values can be optimized. For the purpose of optimization, we propose a computation method for offset values that differs from the one described in [1]. Let $N = [P,T,F,W,s_0]$ be a Petri net and let $a \in \mathbb{Q}^{|P|}$ be chosen arbitrarily but fixed. Then the offset value of any transition $t \in T$ can be computed by $o(t) = a \cdot \Delta t$. As offset values computed this way are linear dependency preserving (see Def. 1), a consistent progress measure will be generated when they are used.

5 Optimized Calculation of Offset Values

The computation method for offset values described in Sect. 4 is suitable for the purpose of optimization, as for attaining any optimization goal, only one vector $a \in \mathbb{Q}^{|P|}$ must be chosen properly to calculate the offset value $o(t) = a \cdot \Delta t$ for any transition t of a Petri net. As we already mentioned, we want to attain two optimization goals: First, the number of RTs should be minimized (as one could expect that this reduces the number of persistent states and thus the number of sweeps) and second, chains of RTs should be avoided (as such chains could lead to an unneccessary re-exploration of huge parts of the state space). In our approach described in the following, at first we will compute several sets of RTs which are as small as possible, and afterwards choose a set R which promises to avoid chains of RTs the most. Afterwards, we can calculate a vector a such that $o(t) < 0$ iff $t \in R$.

5.1 Minimizing the Number of Regress Transitions

For this subsection, let $N = [P,T,F,W,s_0]$ be a Petri net. Minimizing the number of RTs means that we want to compute as few negative offset values as possible for all transitions $t \in T$. Furthermore, we decide that we do not want to assign the offset value $o(t) = 0$ to any transition $t \in T$. Therefore, with regard to the computation method described in Sect. 4, for every transition $t \in T$ we set up the inequality $a \cdot \Delta t > 0$. Thus, we obtain a system of linear inequalities which we call $\mathscr{L}$.

Obviously, if $\mathscr{L}$ is feasible, we can calculate positive offset values for all transitions $t \in T$. We may choose any vector $a \in \mathbb{Q}^{|P|}$ from the set of feasible solutions described by $\mathscr{L}$ (which can be done by a Linear Programming solver) to obtain offset values without RTs. Of course, in this case we do not have to avoid chains of RTs as there are none.

If $\mathscr{L}$ is not feasible, things become more difficult. To minimize the number of RTs, we are looking for a feasible subsystem of $\mathscr{L}$ containing as many inequalities as possible. This almost equates to solving an instance of the *Maximum Feasible Subsystem*

Problem (MAXFS) [5]; we only have to transform every inequality from $a \cdot \Delta t > 0$ to $a \cdot \Delta t \geq \varepsilon$ with a preferably small $\varepsilon \in \mathbb{Q}$ and $\varepsilon > 0$. As MAXFS is NP-hard [6], we decided to use a heuristic based on Linear Programming, namely Algorithm 1 from [7], to solve our MAXFS instances. As this heuristic only provides a single solution (like all MAXFS heuristics known to us), we modified it to obtain multiple solutions (see [8] for details). Now, for any feasible subsystem $\mathscr{L}'$ we obtained, we can choose an arbitrary vector $a \in \mathbb{Q}^{|P|}$ from the set of feasible solutions described by $\mathscr{L}'$ to compute offset values with a minimal number of RTs.

5.2 Avoiding Chains of Regress Transitions

As we are now able to generate multiple sets of RTs, in this section we want to develop a heuristic that should select the set which avoids chains of RTs the most. In contrast to the minimization of RTs which can be performed prior to state space exploration, chains of RTs are initially detected during state space exploration. As offset values are calculated prior to state space exploration, we have to think about how to optimize offset value calculation to avoid chains of RTs anyway. To attain this optimization goal, we introduce the *enabling graph*.

Definition 2 (Enabling Graph). *Let $N = [P,T,F,W,s_0]$ be a Petri net. $G = [V,E]$ is the* enabling graph *of N, where $V = T$, and $(t,t') \in E$ iff $\exists p \in P : t \in {}^\bullet p \wedge t' \in p^\bullet$.*

If there is an edge from t to t' in the enabling graph, then firing t creates at least one token on a place $p \in {}^\bullet t'$. Therefore, firing t supports enabling t'. In our heuristic for avoiding chains of RTs, we use shortest paths within the enabling graph to decide whether a set of RTs is likely to avoid chains or not. Although this approach has some weaknesses (for example, it does not take account of concurrency of transitions and of the actual marking of the net), we consider it to be appropriate for avoiding chains of RTs prior to state space exploration. For a given Petri Net $N = [P,T,F,W,s_0]$ and its enabling graph $G = [V,E]$, our heuristic for avoiding chains of RTs chooses a set R of RTs that satisfies the criteria described in the following.

1. Shortest paths from each t to each t' with $t,t' \in R$ should be as long as possible.
2. Shortest paths from each transition t enabled in s_0 to each $t' \in R$ should be as long as possible.
3. For each $t \in R$ and each $p \in {}^\bullet t$, it should hold $p \notin {}^\bullet t'$ for as many $t' \in R \setminus \{t\}$ and $p \in {}^\bullet t''$ for as many $t'' \in T \setminus R$ with $\Delta t''(p) < 0$ as possible.

Criterion 1 aims at maximizing the number of explored states between any two regress transitions. Especially, t' should not fire immediately after t has fired. Furthermore, in our implementation used for the case study in Sect. 6, we demand that the shortest path among all shortest paths between RTs should be preferably long. It is obvious that this criterion helps us to avoid chains of RTs.

Criterion 2 states that during the first sweep, as many transitions as possible should fire before a regress edge is explored. This does not avoid chains of RTs immediately, but it has the effect that a large part of the state space should be already explored in the first sweep.

Table 1. Experimental results. The entries show for how many nets LoLA_H provides better results (+*x*) and worse results (-*x*) compared to LoLA_O. *complete* refers to successful exploration of the complete state space, *#RT* to the number of RTs, *peak* to the peak number of states during state space exploration, *persistent* to the number of persistent states and *sweeps* to the number of sweeps.

	complete		*#RT*		*peak*		*persistent*		*sweeps*	
-stub	+2	-0	+24	-0	+11	-10	+11	-7	+7	-8
+stub	+0	-0	+24	-0	+16	-7	+17	-7	+12	-9

Criterion 3 takes account of shared places. We want to avoid that a RT t shares a place p with another RT, because if the marking of p is sufficiently big, then these two transitions could fire one after another, which would exactly yield a chain of RTs. On the other hand, if t shares a place with a transition t' that is not a RT and t' consumes more tokens from p than it produces on this place, the firing of t' may disable t. Therefore, such transitions should preferably be chosen as RTs.

In our heuristic, we weight these criteria and combine them to a function f that assigns a value to every set of RTs (see [8] for details). The set to which f assigns the largest value is chosen as the set that should avoid chains of RTs the most.

6 Case Study

In our case study, we checked whether the computation of progress values given in [1] really leads to a worse performance of the sweep-line method compared to the computation method and heuristics described in this paper. For this purpose, we tried to compute the state space of 32 Petri nets from a GALS project [9]. We used two versions of the model checking tool LoLA [10]: the original version (*LoLA_O*) which implements the calculation described in [1] as well as a modified version (*LoLA_H*) where we implemented our offset calculation method and heuristics. We executed the state space computation once with stubborn sets (+*stub*) and once without stubborn sets *(-stub)*.

Table 1 shows a summary of our test results. The state space was too large to be computed completely by LoLA_O for 6 nets with stubborn sets and for 10 nets without stubborn sets. In the table, these nets are not considered except for the columns *complete* and *#RT*.

From the results, we conclude that our optimization goals do not lead to an improved performance of the sweep-line method in general. Moreover, as stated in [8], none of the assumed correlations mentioned above (*#RT* and *persistent*, *#RT* and *sweeps*) is verified by the results. Using -*stub*, for 14 nets where the full state space could be computed, *#RT* was reduced, but *persistent* was only decreased for 8 of them and *sweeps* for 5 of them. For +*stub* and -*stub*, for some nets *#RT* was reduced significantly, but *peak*, *persistent* and *sweeps* even got larger, what means that the values for memory consumption and the number of iterations got worse. Changing the weights of the criteria for avoiding chains mentioned in Sect. 5.2 does not change the results in general.

As the computation of the offset values does not depend on stubborn sets, we obtain the same number of regress transitons for state space exploration with and without stubborn sets.

Although the results for the combination of the sweep-line method and stubborn sets look more promising than those obtained without stubborn sets, note that our optimization goals did not include special improvements for stubborn sets. Especially, computation of stubborn sets combined with the sweep-line method in LoLA depends on the offset values of all transitions; thus, one should not reason about these results without taking a closer look at LoLA.

7 Conclusion and Further Work

The case study in Sect. 6 revealed that the optimization goals examined in Sect. 5 do not improve the performance of the sweep-line method in general. Different optimization goals must be found to achieve this aim.

For estimating the number of persistent states, it turns out that it would be important to know how often RTs are enabled during state space exploration and how many distinct target states they lead to. Furthermore, even offset values of transitions which are no RTs influence the *peak* value. It would be appropriate to provide an offset value calculation that exploits this fact. Finally, as mentioned above, when using LoLA, the implementation of stubborn sets should be taken into consideration when trying to optimize the sweep-line method.

References

1. Schmidt, K.: Automated generation of a progress measure for the sweep-line method. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 192-204. Springer, Berlin Heidelberg (2004)
2. Christensen, S., Kristensen, L.M., Mailund, T.: A Sweep-Line Method for State Space Exploration. In: Margaria, T., Yi, W. (eds.) TACAS 2001. LNCS, vol. 2031, pp. 450–464. Springer Berlin Heidelberg (2001)
3. Kristensen, L.M., Mailund, T.: A Generalised Sweep-Line Method for Safety Properties. In: Eriksson, L.-H., Lindsay, P.A. (eds.) FME 2002. LNCS 2391, pp. 215–239. Springer Berlin Heidelberg (2002)
4. Schmidt, K.: Stubborn Sets for Standard Properties. In: Donatelli, S., Kleijn, H.C.M. (eds.) ICATPN'99. LNCS, vol. 1639, pp. 46–65. Springer, Berlin Heidelberg (1999)
5. Amaldi, E., Pfetsch, M.E., Trotter, L.E. Jr.: On the maximum feasible subsystem problem, IISs and IIS-hypergraphs. Math. Program. 95 no. 3, pp. 533-554 (2003)
6. Amaldi, E., Kann, V.: The Complexity and Approximability of Finding Maximum Feasible Subsystems of Linear Relations. Theor. Comput. Sci. 147 no. 1-2, pp. 181-210 (1995)
7. Chinneck, J.W.: Fast Heuristics for the Maximum Feasible Subsystem Problem. INFORMS J. Comput. 13 no. 3, pp. 210–223 (2001)
8. Prüfer, R.: Optimierung der Sweep-Line-Methode. Humboldt-Universität zu Berlin, diploma thesis (2010)
9. Stahl, C., Reisig, W., Krstic, M.: Hazard Detection in a GALS Wrapper: A Case Study. In: Desel, J., Watanabe, Y. (eds.) Proceedings of the Fifth International Conference on Application of Concurrency to System Design (ACSD05), pp. 234-243. IEEE Computer Society (2005)
10. Schmidt, K.: LoLA: A Low Level Analyser. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 465–474. Springer, Berlin Heidelberg (2000)

Embedding the free-choice semantics of AND/XOR-EPCs into the Boolean semantics

Christoph Schneider[1], Joachim Wehler[2]
[1]ckes@gmx.net, [2]joachim.wehler@gmx.net, Ludwig-Maximilians-Universität München

Abstract. Each Event-driven Process Chain (EPC) translates into a free-choice system if its control flow branches and joins only at AND- or XOR-connectors. This free-choice system defines the free-choice semantics of the AND/XOR-EPC. But free-choice systems are not capable to deal with OR-connectors. Therefore a general EPC with OR-connectors obtains a semantics not until it has been translated into a certain coloured Petri net, named a Boolean system. This Boolean system defines the Boolean semantics of the EPC. We show that for well-behaved AND/XOR-EPCs the Boolean semantics reduces to the free choice semantics in as far as the Boolean system contains the free-choice system. To prove this result we introduce the concept of non-blocking components in live and safe free-choice systems. For each non-blocking component of the free-choice system we then construct a well-behaved bipolar system (bp-system), which is a particular Boolean system. We link the bp-systems of all non-blocking components of a covering to a coloured Petri which is named a linked bp-system. Its semantics is the Boolean semantics of the AND/XOR-EPC.

Keywords: Bipolar system, EPC, free-choice system, linked bp-system, non-blocking component.

1. Introduction

Free-choice systems form an important class of ordinary Petri nets. They are best analyzed and understood, and the theory of free-choice systems is both deep and elegant [DE1995]. Also for commercial applications of information management free-choice systems play an important role. In the context of Business Process Management (BPM) they serve to formalize process languages which have been introduced in a more informal way and lacked a well-defined semantics before.

The process modelling language most widespread in German commercial projects is the language of Event-driven Process Chains (EPC). It has been introduced by Keller, Nüttgens and Scheer in 1992 [KNS1992, Sch1994]. EPCs represent the control flow of a process as the interplay of three components: Events, functions and logical rules. The rules use connectors of logical type AND, XOR and OR. More specific, concurrency is represented by AND-splits and AND-joins. Strong or exclusive alternatives are modelled by XOR-splits and XOR-joins, while OR-splits and OR-joins model weak alternatives. All EPCs in this paper will be considered with a non-empty set of distinguished events, the initial events of the process.

The present paper deals mainly with AND/XOR-EPCs, i.e. with the restricted class of EPCs using only connectors of logical type AND or XOR. Each AND/XOR-EPC translates at once into a free-choice system FS : Functions and AND-connectors of

the EPC translate into transitions while events and XOR-connectors translate into places of FS. Each initial event of the EPC is marked by a token on the corresponding place of FS. The free-choice semantics of the AND/XOR-EPC is defined as the semantics of FS [Aal1999].
But the language of free-choice system is not capable to formalize EPCs with OR-connectors. Therefore we have introduced in a previous paper Boolean systems, a class of simple coloured Petri nets [LSW1998]. Boolean systems have two types of tokens, high tokens and low tokens. The low tokens serve to skip actions and to complete the marking of all pre-sets of a logical transition before a decision about its actual firing mode is possible. With the help of formulas from propositional logic the transitions of Boolean systems control the flow of the high tokens (*true*) and the low tokens (*false*).
A general EPC translates at once into a Boolean system BS: Functions and logical connectors of the EPC translate into transitions and events translate into places of BS. Each initial event of the EPC is marked by a high token at the corresponding place of BS and if necessary a suitable set of low tokens is added.
Those Boolean systems, which are needed for the restricted class of AND/XOR-EPCs, have been invented already in 1984 by Genrich and Thiagarajan [GT1984]. They named them Bipolar Synchronization Schemes, today abbreviated as bipolar systems (bp-system).

How do these two types of Petri nets, bipolar systems and free-choice systems, relate?

It turns out that each bp-system BS has a free-choice companion FS and a canonical morphism $high : BS \longrightarrow FS$ which maps the flow of high tokens of the coloured Petri net BS onto the flow of tokens of the free-choice system FS. Both systems are equivalent in as far as FS is well-behaved if and only if BS is well-behaved. In that case the morphism has the lifting property, i.e., it lifts occurrence sequences of FS to occurrence sequences of BS [Weh2010].
Yet, this equivalence holds only under the restriction that the behaviour of the free-choice system is fair. Here fairness is conceived as the absence of frozen tokens. That type of fairness is even a structural property, named non-blocking.
Therefore the present paper investigates a generalization of the above mentioned relation between the two classes of Petri nets. Relinquishing the non-blocking condition we prove:

For each well-behaved free-choice system FS a well-behaved linked bp-system LBS and a morphism

$$high : LBS \longrightarrow FS$$

exist, which maps the flow of high tokens of the coloured Petri net LBS onto the token flow of the free-choice system FS and satisfies the lifting property (Theor. 17 and Prop. 19).
As a consequence: For an AND/XOR-EPC, which translates into a well-behaved free-choice system FS, a well-behaved linked bp-system LBS exists

with $high(LBS) = FS$. We define the Boolean semantics of the EPC as the semantics of the coloured Petri net LBS. As a consequence, the free-choice semantics and the Boolean semantics of well-behaved AND/XOR-EPCs are equivalent. And this results allows us to consider the Boolean semantics of general EPCs a proper generalization of the free-choice semantics of AND/XOR-EPCs.
During our way in this paper we introduce two new concepts for Petri nets: Firstly *non-blocking components* of well-behaved free-choice systems and secondly the *linking* of bp-systems with respect to a family of morphisms.

2. Free-choice systems

For the convenience of the reader and to fix the notations we recall some fundamental concepts from the theory of ordinary Petri nets and define the subclass of free-choice systems.
A finite *ordinary Petri net* is a pair (N, μ): The *net* $N = (P, T, F)$ comprises a finite set P of *places*, a disjoint finite set T of *transitions* and a set $F \subseteq (P \times T) \cup (T \times P)$ of *directed arcs*. The function $\mu : P \longrightarrow \mathbf{N}$ is named the *initial marking* of the net. The *support* of the marking μ is the set

$$supp(\mu) := \{\, p \in P : \mu(p) > 0 \,\}$$

of all places marked at μ. All Petri nets in this paper will be assumed finite.
A *path* from a node $x_{ini} \in X := P \cup T$ to a node $x_{fin} \in X$ is a sequence $(x_0, x_1, ..., x_n)$ with nodes $x_i \in X$, $x_0 = x_{ini}$, $x_n = x_{fin}$ and $(x_i, x_{i+1}) \in F$. It is named *elementary path*, if $x_i \neq x_j$ for all pairs $i \neq j$. The net N is *strongly connected* if for every two nodes $x_1, x_2 \in X$ a path from x_1 to x_2 and a path from x_2 to x_1 exists.
A transition with a single pre-place and two or more post-places is an *opening* transition, a transition with a single post-place and two or more pre-places is called a *closing* transition. Opening transitions with exactly two post-places and closing transitions with exactly two pre-places are called *binary* transitions. A net N is called *binary* if all its transitions are binary.
For a net N the *firing rule* defines the firing of a transition: A transition $t \in T$ is *enabled* at a marking μ of N iff each place from its pre-set $pre(t)$ is marked at μ with at least one token. Being enabled, t may *occur* or *fire*. Firing t yields a new marking μ', which results from μ by consuming one token from each pre-place of t and by producing one additional token on each post-place of t; this is denoted by $\mu \xrightarrow{t} \mu'$.
A finite *occurrence sequence* from μ is a sequence $\sigma = t_1 ... t_k$, $k \in \mathbf{N}$, such that

$$\mu \xrightarrow{t_1} \mu_1, ..., \mu_{k-1} \xrightarrow{t_k} \mu_k .$$

We denote by $\mu \xrightarrow{\sigma} \mu_k$ the fact, that firing σ yields the marking μ_k. A *reachable marking* of a Petri net (N, μ) is a marking, which results from firing a finite occurrence sequence from μ. If not stated the contrary, occurrence sequences in this paper will be considered finite occurrence sequences. The *concatenation of two occurrence sequences* σ_1 and σ_2 is denoted by $\sigma_1 \cdot \sigma_2$.

A Petri net (N, μ_0) is *live* iff for each reachable marking μ and for each transition $t \in T$ the Petri net (N, μ) has a reachable marking which enables t. A Petri net is k*-bounded* iff a number $k \in \boldsymbol{N}$ exists bounding from above the token content of every place at every reachable marking. If the bound can be chosen as $k = 1$ then the Petri net is named *safe*. A live and safe Petri net is named *well-behaved*. A net N is *well-formed* iff there exists a marking μ_0 of N such that the Petri net (N, μ_0) is live and bounded.

We will often dispense with an explicit notation for the set of places and transitions of a net and use the shorthand $x \in N$ to denote a node of the net.

1. Definition (P-system, T-system, free-choice system)

i) A net N is a *P-net* if all transitions have exactly one pre-place and exactly one post-place, i.e.

$$card\,[pre(t)] = 1 = card[post(t)] \text{ for all transitions } t \in N.$$

A *P-system* is a Petri net (N, μ) with N a P-net.

ii) A net N is a *T-net* if all places have exactly one pre-transition and exactly one post-transition, i.e.

$$card\,[pre(p)] = 1 = card[post(p)] \text{ for all places } p \in N.$$

A *T-system* is a Petri net (N, μ) with N a T-net.

iii) A net N is a *free-choice net* if for every two transitions $t_1, t_2 \in N$

$$\text{either } pre(t_1) \cap pre(t_2) = \varnothing \text{ or } pre(t_1) = pre(t_2).$$

A *restricted free-choice net* is a net which satisfies the stronger condition: For every two transitions $t_1, t_2 \in T$

$$\text{either } pre(t_1) \cap pre(t_2) = \varnothing \text{ or } pre(t_1) = pre(t_2) = \{\, p \,\}$$

with a single place $p \in P$. A marked (restricted) free-choice net (N, μ) is named *(restricted) free-choice system*.

Well-behaved free-choice systems are one of the two classes of Petri nets studied in the present paper. By a theorem of Genrich each well-formed free-choice net FN has a marking μ, such that (FN, μ) is even well-behaved ([De1995] Theor. 5.10).

An important means for the examination of free-choice systems is the study of their P-components and T-components.

2. Definition (Components and their intersection)

Consider a net $N = (P, T, F)$.

i) A subnet $N_P \subseteq N$ which is generated by a nonempty subset $X \subseteq P \cup T$ of nodes, is a *P-component* of N if N_P is a strongly connected P-net with

$$pre(p) \cup post(p) \subseteq X \text{ for all places } p \in X .$$

Consider a marking μ of N. If a P-component $N_P \subseteq N$ is marked at μ with a single token then $(N_P, \mu_P), \mu_P := \mu \mid N_P$, is named a *basic component* of (N, μ).

ii) A subnet N_T of N which is generated by a nonempty subset $X \subseteq P \cup T$ of nodes, is a *T-component* of N if N_T is a strongly connected T-net with

$$pre(t) \cup post(t) \subseteq X \text{ for all transitions } t \in X .$$

iii) The net N is *structurally non-blocking* iff every P-component N_P of N intersects every T-component N_T of N in a non-empty set $N_P \cap N_T \neq \emptyset$. Otherwise the net is named *structurally blocking*. A Petri net (N, μ) is *non-blocking* if its underlying net N is structurally non-blocking. Otherwise the Petri net is named *blocking*.

3. Example (Well-behaved, but blocking free-choice system)

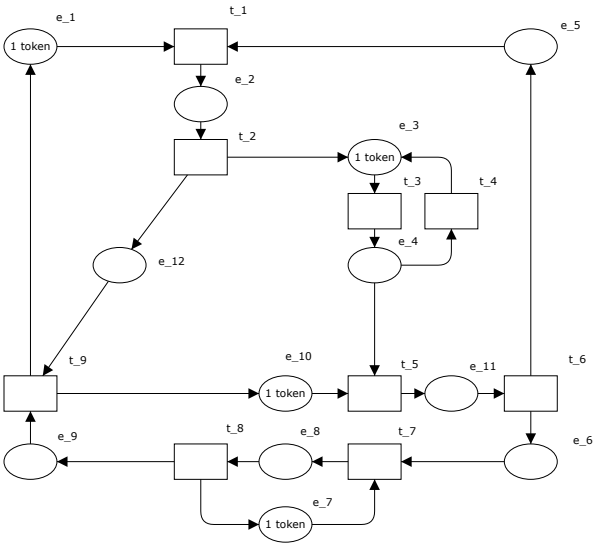


Figure 1: Well-behaved free-choice system FS

Figure 1 shows a well-behaved restricted free-choice system FS. It is blocking: E.g., the P-component N_P generated by the set $\{e_7, t_7, e_8, t_8\}$ and the T-component N_T generated by the set $\{e_3, t_3, e_4, t_4\}$ are disjoint.

To prepare the introduction of the new concept of non-blocking components we recall some properties of well-behaved free-choice systems.
Each well-behaved free-choice system can be covered by basic components. Each occurrence sequence which fires only transitions from one of these basic components lifts to an occurrence sequence of the whole free-choice system. This has been observed by Thiagarajan and Voss first. After introducing the concept of a morphism of Petri nets we will formulate their result as the lifting property of a certain morphism.
Figure 1 shows a well-behaved restricted free-choice system FS. It is blocking: E.g., the P-component N_P generated by the set $\{e_7, t_7, e_8, t_8\}$ and the T-component N_T generated by the set $\{e_3, t_3, e_4, t_4\}$ are disjoint.

To prepare the introduction of the new concept of non-blocking components we recall some properties of well-behaved free-choice systems.
Each well-behaved free-choice system can be covered by basic components. Each occurrence sequence which fires only transitions from one of these basic components lifts to an occurrence sequence of the whole free-choice system. This has been observed by Thiagarajan and Voss first. After introducing the concept of a morphism of Petri nets we will formulate their result as the lifting property of a certain morphism.

4. Remark (Morphisms of Petri nets)
Within the category of coloured Petri nets the concept of a morphism

$$f : PN_1 \longrightarrow PN_2$$

between two coloured Petri nets is well-defined, cf. [Weh2006]. Our concept of a morphism presupposes coloured nets for the domain and range of the morphism, because a morphism maps respectively, certain T-flows and P-flows of PN_1 to binding elements and token elements of PN_2.
The reader, who is not interested in the general definition of a morphism, may use his own descriptive concept of a morphism $PN_1 \xrightarrow{f} PN_2$ to follow the examples of this paper. In most cases the domain of definition PN_1 will be an ordinary Petri net and the coloured Petri net PN_2 will be equivalent to an ordinary Petri net, too. In addition, all morphisms under consideration will be *discrete*, i.e. for any node $y \in PN_2$ the fibre $f^{-1}(y) \subset PN_1$ has only isolated nodes.

Any discrete Petri net morphism $PN_1 \xrightarrow{f} PN_2$ maps occurrence sequences of PN_1 to occurrence sequences of PN_2. The question about the surjectivity of this map is named the lifting problem.

5. Definition (Lifting property of a morphism)
A Petri net morphism

$$PN_1 \xrightarrow{f} PN_2$$

has the *lifting property* iff for any enabled occurrence sequence σ_2 of PN_2 an enabled occurrence sequence σ_1 of PN_1 exists with $f(\sigma_1) = \sigma_2$. The occurrence sequence σ_1 is named *a lift of* σ_2 *against* f.

Any enabled occurrence sequence of a basic component of a well-behaved free-choice system lifts to an enabled occurrence sequence of the whole system.

6. Proposition (Lifting property for basic components)
Consider a well-behaved free-choice system $FS = (N, \mu)$ and a basic component N_B of FS. Then the projection

$$\pi_B : FS \longrightarrow (N_B, \mu \mid N_B)$$

has the lifting property.
Proof. [TV1984], Theor. 2.1 proves the claim under the additional assumption that the free-choice system FN is restricted. But any cluster from a free-choice net can be substituted by two clusters of a restricted free-choice net. Therefore it suffices to prove the claim for restricted free-choice systems, q. e. d.

7. Corollary (Union of basic components)
Consider a well-behaved free-choice system $FS = (N, \mu)$ and a subnet $N_1 \subset N$ which is the union of basic components of FS. Then the projection

$$\pi_1 : FS \longrightarrow FS_1$$

onto the restriction $FS_1 := (N_1, \mu \mid N_1)$ has the lifting property and FS_1 is well-behaved.
Proof. Any union of P-components of a free-choice net is free-choice itself. P-components are transition bounded. Therefore also the subnet $N_1 \subset N$ is transition bounded, which implies that the projection $\pi_1 : FS \longrightarrow FS_1$ is a morphism of Petri nets. In order to verify its lifting property it suffices to consider an occurrence sequence σ_1 of FS_1 with a single transition $t \in N_1$. By assumption the transition t

belongs to one of the distinguished basic components FS_B. We consider the composition of projections

$$FS \xrightarrow{\pi_1} FS_1 \xrightarrow{\pi_B} FS_B$$

According to Proposition 6 the occurrence sequence $\pi_B(\sigma_1)$ lifts against the composition $\pi_B \circ \pi_1 : FS \longrightarrow FS_B$ to an occurrence sequence σ of FS. Therefore σ is also a lift of σ_1 against π_1. The lifting property of $\pi_1 : FS \longrightarrow FS_1$ and the liveness of FS imply that FS_1 is live too. Safeness of FS_1 follows from the fact that FS_1 is a union of basic components of FS and that each of them is also a basic component of FS_1, q. e. d.

The first new concept of this paper is the concept of a *non-blocking component*. It is a maximal well-behaved and non-blocking subsystem of a well-behaved free-choice system.

8. Definition (Non-blocking component)

Consider a well-behaved free-choice system $FS = (N, \mu)$.

i) For a connected subnet $NB \subseteq N$ the restriction

$$NS := (NB, \mu_B),\ \mu_B := \mu \mid NB,$$

is named a *non-blocking component* of FS, iff NS is

- a union of basic components of FS and
- non-blocking and
- maximal with respect to these two properties, i.e. no subsystem of FS exists with these properties and containing NS as a proper subsystem.

ii) A family $(NS_i)_{i \in I}$ of non-blocking components NS_i of FS, $i \in I$, with

$$FS = \bigcup_{i \in I} NS_i$$

is named a *non-blocking covering* of FS.

Apparently any well-behaved free-choice system has a non-blocking covering because each basic component is non-blocking. In addition, each non-blocking component of FS is well-behaved itself due to Corollary 7. This will be a crucial means for the construction in Definition 14.

A covering of a free-choice system is named *unshortenable* if no proper subfamily is a covering too. Each covering contains an unshortenable covering as a subfamily: After successively cancelling covering elements contained in the union of other elements we eventually obtain an unshortenable covering.

9. Example (Non-blocking covering)

The well-behaved blocking free-choice system *FS* from Figure 1 has an unshortenable non-blocking covering with two non-blocking components, cf. Figure 2. One non-blocking component is the union of three different basic components while the other non-blocking component is a single basic component.

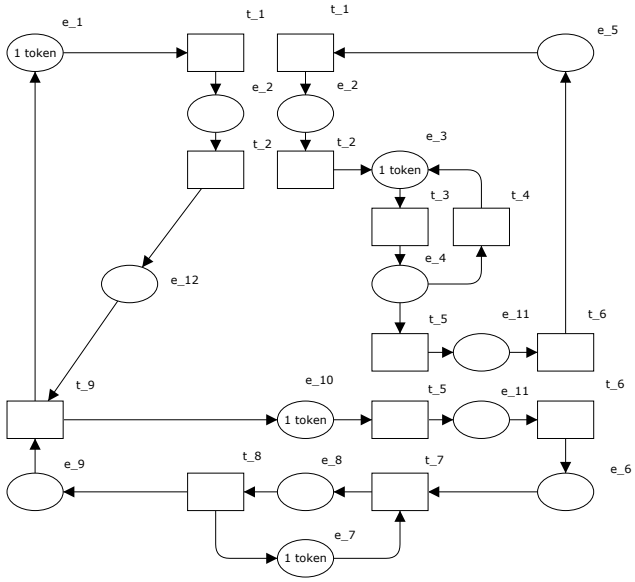


Figure 2: Two non-blocking components of the free-choice system from Figure 1

3. Linked bp-systems

Bp-systems are a simple class of coloured Petri nets. As mentioned in the *Introduction* they can be used to define the Boolean semantics of AND/XOR-EPCs. For the present paper we do not need the concept of coloured Petri nets in full generality, the interested reader is referred to [Jen1992].

10. Definition (bp-system)

i) A *bipolar synchronization graph* (bp-graph) *BG* is a coloured net. It extends a T-net $N = (P, T, F)$ by attaching to each place $p \in P$ the fixed set

$$C(p) = Boole := \{ high, low \}$$

with two token colours and provides each transition $t \in T$ with one from two types of logic:

- An AND-transition $t = t_{AND}$ has a set of firing modes $B(t) = \{ high, low \}$ with two elements: The high mode (respectively low mode) is enabled iff all pre-places of t_{AND} are marked with at least one high token (respectively low token).

Its firing consumes one high token (respectively low token) from each pre-place and creates one high token (respectively low token) on every post-place.

- An XOR-transition $t = t_{XOR}$ with n pre-places and m post-places has a set of firing modes $B(t) = \{b_{(i,j)}\}$ with $n \cdot m$ high modes and one low mode: The high mode with index $(i,j), 1 \le i \le n, 1 \le j \le m$, is enabled iff the i-th pre-place is marked with at least one high token and all other pre-places with at least one low token. Firing the high mode consumes a high token from the i-th pre-place and a low token from every other pre-place and creates a high token at the j-th post-place and a low token at every other post-place. The low mode is enabled iff all pre-places are marked with at least one low token. Firing the low mode consumes a low token from each pre-place and creates a low token at every post-place.

Adhering to the common notation of coloured nets we call a pair (p,c) with $p \in P, c \in C(p)$, a *token element* and a pair (t,b) with $t \in T, b \in B(t)$, a *binding element*. A binding element is named *low binding element*, if its firing consumes and creates only low tokens. Otherwise it is named *high binding element*.

ii) A *bipolar synchronization system* (bp-system) is a coloured Petri net $BS = (BG, \mu)$ with a bp-graph BG and an initial marking μ with at least one high token.

Bp-systems are a special case of Boolean systems which have been introduced in [LSW1998].

11. Definition (Well-behavedness of a bp-system)

i) A bp-system BS is *safe* iff each reachable marking marks every place with at most one token.

ii) A binding element of a bp-system BS is *live* iff for every reachable marking μ_1 of BS the bp-system (BG, μ_1) has a reachable marking which enables the given binding element. BS is *live with respect to all its high bindings* iff every high binding element of BS is live.

iii) A bp-system BS is *well-behaved* iff it is safe and live with respect to all its high bindings.

In a previous paper [Weh2010], Chap. 2, we have attached several ordinary Petri nets to a given bp-system $BS = (BG, \mu)$. Notably, a bp-system BS has a restricted free choice system

$$BS^{high} = \left(BG^{high}, \mu^{high} \right),$$

the *high-system* of BS, together with a morphism $high : BS \longrightarrow BS^{high}$ as well as a T-system

$$BS^{skel} = \left(BG^{skel}, \mu^{skel} \right),$$

the *skeleton* of BS, together with a morphism $skel : BS \longrightarrow BS^{skel}$.

Conversely, each restricted free-choice system FS extends to a bp-system BS with $BS^{high} = FS$. Hereby one introduces an AND-transition of BS for a branched transition of FS, an XOR-transition of BS for a branched place of FS and a high token of BS for each token of FS.

12. Example (Well-behaved bp-systems)

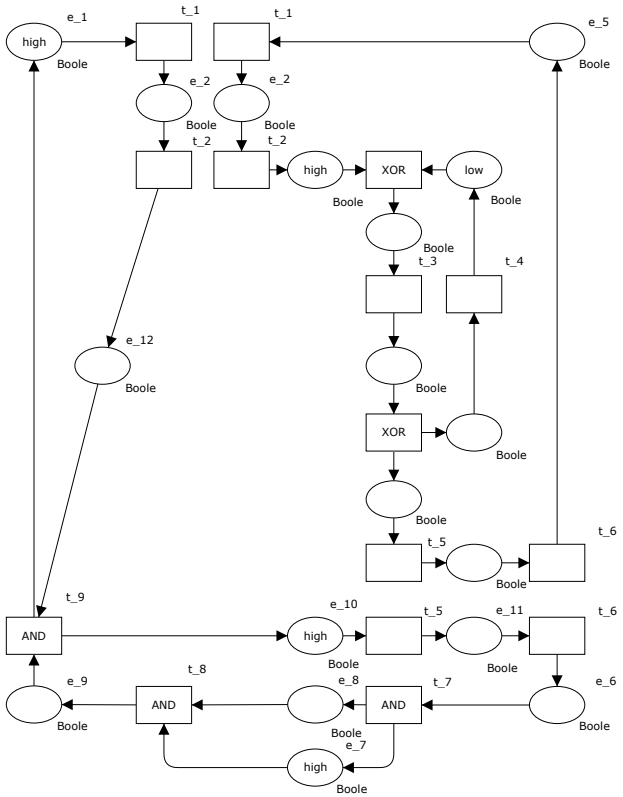


Figure 3: Bp-systems of the non-blocking components from Figure 2

Figure 3 shows two bp-systems $BS_i, i = 1,2$. Both are well-behaved. Their high-systems BS_i^{high} are the two non-blocking components from the well-behaved free-choice system from Figure 2. Note the low token marking the post-place of transition t_4.

The following Proposition 13 shows the relation between well-behaved bp-systems and well-behaved restricted free-choice systems. The proposition has been proven in [Weh2010].

13. Proposition (Bipolar systems and non-blocking free-choice systems)

i) A bp-system is well-behaved iff its skeleton and its high system are well-behaved and its high-system is non-blocking.

ii) The high-morphism $high : BS \longrightarrow BS^{high}$ of a well-behaved bp-system BS has the lifting property.

iii) Any well-behaved, non-blocking restricted free-choice system FS is the high-system of a well-behaved bp-system BS.

To obtain a marking such that the bp-scheme BS in Proposition 13, iii) is well-behaved, possibly some low tokens have to be added in addition to the high tokens prescribed by the marking of FS.

On the other hand, if a restricted free-choice system FS is well-behaved but blocking, no well-behaved bp-system BS exists with $BS^{high} = FS$.

It is our particular concern in this paper to remedy this situation. Therefore we will apply Proposition 13, iii) separately for each element from a non-blocking covering $(FS_i)_{i \in I}$ of FS: For each non-blocking component FS_i we obtain a well-behaved bp-system BFS_i with $BFS_i{}^{high} = FS_i$. For each pair of bp-systems (BFS_i, BFS_j) we fuse those subsystems of BFS_i and BFS_j which project along the high morphisms onto the same subsystem of FS. In the Petri net BS_i, which results from BFS_i, we consider the low tokens from BFS_i to belong to BS_i exclusively. For each pair (FS_i, FS_j) of non-blocking components of FS we substitute each branched transition $t^{high} \in \partial (FS_i \cap FS_j) \subset FS$ from the boundary by a transition which fuses the corresponding bp-systems (BS_i, BS_j). The fusing transition has to satisfy the following requirements:

- When firing it consumes and creates high tokens from BS_i and BS_j in the same manner as t^{high} processes tokens from FS_i and FS_j.
- It consumes and creates low tokens from BS_i without synchronizing them with high tokens or with low tokens from BS_j. Analogously it consumes and creates low tokens from BS_j.

The resulting coloured Petri net is named a *linked bipolar system* (bp-system). It is the second new concept introduced in this paper.

14. Definition (Linked bp-system)

Consider a well-behaved free-choice system FS and a covering $(FS_i)_{i \in I}$ of FS by non-blocking components. According to Corollary 7 each non-blocking

component FS_i, $i \in I$ is a well-behaved free-choice system. It is the high-system of a well-behaved bp-system BFS_i, and the high-morphism $high_i : BFS_i \longrightarrow FS_i$ has the lifting property according to Proposition 13. We define a coloured Petri net

$$LBS := \frac{\dot{\bigcup}_{i \in I} BFS_i}{(high_i)_{i \in I}}$$

by forming the quotient of the disjoint union of the bp-systems BFS_i, $i \in I$, modulo the identification with respect to the family of high morphisms $high_i : BFS_i \longrightarrow FS_i$. The coloured Petri net LBS is named a *linked bipolar-system (bp-system) attached to* FS *with respect to the covering* FS_i, $i \in I$. The high morphisms induce a well-defined morphism of Petri nets

$$high : LBS \longrightarrow FS .$$

Note that for each index $i \in I$ a projection $\pi_i : BFS_i \longrightarrow LBS$ onto the quotient exists. The image is a transition bounded subsystem

$$BS_i := \pi_i(BFS_i) \subset LBS .$$

Using the notations

$$LBS = (LBN, \mu),\ BFS_i = (BFN_i, \mu_i),\ BS_i = (BN_i, \mu_i),$$
$$FS = (FN, \nu),\ FS_i = (FN_i, \nu_i)$$

Definition 14 of a linked bp-system LBS can be made explicit as follows:

- Nodes LBX : Two nodes $x_i \in BFX_i$ and $x_j \in BFX_j$ fuse to a node $x \in LBX$ iff $high_i(x_i) = high_j(x_j) \in (FX_i \cap FX_j) \subset FX$. A well-defined map $high : LBX \longrightarrow FX$ results. We define

  $$I(x) := \{\, i \in I : x \text{ has a representative } x_i \in BFX_i \,\}.$$

- Token colours of LBN : A place $p \in LBX$ gets the set of token colours

  $$C(p) := \{\, high \,\} \cup \{\, low_i : i \in I(p) \,\} .$$

- Bindings and firing rules of LBN : For a transition $t \in LBX$ the binding set $B(t)$ has as low bindings the low bindings of all representatives $t_i \in BFN_i, i \in I(t)$, of t, each taken with its firing rule.
  On the other hand, the high bindings in $B(t)$ correspond bijectively to the high bindings of one arbitrary t_i. Each high binding gets an unchanged flow of high

tokens. If all representatives t_i have logical type AND, then the firing rule of a high binding of t does not consider any low tokens. When all representatives t_i have logical type XOR, then the firing rule of a high binding may change the flow of low tokens: When a high binding of t_{i_0} consumes a single low token of type low_{i_0} at a pre-place of t_{i_0} or creates a single low token of type low_{i_0} at a post-place, then the corresponding high binding from $B(t)$ respectively consumes and creates all low tokens of type $low_i, i \in I(t)$, at the corresponding place of LBN.

- Initial marking of LBS : At a place $p \in LBN$ the initial marking μ is defined as

$$\mu(p) := v(\, high(p)\,) + \sum_{i \in I(p)} \mu_i^{low}(p_i) \in C(p)_N .$$

- High morphism: For each index $i \in I$ the morphism $high_i : BFS_i \longrightarrow FS_i$ induces a morphism $high_i : BS_i \longrightarrow FS_i$ from the quotient $BS_i := \pi_i(BFS_i)$. These local morphisms fuse to a global morphism $high : LBS \longrightarrow FS$, such that

$$\begin{array}{ccc} LBS & \longrightarrow & LBS_i \\ \downarrow high & & \downarrow high_i \\ FS & \longrightarrow & FS_i \end{array}$$

commutes for all $i \in I$, the horizontal maps being the restrictions onto closed subsystems.

Figure 4 displays the most simple case of linking two binary bp-systems BFS_i, $i = 1, 2$, at a transition $t \in \partial(BN_1 \cap BN_2)$. The token colours of its pre- and post-places are

$$Boole_i = \{\, high, low_i \,\}, i = 1, 2 \text{ , and } Boole_{12} = \{\, high, low_1, low_2 \,\}.$$

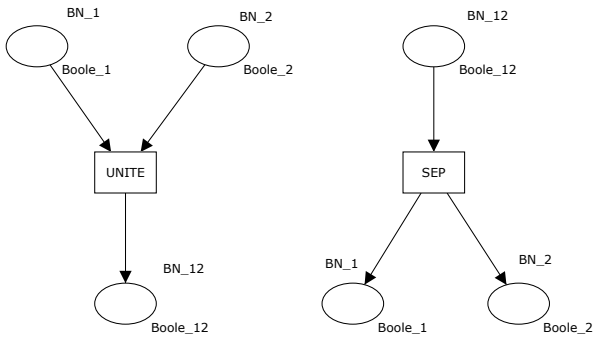


Figure 4: Transitions of type UNITE and SEP

Figure 4 shows on the left the neighbourhood of a fused closing transition of logical type UNITE and on the right of an opening transition of logical type SEP (separate).

Table 1 shows the corresponding firing rules of all transitions from $BN_{12} := BN_1 \cap BN_2$. Note in particular the different types of low tokens and their flow.

Transition	Bindings	consumes	produces
$t \in \partial(BN_1 \cap BN_2)$ of type SEP	$high$	$high$	$(high, high)$
	low_1	low_1	$(low_1, -)$
	low_2	low_2	$(-, low_2)$
UNITE: reverse SEP			
$t \in (BN_1 \cap BN_2)^\circ$ from opening XOR	$high_{left}$	$high$	$(high, low_1 + low_2)$
	$high_{right}$	$high$	$(low_1 + low_2, high)$
	low_1	low_1	(low_1, low_1)
	low_2	low_2	(low_2, low_2)
closing XOR: reverse opening XOR			
$t \in (BN_1 \cap BN_2)^\circ$ from opening AND	$high$	$high$	$(high, high)$
	low_1	low_1	(low_1, low_1)
	low_2	low_2	(low_2, low_2)
closing AND: reverse opening AND			

Table 1: Binding elements of BN_{12}

15. Example (Linked bp-system)

Figure 5 shows the linked bp-system attached to the well-behaved free-choice system from Example 3 and its non-blocking covering from Example 9.

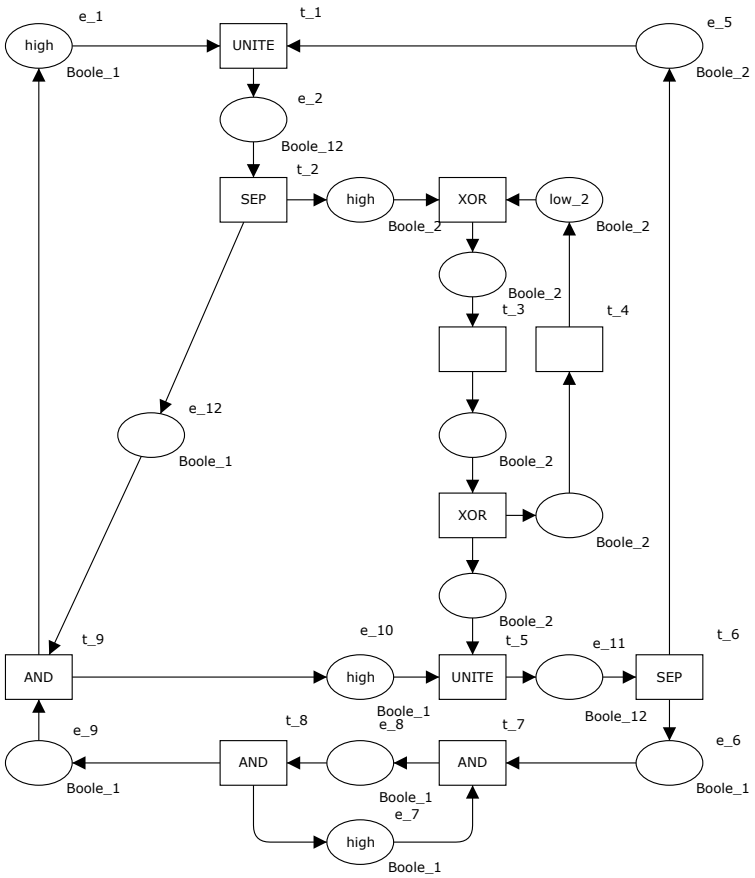


Figure 5: Linked bp-system attached to the free-choice system from Figure 1

16. Remark (Non-uniqueness of a linked bp-system)

According to the construction from Definition 14 a well-behaved free-choice system $FS = (FN, \nu)$ has more than one linked bp-system $LBS = (LBN, \mu)$ in general.

A first reason for non-uniqueness is the choice of a non-blocking covering $\mathcal{N} = (NS_i)_{i \in I}$ of FS. In general FS has more than one non-blocking covering which is non-shortenable. Therefore the underlying net $LBN = LBN(FS, \mathcal{N})$ depends not only on FS but also on $\mathcal{N}$.

This kind of dependency is similar to other situations from mathematics. E.g., compare the definition of a differentiable manifold, which is considered a pair $(X, \mathcal{A})$ formed by a topological space X and a maximal differentiable atlas $\mathcal{A}$ on X. But different from the situation of differentiable manifolds two non-blocking coverings $\mathcal{N}_1$ and $\mathcal{N}_2$ of a well-behaved free-choice system FS are compatible with

each other: Their union $\mathcal{N}_1 \cup \mathcal{N}_2$ is a non-blocking covering of FS again. Employing the definition of morphisms between coloured Petri nets from [Weh2006] one can show that the net of the corresponding linked bp-system is the fibre product

$$LBN(FS, \mathcal{N}_1 \cup \mathcal{N}_2) = LBN(FS, \mathcal{N}_1) \times_{FS} LBN(FS, \mathcal{N}_2)$$

with respect to the high-morphisms

$$high_i : LBN(FS, \mathcal{N}_i) \longrightarrow FN, i = 1,2 .$$

As a consequence the covering $\mathcal{N}_{\max}$ formed by all non-blocking components of FS is the unique maximal non-blocking covering of FS and one can define $LBN(FS, \mathcal{N}_{\max})$ as the underlying net of any linked bp-system of FS.
A second reason for non-uniqueness is the choice of the low tokens when considering a fixed non-blocking component FS_i of FS. In general more than one marking μ_i exists with $BFS_i = (BFN_i, \mu_i)$ well-behaved and $BFS_i^{high} = FS_i$. Two different markings differ by the distribution of low tokens. As a consequence, there may exist more than one marking μ on $LBN = LBN(FS, \mathcal{N}_{\max})$ with $LBS = (LBN, \mu)$ a linked bp-system of FS.

The following Theorem 17 and its corollary Proposition 19 are the main results of the present paper. They prove that any linked bp-system of a well-behaved free-choice system is well-behaved too.

17. Theorem (High morphism of a linked bp-system)
Consider a well-behaved free-choice system FS and a non-blocking covering $(FS_i)_{i \in I}$ of FS. The high morphism

$$high : LBS \longrightarrow FS$$

from a linked bp-system LBS attached to FS with respect to $(FS_i)_{i \in I}$ has the lifting property.
Proof. We use the following notations from Definition 14

$$LBS = (LBN, \mu),\ BFS_i = (BFN_i, \mu_i),\ FS = (FN, \mu^{high}),\ FS_i = (FN_i, \mu_i^{high})$$

and denote by

$$pr_i : FN \longrightarrow FN_i$$

the canonical projection.
In order to prove the lifting property of $high : LBS \longrightarrow FS$ we start considering an occurrence sequence

$$\mu^{high} \xrightarrow{\sigma^{high}} \nu^{high}$$

of FS. Without loss of generality we may assume that σ^{high} comprises only a single transition $r \in FN$. For each index $i \in I$ we define the transition $r_i := pr_i(r) \in FN_i$. The condition $high(t,b) = r$ determines a unique transition $t \in LBN$ and a unique high binding $b \in B(t)$ of t. Analogously, for each index $i \in I$ the condition $high_i(t_i, b_i) = r_i$ determines a unique transition $t_i \in BFN_i$ and a unique high binding $b_i \in B(t_i)$ of t_i. In addition, according to Proposition 13 an occurrence sequence

$$\mu_i \xrightarrow{\sigma_i^{low}} \tilde{\mu}_i$$

exists in the low-system BFS_i^{low} such that the marking $\tilde{\mu}_i$ of BFS_i activates the binding element (t_i, b_i). By catenation we obtain an occurrence sequence

$$\mu_i \xrightarrow{\sigma_i} \nu_i$$

of BFS_i with $\sigma_i := \sigma_i^{low} \cdot (t_i, b_i)$. Because $BS_i^{low} \subset LBS$ is a place bounded subsystem the occurrence sequence $\pi_i(\sigma_i^{low})$ of BS_i^{low} can be considered an enabled occurrence sequence of BS. Firing $\pi_j(\sigma_j^{low})$ for an arbitrary index $j \neq i$ does not remove the firing concession from $\pi_i(\sigma_i^{low})$. By catenation we obtain an occurrence sequence of LBS

$$\mu \xrightarrow{\sigma^{low}} \tilde{\mu} \text{ with } \sigma^{low} := \pi_1(\sigma_1^{low}) \cdot \ldots \cdot \pi_n(\sigma_n^{low})$$

Due to Definition 14 the occurrence sequences

$$\tilde{\mu}_i \xrightarrow{(t_i, b_i)} \nu_i, \; i \in I,$$

of all bp-systems BFS_i link to an occurrence sequence

$$\tilde{\mu} \xrightarrow{(t,b)} \nu$$

of LBS. By catenating $\sigma := \sigma^{low} \cdot (t,b)$ we obtain the occurrence sequence of LBS sought-after

$$\mu \xrightarrow{\sigma} \nu$$

satisfying $high(\sigma) = \sigma^{high}$, q.e. d.

For a linked bp-system the definition of *liveness with respect to all its high bindings* is literally the same as in Definition 11, part ii) for a bp-system.

18. Definition (Well-behavedness of linked bp-systems)

Consider a linked bp-system LBS attached to a well-behaved free-choice system FS and a non-blocking covering of FS with $k \geq 1$ elements.

i) LBS is *high-safe* iff every reachable marking of LBS marks each place either with a single high token but no low token or with at most k low tokens but no high token.

ii) LBS is *well-behaved* iff it is high-safe and live with respect to all its high bindings.

19. Proposition (Well-behavedness of a linked bp-system)

Any linked bp-system LBS attached to a well-behaved free-choice system FS with respect to a non-blocking covering $(FS_i)_{i \in I}$ is well-behaved.

Proof. With the notations of Definition 14 we set

$$LBS = (LBN, \mu_0),\ FS = (FN, high(\mu_0)) = LBS^{high} \text{ and } (BFS_i)_{i \in I}.$$

i) High-safeness of LBS follows from the existence of the morphism $high : LBS \longrightarrow FS$ and the safeness of each $BFS_i, i \in I$.

ii) For the proof that LBS is live with respect to all high bindings we employ the lifting property of $high : LBS \longrightarrow FS$. We consider an occurrence sequence $\mu_0 \xrightarrow{\sigma_1} \mu_1$ and a transition $t \in LBN$ with a high binding $b \in B(t)$. By definition the binding element (t,b) of LBN is a transition of FN. Liveness of FS implies the existence of an occurrence sequence

$$high(\mu_1) \xrightarrow{\sigma_2{}^{high}} \mu_2{}^{high}$$

such that the marking $\mu_2{}^{high}$ activates (t,b). According to Theorem 17 the occurrence sequence $\sigma_2{}^{high}$ lifts to an occurrence sequence $\mu_1 \xrightarrow{\sigma_2} \mu_2$ such that the following diagram commutes

$$\begin{array}{ccccc} \mu_0 & \xrightarrow{\sigma_1} & \mu_1 & \xrightarrow{\sigma_2} & \mu_2 \\ \downarrow high & & \downarrow high & & \downarrow high \\ high(\mu_0) & \xrightarrow{high(\sigma_1)} & high(\mu_1) & \xrightarrow{\sigma_2{}^{high} = high(\sigma_2)} & \mu_2{}^{high} = high(\mu_2) \end{array}$$

Therefore LBS is live with respect to all high bindings, q. e. d.

20. Remark (Semantics of AND/XOR-EPCs)

Let EPC be an AND/XOR-EPC. As described in the *Introduction* a free-choice system FS exists, which defines the free-choice semantics of EPC. Also a translation of EPC into a bp-system BS exists. We have $BS^{high} = FS$.

i) If FS is non-blocking, then BS is well-behaved iff FS is well-behaved, cf. Proposition 13.
ii) If FS is well-behaved, then the Boolean semantics of EPC is defined by a well-behaved linked bp-system LBS with $LBS^{high} = FS$, cf. Proposition 19.
iii) If FS is non-blocking and well-behaved then $LBS = BS$, cf. Definition 14.

4. Outlook

We have shown that any AND/XOR-EPC, which is well-behaved with respect to its free-choice semantics, can be provided with a Boolean semantics which is well-behaved too. In order to obtain this result, we had to generalize bp-systems to linked bp-systems, a class of coloured Petri nets which is slightly more general.
This step is necessary to tackle EPC with connectors of arbitrary logical type. Apparently one can translate every EPC literally into a Boolean system. But Example 15 indicates that the literal translation possibly has to be altered afterwards to avoid the blocking of low tokens.
Future investigations have to consider the literal translation of an EPC into a Boolean system only as a starting point. For the next step we need an algorithm which identifies well-behaved components of the Boolean system. Then it should link these components to a linked Boolean system, which avoids the blocking of low tokens from different components. These well-behaved components generalize the bp-systems of non-blocking components while linked Boolean systems generalize the concept of linked bp-systems introduced in the present paper.

5. References

[Aal1999] *van der Aalst, W.M.P.*: Formalization and verification of event-driven process chains. Information & Software Technology, 41 (10), 1999, p. 639-650

[DE1995] *Desel, Jörg, Esparza, Javier*: Free Choice Petri Nets. Cambridge University Press, Cambridge 1995

[GT1984] *Genrich, H.J.; Thiagarajan, P.S.*: A Theory of Bipolar Synchronization Schemes. Theoretical Computer Science 30 (1984), p. 241-318

[Jen1992] *Jensen, Kurt*: Coloured Petri nets. Basic concepts, analysis methods and practical use. Vol. 1. Springer, Berlin et al. 1992

[KNS1992] *Keller, Gerhard; Nüttgens, Markus; Scheer, August-Wilhelm*: Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)". Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89, Saarbrücken 1992

[LSW1998] *Langner, Peter; Schneider, Christoph; Wehler, Joachim*: Petri Net based Certification of Event-driven Process Chains. In: Desel, Jörg; Silva, Manuel (Eds.): Application and Theory of Petri Nets 1998. Lecture notes in Computer science, vol. 1420. Springer, Berlin et al. 1998

[Sch1994] *Scheer, August-Wilhelm*: Business Process Reenginering. Reference Models for Industrial Enterprises. Berlin, 2ed 1994

[TV1984] *Thiagarajan, P.S.; Voss, Klaus*: A Fresh Look at Free Choice Nets. Information and Control 62 (1984), p. 85-113

[Weh2006] *Wehler, Joachim*: Morphisms of Coloured Petri Nets, arXiv, cs.SE/0608038, 2006

[Weh2010] *Wehler, Joachim*: Free-Choice Petri Nets without frozen tokens, and Bipolar Synchronization Systems. Fundamenta Informaticae 98 (2010), p. 283-320

Modular and Hierarchical Modelling Concept for Large Biological Petri Nets Applied to Nociception

Mary Ann Blätke[1] and Wolfgang Marwan[1]

[1]Magdeburg Centre for Systems Biology (MaCS), Otto-von-Guericke Universität Magdeburg, Universitätsplatz 2, 39106 Magdeburg, Germany
marwan@mpi-magdeburg.mpg.de

Abstract Here, we introduce a modular and hierarchical modeling concept for large biological Petri nets. This modeling concept suggests representing every functional system component of a molecular network by an autonomous and self-contained Petri net, so-called module. Due to the specific architecture of the modules, they need to fulfill certain properties important for biological Petri nets to be valid. The entire network is build-up by connecting the modules via common places corresponding to shared molecular components. The individual modules are coupled in a way that the structural properties that are common to all modules apply to the composed network as well. We applied this modeling concept on nociceptive signaling in DRG-neurons to compose a model describing pain on a molecular level for the first time. We verified the applicability of our modeling concept for very complex components and confirmed preservation of the properties after module coupling.

1 Introduction

A major issue in systems biology is the construction and validation of large biological networks, especially if the involved mechanisms should be considered in depth. This is the case for the nociceptive network in the peripheral endings of DRG-neurons (nociceptors) that are responsible for pain signaling (see Figure 1). Pain is a very complex phenomenon with behavioral, peripheral and central nervous system components, wherein nociception comprises the underlying molecular mechanisms [2]. (Chronic) pain is certainly one of the most serious public health issues (see [3,6] and references therein).Hitherto, there exists no coherent computational model for pain due to the complexity and lack of knowledge on the underlying molecular mechanisms. A complete and validated pain model would be an important progress to develop a mechanism-based pain therapy to successfully treat pain suffering.

In general, modular approaches have always been useful to manage large networks. So far, in systems biology just single pathways have been regarded as modules [1]. Our modular and hierarchical modeling concept is beyond this scope. It is a promising approach to handle large biological systems by treating

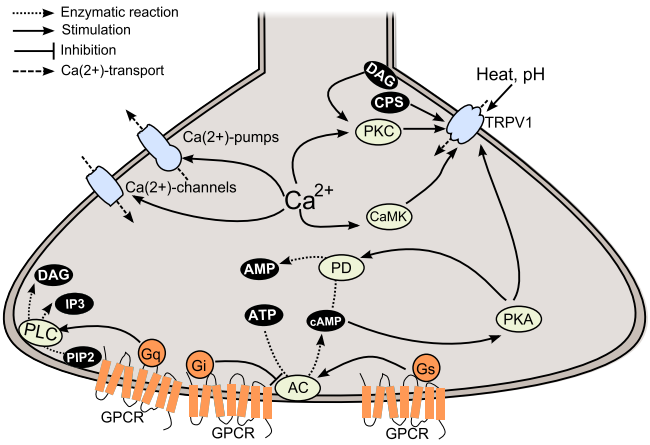


Figure 1. Illustration of a nociceptor. Primary subunits (modules) of the nociceptive network are enzymes (green), receptors (orange) and channels (blue). The secondary subunits like cAMP, Ca2+, DAG etc. are colored in black.

functional molecular components as single independent entities. In this respect, Petri nets are an appropriate tool. They are designed for concurrent systems. Thus, Petri nets are ideally suited to describe biological systems [5], like the nociceptive system. Also, they allow for a hierarchical arrangement of large and complex networks in the form of a neat graphical representation. Single functional proteins (receptors, channels, enzymes etc.) are represented by hierarchical, autonomous and self-contained Petri nets, called modules, which have to fulfill certain properties important for biological Petri nets [5]. Those firstly qualitative modules are validated by a comprehensive analysis and are subsequently subjected to stochastic simulation studies. The modular and hierarchical modeling concept implies a special coupling procedure of the modules to an entire network of communicating components. Advantageously, the properties of the entire network can be predicted due to the adhered properties of the single modules and the special module coupling.

The constructed pain model is a first approach to integrate the currently published neurobiological and clinical knowledge about nociception in one coherent and validated model describing all the interactions between the involved components. Hitherto, it contains 31 modules that have been constructed and connected by the modular and hierarchical modeling concept (see also section "'Nociceptive Network"'). For the construction and validation of the modules and the entire network we used the Petri net editor *Snoopy* [9] and the place/transition analysis tool *Charlie* [7].

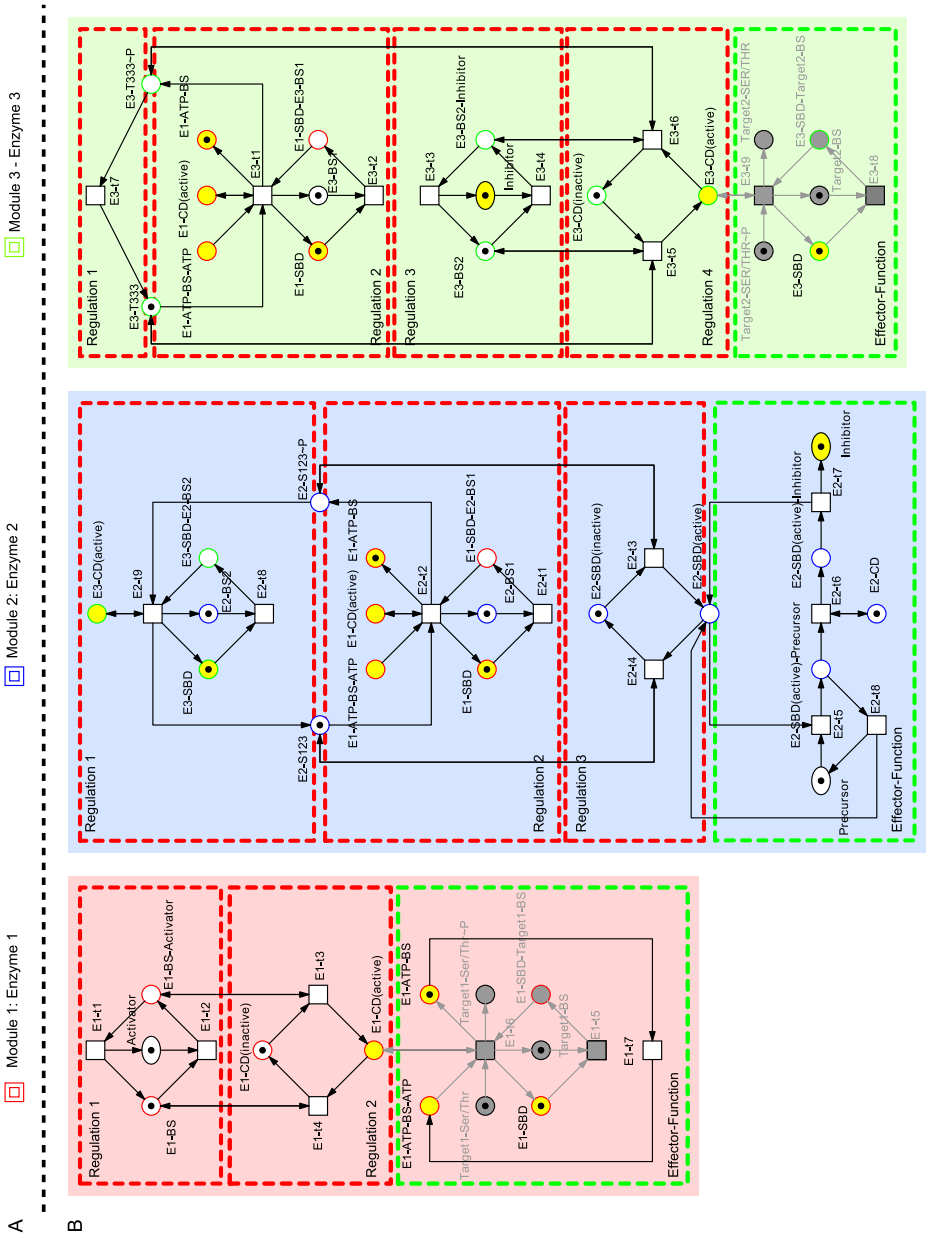


Figure 2. Example of a small toy network composed of three different enzymes to explain the method: enzyme 1 (Kinase, stimulated by the activator), enzyme 2 (Synthase for the inhibitor, regulated by phosphorylation), enzyme 3 (Phosphatase, regulated by the inhibitor and phosphorylation). (A) Top-level of the entire network containing three modules wrapped in coarse transition. (B) Flat representation of the network graphs of the modules showing enzyme 1 (red), enzyme 2 (blue) and enzyme 3 (green) and places are framed with the corresponding color. Circles indicate places belonging to the primary entities and oval places indicate secondary entities (activator, inhibitor and precursor). The modules consist of regulative subnets (red dashed rectangles') and subnets of effector function (green dashed rectangles). Logical places (yellow) connect the modules at deeper levels of the hierarchy tree. The grey places, transitions and arcs have been deleted after coupling.

2 Method

First, the identified components in the regarded system have to be categorized in primary and secondary entities. Primary entities are proteins or protein complexes (enzymes, receptors, ion channels, adaptor proteins etc.), whose function and activity can be regulated due to modification by other components. Secondary entities cannot undergo modifications of their activity and function. This group contains ligands, second messengers, precursor molecules, ions and energy equivalents. Secondary entities are regulators or substrates of primary entities or they are transported by those. Primary entities can be further differentiated by their function, whether they regulate other primary entities or process secondary entities. Every primary entity constitutes a module that contains a hierarchical arranged, autonomous and self-contained Petri net. Detailed information about the introduced modeling concept can be found in reference [4].

Architecture of a Module. Places of a module correspond to different states of functional domains of primary entities (phosphorylation sites, binding domains, inhibitory sequences etc.) or different states of secondary entities (free or bound, precursor or proceeded molecule etc.). In this context, transitions of a module describe inter-/intramolecular actions that occur within the corresponding primary entity (like binding/dissociation, (de-)phosphorylation, conformational changes or processing of substrates etc) and change the states of the involved entities. Every module contains two classes of subnets indicating the regulation or the effector function of a primary entity. The effector function subnets of those primary entities that might regulate a variety of other primary entities are generalized. The possible targets are fused to one abstract target. Such subnets can be reused for the construction of the regulatory subnets of discrete targets. An illustrative example of a regulative network consisting of three different enzymes is shown in Figure 2.

Validation of a Module. The constructed modules have to fulfill certain properties important for biological networks [5] to be valid which are considered by a comprehensive analysis. Table 1 gives an overview about the properties that every module must fulfill (see also Figure 3A). Having successfully validated the qualitative modules, they are subjected to a stochastic simulation, even if experimental parameters are not available so far. Simulation studies are carried out to analyze whether the dynamic behavior of the modules can in principle reflect the assigned effector function as indicated by the time-dependent token-flow. A stochastic mass action function is assigned to every transition that can be modulated by a parameter according to biological needs. The parameters are determined by 'in silico' experiments.

Assembling of the Modules to an Entire Network. The single modules can easily be connected to a larger network. The prerequisites for the direct and indirect coupling of the modules have been established separately. The subnets of the modules already consider all possible interactions. Thus, the modules are 'naturally' connected by places that are equivalent to complexes between the different entities (indicated as logical places) and actions on which the different entities participate. At the top-level of the entire network the modules are just

visible as coarse transitions. Thus, the connection of the modules is not immediately obvious and the network seems to be very compact. Due to logical places the complex branching of the modules is only visible on lower levels. The effector function subnets of primary entities showing the regulation of a variety of other primary entities are not needed anymore. Therefore, all places corresponding to abstract targets and transitions connected with abstract targets have to be be abolished. The entire network already contains all specified targets of those primary entities. Figure 2 shows also the coupling of the enzymes to an entire network.

Deducing Properties for the Entire Network from the Modules. Due to the way of coupling, it is possible to transfer the structural properties of the modules on the entire network (see Table 1A). We show that they do not change after the coupling procedure. The entire network still contains no boundary transitions but boundary places of secondary entities. Therefore, it cannot be covered with T-invariants. We observe that all T-invariants of the coupled modules are conserved in the entire network. Furthermore, the coverage of the entire network with P-invariants is achieved. Due to the special module coupling just certain actions can occur to the P-invariants. The P-invariants of each module can be retained or deleted without changing the coverage with P-invariants of the entire network. The retention of P-invariants can be divided in five cases: *(1)* Retention of unique P-invariants, *(2)* Melting of identical P-invariants, *(3)* Combination of overlapping P-invariants, *(4)* Deletion of states of abstract targets in a P-invariant and replacement by all possible specified targets, *(5)* Integration of P-invariants in retained P-invariants. A P-invariant that contains only states of an abstract target is deleted in the entire network, because the equivalent places have been deleted before. Due to the coverage of the entire network with P-invariants it is bounded. By virtue of boundness and the non-coverage with T-invariants the entire network cannot be live and reversible (see also Table 1B). After validating the entire network by its properties, the dynamic behavior must be investigated by simulation studies (see Figure 2B).

Table 1: Properties of the modules and the entire network

Properties	Fulfilled	Explanation
A - Structural Properties		
Pure	No	Every module contains actions that process just under certain intra-/intermolecular circumstances like a special state of a domain. The corresponding places of such domains are connected with the transition of an action by an double arc.
Ordinary	Yes	The arc weigth is "1" because just elementary actions are considered. Meaning just one element of every secondary entity and one state of every domain can attend on the educte side as well as on the product side.
Homogenous	Yes	Due to Ordinary.
Input transition	No	There are no boundary transitions (sinks or sources) that add or withdraw any tokens.
Output transition	No	

Input place	Yes[1]	The modules are bordered by places corresponding to domains of other primary entities or secondary entities.
Output place	Yes[1]	
Non-blocking multiplicity	No[1]	Due to boundary places this property cannot be fulfilled.
Conservative	No	Modules contain certain domains of primary substances that can build complexes with domains of the same or another primary substance as well as with secondary substances.
Static conflict free	No	Modules contain certain domains of primary entities and secondary entities that can attend on more than one action on the reactant side.
Connected	Yes	Every module must be connected, as well as the entire network.
Strongly Connected	No[1]	The boundery nodes preclude strong connectedness.
Covered with P-invariants	Yes	Every Module has to be covered with P-Invariants, because: – Every domain of a primary entity and every secondary entity must exist in one of the possible state. – There can just exist one of the possible states of a domain of a primary entity or a secondary entity at the same time. – There can just exist certain combinations of those states at the same time. Every P-Invariant has an important biological interpretation that contributes to the function of the module.
Covered with T-invariants	No[1]	Due to boundary places this property cannot be fulfilled. The same is valid for the entire network. But every T-Invariant has also an important biological interpretation that describes reversible processes like binding/dissociation, phosphorylation/dephosphorylation, activation/inactivation etc.)
Deadlock trap property	No[1]	Due to boundary places this property cannot be fulfilled. The same is valid for the entire network.
B - Behavioral Properties		
Structurally/ k-bounded	Yes	Due to the coverage with P-invariants the modules are bounded.
Strongly covered with T-invariants	No	Due to boundary places this property cannot be fulfilled. Also exist transitions describing two reverse actions.
Dead Transitions	No	The initial marking must assure that every action can proceed.
Dynamically conflict free	Yes[1]	Modules can contain actions that inhibit the feasibility of other actions.
Dead States	No[1]	Modules can contain actions that can act independent of the limitations by secondary entities.
Liveness	No[1]	Cannot be fulfilled because boundness and non-coverage with T-invariants.
Reversibility	No[1]	Due to boundary places this property cannot be fulfilled.

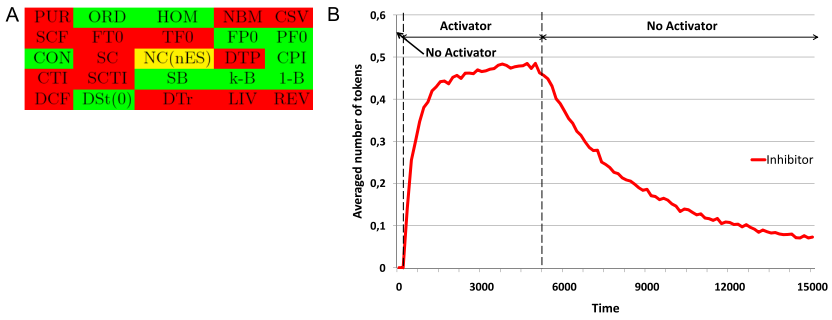


Figure 3. Validation of the modules and the entire network shown in Figure 1. (A) Identical properties of the modules and the entire network (exception: the module of enzyme 2 contains two dead states) determined with Charlie (red = no, green = yes). (B) Stochastic simulation study with the entire network showing the dependence of the inhibitor synthesis on the activator. The simulation result is conforming to the expected behavior, the inhibitor is mainly produced if the activator for enzyme 1 is available. The high amount of the inhibitor inactivates the antithetic enzyme 3.

3 Nociceptive Network

Currently, we have constructed 31 modules (see also figure 1) with the help of modular and hierarchical modeling concept on the basis of 320 scientific articles. All modules have been connected to an entire nociceptive network with a total size of 709 places, 800 transitions and 4391 arcs that are spread over 291 pages with a nesting depth of up to 4. The modules of nociceptive signaling components as well as the resulting nociceptive network have been validated. They adhere the given properties of the modular and hierarchical modeling concept. All modules and the entire nociceptive network as well as detailed results of the analysis and simulations studies can be found in reference [4].

4 Conclusion

With the help of the modular and hierarchical modeling concept we were able to construct and validate a number of modules of important nociceptive signaling components and assemble them to an entire nociceptive network [4]. Hitherto, the nociceptive network is not complete. Twice as many modules will be needed to describe all known interactions.
Nevertheless, we verified the applicability of our modeling concept even for very complex components and the preservation of the properties after module coupling.

[1] Exception for single modules are possible due to their functionality.

All constructed modules are well documented and organized in a library for reuse in other systems. The modules can be connected according to the specific demands of any 'wet lab' or 'in silico' experiments.
To investigate the whole nociceptive system with 'in silico' experiments, we first need to modularize the missing nociceptive components and parameterize the modules. We plan to establish a possible parameter set by trial and error. This parameter set can then be challenged by error analysis and model checking. With an initially parameterized nociceptive network we will presumably be able to: *(1)* investigate changes in network behavior on perturbations of the network, *(2)* predict experiments, *(3)* suggest possible targets for new intervention strategies in pain therapy based on sensitivity analysis. To investigate multiple copies of signaling components as well as diverse DRG-neuron population we also intend to color our low level net [8]. Further we want to extend reconstructed networks [10] out of experimental data by module mapping. We are still searching for new methods to screen the modules and the nociceptive network for non-obvious properties that are defined by their structure.
In summary, our modular and hierarchical modeling concept seems to be a promising way to handle and investigate large biological system, to develop new analysis approaches and Petri net applications.

5 Acknowledgements

This work is supported by the Modeling Pain Switches (MOPS) program of Federal Ministry of Education and Research (Funding Number: 0315449F). We thank Prof. Monika Heiner and Sonja Meyer for the outstanding support and cooperation during this work.

References

1. Alberghina, L. et al.: A modular systems biology analysis of cell cycle entrance into S-phase. Topic in Current Genetics 13 (2005)
2. McMahon, S. and Koltzenburg, M.: Textbook of Pain. Churchill Livingstong (2005)
3. Hucho, T. and Levine, J.: Signaling Pathways in Sensitization: Toward a Nociceptor Cell Biology. Neuron 55 (2007)
4. Blätke, M.-A.: Petri-Netz Modellierung mittels eines modularen and hierarchischen Ansatzes mit Anwendung auf nozizeptive Signalkomponenten (Diploma thesis). Otto von Guericke University Magdeburg (2010)
5. Heiner, M., Gilbert, D., and Donaldson, R.: Petri Nets in Systems and Synthetic Biology. In School on Formal Methods , Springer LNCS 5016 (2008)
6. Stein, C. and Lang, L.J.: Peripheral Mechanisms of Opioid Analgesia. Current Opinion in Pharmacology 9 (2009).
7. Franzke, A.: Charlie 2.0 - A Multithreaded Petri Net Analyzer (Master's thesis). Brandenburg University of Technology Cottbus (2009)
8. Liu, F, Heiner, M: Colored Petri nets to model and simulate biological systems; Int. Workshop on Biological Processes & Petri Nets (BioPPN), satellite event of Petri Nets 2010, Braga, Portugal, June 21 2010.

9. C Rohr, W Marwan, M Heiner: Snoopy - a unifying Petri net framework to investigate biomolecular networks; Bioinformatics 2010 26(7)
10. Marwan, W., Wagler, A. and Weismantel, R.: Petri nets as a framework for the reconstruction and analysis of signal transduction pathways and regulatory networks. Natural Computing (2009)

Computation of enabled transition instances for colored Petri nets

Fei Liu and Monika Heiner

Department of Computer Science, Brandenburg University of Technology
Postbox 10 13 44, 03013 Cottbus, Germany
{liu, monika.heiner}@informatik.tu-cottbus.de

Abstract. Computation of enabled transition instances is a key but difficult problem of animation of colored Petri nets. To address it in our colored Petri net tool, we give an algorithm for computing enabled transition instances. This algorithm is based on pattern matching. So it first tries to bind tokens to variables covered by patterns. If some variables are not covered by any pattern, the algorithm will bind all the colors in the corresponding color sets to the variables. This algorithm uses the new principle of partial binding - partial test and adopts some optimization techniques for preprocessing to improve efficiency. The principle of partial binding - partial test allows us to test the expressions during the partial binding process so as to prone invalid bindings as early as possible. The preprocessing with optimization techniques not only prunes a lot of invalid potential bindings before the binding begins, but also finds disabled transitions at an early phase.

1 Introduction

Animation is an important technique for getting an intuitive understanding of a Petri net model as it demonstrates the dynamic behavior of the model in a visual way. Nearly all the visual tools for modeling Petri nets provide the animation functionality [Pet10]. For low-level Petri nets, the core of the animation is the scheduling algorithm of the transitions. However, for colored Petri nets, we have to consider another key problem, the computation of enabled transition instances. When checking whether a transition is firable or not at a given marking, we have to assign values to the variables (which are called bindings. A binding of a transition corresponds to a transition instance.) that occur in the arc expressions and the guard of the transition, and then evaluate if it respects the firing rule.

The introduction of colors to Petri nets makes it difficult to compute their firing rules [JKW07]. The efficiency of the animation for large-scale colored Petri nets is mainly determined by the efficiency of the computation of the enabled transition instances, which, however, is a NP-hard search problem because of the expressiveness of colored Petri nets. One possible way is to make an exhaustive search to check all the bindings and then prune the invalid ones, which is

inefficient at all especially if the transitions have many variables, but only a few bindings can fire the transitions.

In this paper, we focus on the problem of the compuatation of enabled transition instances for colored Petri nets. We adopt the same idea given in [KC04], that is, extracting patterns from input arc expressions and guards, then binding the tokens residing on the input places to these patterns, and thus obtaining an enabled binding set. The main contribution of this paper is that we give a more efficient algorithm for our colored Petri net tool [RMH10], [LH10], in which we use a new principle of partial binding - partial test and several heuristics techniques to compute enabled transition instances.

This paper is organized as follows. Section 1 describes the patterns that are used in the computation of enabled transition instances, and discusses how to classify and find patterns. Section 2 discusses the computation process and recalls some concepts. Section 3 gives the computation algorithm. Section 4 discusses some heuristics to optimize the computation process. Section 5 summarizes and compares the related work. Section 6 gives the conclusion.

2 Patterns

We use the same pattern match mechanism as CPN tools [KC04]. A pattern is defined as an expression with variables which can be matched with other expressions to assign the values of variables [KC04]. The difference is that CPN tools are based on the SML, but we do not. We do not employ all the patterns defined in SML [Ull98], but a subset, as we use less data types than CPN tools. The patterns that we use have the following syntactical structure:

$$
\begin{aligned}
Pattern \quad & ::= "Variable" \\
& \quad |"Constant" \\
& \quad |TuplePattern \\
TuplePattern & ::= (Pattern(,Pattern)*)
\end{aligned}
$$

Consider the example illustrated in Figure 1. According to the syntax of the patterns, we can see that (x, y) is a tuple pattern. If we bind the token $(1, a)$ residing on the place $P2$ to the pattern (x, y), after matching, we get an assignment $x = 1$ and $y = a$. This process is called the pattern match.

Pattern matching provides an easy and efficient way to compute enabled transition instances; therefore, to improve the efficiency of the computation, we have to find the patterns and use them to bind the tokens on the places as much as possible. To do so, we have to search all the input arc expressions of a transition to find available patterns, which we call a pattern set concerning arc expressions, denoted by $AS(t)$ for a transition t. Besides, we also can search the guard of the transition t to find the patterns in the guard, denoted by $GS(t)$. These two sets constitute the overall pattern set, $PS(t) = AS(t) \cup GS(t)$, which are used to bind tokens to variables. In the following, we in detail discuss how to find the patterns.

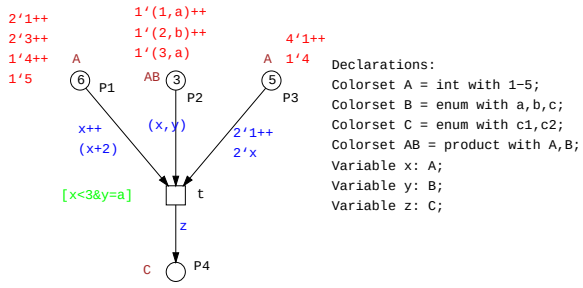


Figure 1. An example to demonstrate the patterns

2.1 Patterns in arcs

The patterns in arcs are the basic patterns that are used for the computation of enabled transition instances. To get them, we have to search though all the input arc expressions in the following two ways.

(1) If an input arc expression is exactly a pattern illustrated above, then we put it in the $PS(t)$. For example, in Figure 1, we can get such kind of pattern: (x, y).

(2) If an input arc expression is a multiset expression, having the form c_1‘$expr_1 + +... + +c_n$‘$expr_n$, where c_i is the multiplicity, and $expr_i$ has the type of its corresponding input place. If $expr_i$ is a pattern, then we add $expr_i$ to the $PS(t)$. For example, for the expression 2‘$1 + +2$‘x in Figure 1, we get two such kinds of patterns: the constant pattern, ’1’ and the variable pattern, x, but for the expression $x + +(x + 2)$, we only get one variable pattern, x, as $x + 2$ is not a pattern. At the same time, we record the multiplicity of the corresponding pattern so as to use them to test bindings once the pattern is used. For example, for the variable pattern x, once we bind a token to it, for instance ’4’ on the place $P3$, we immediately test if there are enough tokens with color ’4’ on the place $P3$. To do so, the invalid bindings can be discarded earlier.

2.2 Patterns in guards

As the guard of a transition imposes often a rather strong constraint on efficient bindings, it is better to consider it early when computing the bindings. For this, we adopt the similar appoach to that in [KC04], but we extend the forms of the guard that are used for binding, moreover we use some forms of guards for test during the binding process.

Like [KC04], we consider the guard in the conjunctive form, $G(t) \equiv \wedge_{i=1}^{n} g_i(t)$. For one of these conjuncts $g_i(t)$, we consider the following forms: $g_{il}(t) = g_{ir}(t)$, where $g_{il}(t)$ and $g_{ir}(t)$ are expressions with constant or variable patterns, but one of them must be a constant. We put these special expressions into the pattern set $GS(t)$. The advantage of using the patterns in the guards for binding is obvious. For example, consider the pattern, $y = a$, it directly makes the bindings relating to tokens without the color, ”a”, invalid.

2.3 Binding color sets to variables

If there are variables that are not covered by $PS(t)$, we have to let them bind to their corresponding color sets, otherwise they can not be bound. For example, for the variables that only appear in the output arcs, we have to bind them to their color sets. In Figure 1, we can see that the variable z is of this case, which has to be bound to the corresponding color set C.

2.4 Optimized pattern set

We herein give a formal representation of each pattern in $PS(t)$, which is a five-tuple $S = \langle P, E, X, M, m\rangle \in PS(t)$, where

- P, the type of the pattern: variable, constant, tuple, or guard,
- E, the expression that the pattern belongs to,
- X, the set of the variables in the pattern,
- M, the initial/current marking of the place that connects the arc whose expression the pattern belongs to, and
- m, the multiplicity of the pattern.

For the pattern of $AS(t) \subseteq PS(t)$, all the components above would be used, but for the pattern of $GS(t) \subseteq PS(t)$, only the first three components P, E, and X would be used. For a constant pattern, X will be always $\{\phi\}$.

For example, the patterns $PS(t)$ in Figure 1 can be formally written as follows:

$S_1 = \langle Variable, x, \{x\}, \{2`1, 2`3, 1`4, 1`5\}, 1\rangle$
$S_2 = \langle Tuple, (x, y), \{x, y\}, \{1`(1, a), 1`(2, b), 1`(3, a)\}, 1\rangle$
$S_3 = \langle Constant, 1, \{\phi\}, \{4`1, 1`4\}, 2\rangle$
$S_4 = \langle Variable, x, \{x\}, \{4`1, 1`4\}, 2\rangle$
$S_5 = \langle Guard, y = a, \{y\}\rangle$

In order to improve the efficiency of computation, we define an optimized pattern set. Let t be a transition, $PS(t) = AS(t) \cup GS(t)$ is the pattern set. An optimized pattern set $OPS = \{S_i | 1 \leq i \leq N\}$ for transition t is a set satisfying the following conditions:

1. $V(OPS(t)) = V(PS(t))$, where $V(PS(t))$ represents the set of all the variables in $PS(t)$,
2. $OPS(t) \subseteq PS(t)$,
3. $\forall S_i \in GS(t)$, $S_i \in OPS(t)$.

The first item ensures that the optimized pattern should cover all the variables that are covered by $PS(t)$. Please note that there may be some variables of transition t that are not covered by $PS(t)$, and these variables will be bound by their color sets. The second item ensures that all the elements in the $OPS(t)$

come from $PS(t)$. The third item states that all the guard patterns must be included in the $OPS(t)$. In the preprocessing section below, we will give the steps to obtain an optimized pattern set, $OPS(t)$ for a transition t from its pattern set, $PS(t)$, where we will see more conditions that an optimized pattern set should satisfy.

Besides, we collect other expressions that are not in the optimized pattern set to a set $NS(t)$, whose elements are non-multiset expressions. If an expression is a multiset expression, then we divide it first into a set of non-multiset expressions. Each expression in $NS(t)$ is denoted by a tuple $S = \langle E, X, M \rangle$, where

- E, the expression,
- X, the set of the variables in the expression,
- M, the initial marking of the place that connects the arc the expression belongs to.

For these expressions in $NS(t)$, we do not leave them until finishing all bindings and then test them. We will use the partial binding - partial test principle to test an expression that is not a pattern once we find that all the variables of it have been bound during the partial binding process. This could prone invalid bindings as early as possible.

For example, in Figure 1, if the variable x in $P1$ is bound by the value $1, 3, 4, 5$, we can immediately evaluate and test the expression $x + 2$. As a result, at this moment we can exclude the partial bindings $x = 3$,$x = 4$, and $x = 5$, as the place $P1$ has no enough tokens for these bindings.

3 Binding process

In this section, we recall the binding process and some definitions, which are adapted from [KC04].

In order to evaluate the arc expressions and the guard of a transition t, the variables relating to the transition (denoted by $V(t)$) must be bound by values (tokens). A binding of a transition is written in the form: $\langle v_1 = c_1, ...v_n = c_n \rangle$, where $v_i \in V(t)$, c_i is the color value belonging to a corresponding color set, $i = 1, 2, ..., n$.

Matching a token of an input place to a pattern would usually only bind to a subset of the variables of the transition t. For example, consider the transition t in Fig. 2, matching the token (1,a) to the pattern (x, y) will bind the variable x to 1, and y to a, but it will not bind any value to the variable z. So the concept of the partial binding is present, which means that a partial binding of a transition is a binding in which not all variables of the transition are bound by values. In the following, we use the $PartialBinding(p, c)$ to denote the partial binding by matching a pattern p with a token value c. If they are not matched, $PartialBinding(p, c) = \perp$.

In order to get a complete binding, we have to gradually merge the partial bindings. For example, matching the pattern x and the tokens of $P1$ yields the

following four partial bindings:

$\langle x = 1, y = \perp, z = \perp \rangle$
$\langle x = 3, y = \perp, z = \perp \rangle$
$\langle x = 4, y = \perp, z = \perp \rangle$
$\langle x = 5, y = \perp, z = \perp \rangle$

Matching the pattern (x, y) and the tokens of P2 yields the following three partial bindings:

$\langle x = 1, y = a, z = \perp \rangle$
$\langle x = 2, y = b, z = \perp \rangle$
$\langle x = 3, y = a, z = \perp \rangle$

If we merge them, we obtain the following two partial bindings:

$\langle x = 1, y = a, z = \perp \rangle$
$\langle x = 3, y = a, z = \perp \rangle$

The merging of two partial bindings relates to the concept of the compatible bindings. Two binding b_1 and b_2 are compatible (written as $Compatible(b_1, b_2)$), if and only if

$$\forall v \in V(t) : b_1(v) \neq \perp \wedge b_2(v) \neq \perp \Rightarrow b_1(v) = b_2(v)$$

For two compatible partial bindings b_1 and b_2, the combined partial binding (written as $Combine(b_1, b_2)$) satisfies:

$$b(v) = \begin{cases} b_1(v) \text{ if } b_1(v) \neq \perp \\ b_2(v) \text{ if } b_2(v) \neq \perp \\ \perp \quad \text{otherwise} \end{cases}$$

Based on those defintions above, the merging of two partial binding sets B_1 and B_2 (written as $Merge(B_1, B_2)$) is defined as:

$$Merge(B_1, B_2) = \{Combine(b_1, b_2) | \exists (b_1, b_2) \in B_1 \times B_2 : Compatible(b_1, b_2)\}$$

4 An algorithm for computing enabled transition instances

In this secition,we first give a top-level algorithm for computing enabled transition instances, which is illustrated in Algorithm 1. The algorithm inputs the pattern set $PS(t)$, and the non-pattern expression set of the transition, $NS(t)$, and outputs a complete binding set C.

The algorithm works as follows. First the algorithm performs a preprocessing (line 1) on $PS(t)$, and obtains an optimized pattern set, $OPS(t)$ by considering some optimization techniques. Afterwards, the algorithm executes the

BindbyPatterns process (line 2) to bind tokens residing on the places to the patterns. After that, the algorithm executes the BindbyColorSets process (line 3) to bind color sets to the variables that are not contained in the pattern set, $V(NS(t))\backslash V(OPS(t))$. During this process, the algorithm checks whether the guard is satisfied and whether the input places have sufficient tokens. So, finally we get all valid complete bindings. In the next sections, we will continue to discuss the three processes of the algorithm in more details.

Algorithm 1: An algorithm for computing enabled transition instances.

Input: $PS(t), NS(t)$
Output: C
1 $OPS(t)$ = Preprocess($PS(t)$);
2 C = BindbyPatterns($OPS(t), NS(t)$);
3 C = BindbyColorSets($C, NS(t), V(NS(t)) \setminus V(OPS(t))$);

4.1 Preprocessing of the pattern set

The preprocessing of the pattern set is very important as it may prune a lot of invalid partial bindings in advance and find if the transition can be enabled as early as possible, thus improving the efficiency of computation of enabled transition instances. In this section, we give the steps to preprocess the pattern set and as a result obtain an optimized ordered pattern set.

(1)Testing multiplicity.

We begin the preprocessing of the pattern set with multiplicity testing. During this step, we can discard the tokens in the current marking that do not contribute to the valid bindings. This is performed by checking whether the number of tokens with the same color is greater than or equal to the muliplicity of the pattern. For a constant pattern, if this is evaluated to false, we immediately stop the preprocessing process, and directly disable this transition. If true, we can now remove the constant patterns from the binding pattern set, as we will not use it any longer for succedent processes. For a variable or tuple pattern, if this is evaluated to false, we will remove these tokens from the current marking set. If the current marking set becomes empty, we stop the preprocessing process, and directly disable this transition.

The algorithm is illustrated in Algorithm 2, which works as follows. The algorithm executes a loop for each pattern in the pattern set $PS(t)$. If a pattern is a constant pattern, the multiplicity of the constant pattern is checked with the tokens of the constant color. Here $S_i.M_i\langle c\rangle$ represents the number of the tokens with the color "c". If this is evaluated to false, the transition is determined not to be enabled (lines 2-6). If a pattern is a variable or tuple pattern, for each color in the initial marking, the multiplicity is tested. The tokens will be removed if the testing is false. If the current marking set of the pattern becomes empty, the transition is determined not to be enabled (lines 8-17).

After the multiplicity testing for the example in Figure 1, we get the following pattern set, where pattern S_3 is removed.

$S_1 = \langle Variable, x, \{x\}, \{2`1, 2`3, 1`4, 1`5\}, 1\rangle$
$S_2 = \langle Tuple, (x, y), \{x, y\}, \{1`(1, a), 1`(2, b), 1`(3, a)\}, 1\rangle$
$S_4 = \langle Variable, x, \{x\}, \{4`1\}, 2\rangle$
$S_5 = \langle Guard, y = a, \{y\}\rangle$

Algorithm 2: Multiplicity testing.

```
   Input: PS(t)
   Output: OPS(t)
 1 for each pattern S_i ∈ PS(t) do
 2     if S_i is a constant pattern then
 3         c ← S_i.E_i;
 4         if S_i.M_i⟨c⟩ < S_i.m_i then
 5             transition t is not enabled;
 6         endif
 7     endif
 8     if S_i is an variable or tuple pattern then
 9         for each color c ∈ S_i.M_i do
10             if S_i.M_i⟨c⟩ < S_i.m_i then
11                 S_i.M_i ← S_i.M_i\{S_i.M_i⟨c⟩};
12             endif
13         endfor
14         if S_i.M_i is empty then
15             transition t is not enabled;
16         endif
17     endif
18 endfor
```

(2)Merging identical patterns.

Usually, there exist several identical patterns (identical expressions) for a transition. Merging them can remove invalid partial bindings as much as possible before the binding begins. The algorithm is illustrated in Algorithm 3. To merge two identical patterns, for example, S_i and S_j, $i \neq j$, we need to get their colors that have tokens, denoted by C_i and C_j (lines 1-2), respectively. We calculate the merged colors by $C_k = C_i \cap C_j$ (line 3). If C_k is not empty, we build a new pattern S_k, where M_k stores the merged color with the multiplicity being 1 and m_k is set to 1 (lines 4-10). At the same time, we remove the old patterns S_i and S_j and add a new pattern S_k to the pattern set. If the set C_k is empty, we can directly set the transition disabled.

For example, Figure 1 has two identical patterns: S_1 and S_4. The colors with current tokens are $\{1, 3, 4, 5\}$ and $\{1\}$, respectively, and the merged color is $\{1\}$.

Algorithm 3: Merging identical patterns.

Input: S_i, S_j
Output: S_k

```
1  C_i ← S_i.M_i;
2  C_j ← S_j.M_j;
3  C_k ← C_i ∩ C_j;
4  if C_k is not empty then
5      S_k.P_k ← S_i.P_i;
6      S_k.E_k ← S_i.E_i;
7      S_k.X_k ← S_i.X_i;
8      S_k.M_k ← C_k;
9      S_k.m_k ← 1;
10 endif
11 if C_k is empty then
12     transition t is not enabled ;
13 endif
```

So we remove the patterns, S_1 and S_4 and add a new pattern, S_{14}. Now the patterns for Figure 1 become:

$S_2 = \langle Tuple, (x, y), \{x, y\}, \{1`(1, a), 1`(2, b), 1`(3, a)\}, 1\rangle$
$S_{14} = \langle Variable, x, \{x\}, \{1`1\}, 1\rangle$
$S_5 = \langle Guard, y = a, \{y\}\rangle$

(3)Sorting patterns in terms of the less different tokens first policy [Cae96].

After that, we can sort the patterns in terms of the less different tokens first policy that will be discussed in detail later. For example, after sorting, the binding patterns in Figure 1 become:

$S_5 = \langle Guard, y = a, \{y\}\rangle$
$S_{14} = \langle Variable, x, \{x\}, \{1`1\}, 1\rangle$
$S_2 = \langle Tuple, (x, y), \{x, y\}, \{1`(1, a), 1`(2, b), 1`(3, a)\}, 1\rangle$

After finishing the preprocessing, we finally get an optimized pattern set $OPS(t)$, which will be used as the input of the following algorithm.

4.2 Binding by matching tokens and patterns

In this section, we describe a key component of the algorithm for the computation of enabled transition instances (illustrated in Algorithm 4), binding by matching tokens residing on the places and patterns in the set, $OPS(t)$, which is based on the algorithm in [KC04].

The algorithm executes a loop to handle each member of the pattern set $OPS(t)$. Lines 4-9 consider the case of the guard patterns, in which the right

hand side of the guard is matched against the left hand side of it. Lines 10-18 consider the case of matching the tokens in the current marking and the arc expression patterns. Lines 19-31 test if each partial binding is valid using the non-pattern set $NS(t)$. For an expression of $NS(t)$ whose variables are fully bound, if it is a guard and is evaluated to be false for a partial binding, then the partial binding is invalid. If the expression is an arc expression and can not get enough tokens by evaluating it with a partial binding, then the partial binding is also invalid.

Compared to the algorithm in [KC04], our algorithm has the following distinguished features:

- The biggest difference is that our algorithm employs the partial binding - partial test principle, that is, during a partial binding process, if the variables in a non-pattern expression have been detected to be fully bound, then we evaluate and test it immediately. As a result, this would not produce any invalid complete binding when the binding process ends.
- The overall algorithm considers the case of the variable binding to the color set, as this case may be encountered in our colored Petri nets, which will be discussed in the next section.

We still use the example in Figure 1 to demonstrate how the algorithm above works. For the first loop, the guard pattern $y = a$ is processed, and let 'a' bind to y. Then the pattern S_{14} is processed, and let x be bound by '1'. After that, the non-pattern expression $x < 3$ begins to work as it finds that the variable x has been bound and keeps the binding '1' to x. We continue to bind $(1, a)$ to the pattern (x, y) and merge them with existing bindings. After these steps, we get the following partial bindings.

$\langle x = 1, y = a, z = \perp \rangle$

4.3 Binding colors of color sets to variables

When the variables are not contained by all the patterns, they have to be bound to colors of their color sets. The algorithm is illustrated in Algorithm 5. The algorithm works as follows. The algorithm executes a loop for each variable $v \in V(NS(t)) \backslash V(OPS(t))$, which stores all the variables that have to be bound by the color sets. Lines 3-9 bind the colors to the variables. Here $c(v)$ represents the color set of variable v. Lines 10-22 test if the expressions in $NS(t)$ satisfy the guard or have sufficient tokens in the corresponding places.

We continue to apply this algorithm to the example in Figure 1. Here, we bind the color set C with colors, $c1$ and $c2$, to the variable z. Then we get the following complete bindings.

$\langle x = 1, y = a, z = c1 \rangle$
$\langle x = 1, y = a, z = c2 \rangle$

Algorithm 4: An algorithm for matching tokens and patterns.

Input: $OPS(t) = \{S_i | 1 \leq i \leq N\}$, $V(t)$
Output: C

$C \leftarrow \phi$;
for *each pattern* $S_i \in OPS(t)$ **do**
  $C^{'} \leftarrow \phi$;
  `// binding`
  if $S_i \equiv g_{il} = g_{ir}$**then**
    $b^{'} \leftarrow PartialBinding(g_{il}, g_{ir})$;
    if $b^{'} \neq \perp$ **then**
      $C \leftarrow C \cup \{b^{'}\}$;
    endif
  endif
  if $S_i \in AS(t)$ **then**
    for *each colored token* $c \in S_i.M_i$ **do**
      $b^{'} \leftarrow PartialBinding(S_i.E_i, c)$;
      if $b^{'} \neq \perp$ **then**
        $C^{'} \leftarrow C^{'} \cup \{b^{'}\}$;
      endif
    endfor
    $C \leftarrow Merge(C, C^{'})$;
  endif
  `// testing`
  for *each expression* $S_k \in NS(t)$ **do**
    if $V(S_k) \subseteq V(C)$ **then**
      for *each binding* $b \in C$ **do**
        if $S_k.E_k$ *is a guard expression and* $S_k.E_k\langle b\rangle$ *is false* **then**
          $C = C \backslash \{b\}$;
        endif
        if $S_k.E_k$ *is an arc expression and* $S_k.E_k\langle b\rangle > S_k.M_k\langle c\rangle$ **then**
          $C = C \backslash \{b\}$;
        endif
      endfor
      $NS(t) \leftarrow NS(t) \backslash \{S_k\}$;
    endif
  endfor
endfor

Algorithm 5: An algorithm for binding colors of color sets to variables.

Input: C, $TS(t)$, $NS(t)$, $V(NS(t))\backslash V(OPS(t))$
Output: C
`// binding`
for *each variable* $v \in V(NS(t))\backslash V(OPS(t))$ **do**
  $C^{'} \leftarrow \phi$;
  for *each color* $c \in c(v)$ **do**
    $b^{'} \leftarrow PartialBinding(v, c)$;
    if $b^{'} \neq \perp$ **then**
      $C^{'} \leftarrow C^{'} \cup \{b^{'}\}$;
    endif
  endfor
  $C \leftarrow Merge(C, C^{'})$;
  `// testing`
  for *each expression* $S_k \in NS(t)$ **do**
    if $V(S_k) \subseteq V(C)$ **then**
      for *each binding* $b \in C$ **do**
        if $S_k.E_k$ *is a guard and* $S_k.E_k\langle b\rangle$ *is false* **then**
          $C = C\backslash\{b\}$;
        endif
        if $S_k.E_k$ *is an arc expression and* $S_k.E_k\langle b\rangle > S_k.M_k\langle c\rangle$ **then**
          $C = C\backslash\{b\}$;
        endif
      endfor
      $NS(t) \leftarrow NS(t)\backslash\{S_k\}$;
    endif
  endfor
endfor

5 Optimization Techniques

In this section, we briefly summarize some of the optimization techniques that we use to improve the efficiency of the computation of enabled transition instances.

(1)Partial binding - partial test.

As described above, we collect all the arc or guard expressions that are not covered by the pattern set. We do not leave them until finishing all complete bindings and then test them. Rather, we will test them once we find that all the variables of them have been bound during the partial binding process. For example, in Figure 2, the optimized pattern set is x, y and z. If we do not use this policy, we would first get $20 \times 30 \times 40$ complete bindings, then test these bindings by evaluating $x+1$ and $y+1$ and then get 480 valid bindings. However, if we use this policy, x is first bound and 20 partial bindings are gotten. After that the expression $x+1$ is tested, and the valid bindings for x are now 3. Then y is bound, and the partial bindings for x and y become 90. When the expression $y+1$ is tested, the partial bindings become 12. Finally, the variable z is bound, and the final complete bindings are gotten, whose number is 480. Obviously, using this policy usually reduces the amount of computation greatly.

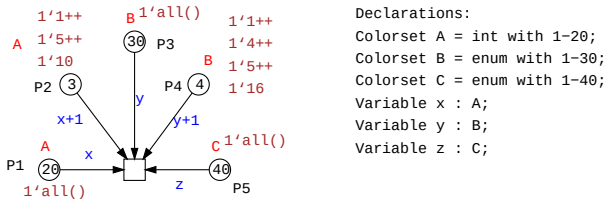


Figure 2. An example to demonstrate the policy of partial binding - partial test.

(2) Less different tokens first policy [Cae96].

As can be easily noticed and analyzed, the information of the tokens residing on different places can affect the efficiency of computation of enabled transition instances. For example, in Figure 1, for transition t, if we bind first x to the tokens of $P1$, we have 4 bindings, but if we bind the tokens in $P2$ to x first, we get 2 bindings. That is to say, the binding order of variables is quite different in efficiency; therfore, we can optimize the binding order of the patterns. We use the less different tokens first policy, which has been given by [Cae96].

(3) Multiplicity test.

When finding patterns, we also record the multiplicities of the patterns. We use them to test if the places contain enough tokens for enabling before the binding begins, which already is reflected in the Algorithm 2.

(4) Merging identical patterns.

Merging the same patterns before binding is more efficient than binding a pattern and then testing another pattern during binding. In the algorithm presented

above, we consider the merging of identical patterns during the preprocessing phase. This heuristics is very useful when there are many identical patterns for a transition and the tokens for each pattern are notably different.

All the heuristics have been used in our algorithm, which can be seen in different parts in Algorithm 1-5.

6 Related work

In this section, we describe and compare some related work concerning the computation of enabled transition instances.

Mäkelä [Mak01] used a unification technique to calculate enabled transition instances for the algebraic system nets that are in fact another kind of high-level Petri nets, which gave a different idea on finding enabled bindings.

Sanders [San00] considered the calculation of enabled binding as a constraint satisfaction problem. He imposed strong constraints on the form of arc expressions, only considering the form $n`exp$, which is impossible for nearly all the colored Petri nets.

Gaeta [Cae96] studied the enabled test problem of Well-Formed Nets, and gave some heuristics for determining the binding elements, i.e., less different tokens first policy, which are very useful for the efficiency of calculation of enabled transition instances.

Mortensen [Mor01] described data structure and algorithms used in the CPN tools. He used the locality principle to dicover enabled transitions rather than calculating all the transition each time. He also discussed how to optimize the binding sequences.

Kristensen et al. [KC04] gave a pattern reference algorithm for enabled binding calculation of CPN tools. We also adopt that idea to design our binding algorithm, but compared their algorithm, our algorithm considers more optimization techniques.

In our work, we take into account the main idea of [KC04] and also some ideas of [Mak01] and [Cae96], but comapared with all the previous work, we adopt a new principle, partial binding - partial test, and consider more optimization techniques to improve the efficiency of computing enabled transition instances for colored Petri nets.

7 Conclusions

In this paper, we present an algorithm for computation of enabled transition instances for colored Petri nets. This algorithm uses the principle of partial binding - partial test and adopts some optimization techniques for preprocessing. The principle of partial binding - partial test allows us to test the expressions during the partial binding process so as to prone invalid bindings as early as possible. The use of optimization techniques prunes invalid potential bindings before the binding begins, and also finds the disable transitions at an early

phase. Among them, the less different tokens first policy allows variables to have less bindings, the multiplicity test excludes insufficient tokens before the binding begins and the merging of identical patterns avoids repeated bindings for identical patterns. All these techniques contribute to the improvements of efficiency.

This algorithm can realize an efficient computation of enabled transition instances for large-scale colored Petri nets. At the same time, we are adapting this algorithm to unfold colored Petri nets, which will improve the efficiency of unfolding of colored Petri nets and thus simulation of colored stochastic Petri nets. In the future, we will investigate more optimization techniques to further improve the efficiency.

Acknowledgments

This work has been supported by the Modeling of Pain Switch (MOPS) program of Federal Ministry of Education and Research (Funding Number: 0315449H).

References

[Cae96] R. Gaeta: Efficient Discrete-Event Simulation of Colored Petri Nets. IEEE Transactions on Software Engineering. 22(9), 629-639 (1996)

[JKW07] K. Jensen, L. M. Kristensen, L. Wells: Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. International Journal on Software Tools for Technology Transfer. 9, 213-254 (2007)

[KC04] L. M. Kristensen, S. Christensen: Implementing Coloured Petri Nets Using a Functional Programming Language. Higher-Order and Symbolic Computation. 17(3), 207-243 (2004)

[LH10] F. Liu, M. Heiner: Colored Petri nets to model and simulate biological systems. International Workshop on Biological Processes and Petri Nets (BioPPN). (2010)

[Mak01] M. Mäkelä: Optimising Enabling Tests and Unfoldings of Algebraic System Nets. International Conference on Application and Theory of Petri Nets, LNCS, 2075, 283-302 (2001)

[Mor01] K. H. Mortensen: Efficient Data-Structures and Algorithms for a Coloured Petri Nets Simulator. 3rd Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (CPN'01), 57-74 (2001)

[Pet10] Petri Nets World. http://www.informatik.uni-hamburg.de/TGI/PetriNets. (2010)

[San00] M. J. Sanders: Effecient Computation of Enabled Transition Bindings in High-Level Petri Nets. IEEE International Conference on Systems, Man and Cybernetics. 3153-3158 (2000)

[RMH10] C. Rohr, W. Marwan, M. Heiner: Snoopy - a unifying Petri net framework to investigate biomolecular networks. Bioinformatics. 26(7): 974-975 (2010)

[Ull98] J. D. Ullman: Elements of ML Programming. Prentice-Hall. (1998)

A Hybrid Petri Net for Modelling Hybrid Biochemical Interactions

Mostafa Herajy and Monika Heiner

Data Structures and Software Dependability, Computer Science Department,
Brandenburg University of Technology, Cottbus, Germany
http://www-dssz.informatik.tu-cottbus.de/

Abstract. Recently hybrid modelling and simulation of biochemical systems have attracted increasing interest. This is motivated by the need of simulating systems which integrate different sub-cellular models, and the fact that bio networks themselves are inherently stochastic, however stochastic simulation is time expensive. Compared to other methods of biological modelling, Petri nets are characterized by their intuitive visual representation and executability of biological models. In this paper, we present a hybrid Petri net class that incorporates both continuous and stochastic capabilities. The presented class is intended to model and simulate hybrid biological systems such that they contain some parts which are simulated deterministically while other parts are simulated stochastically.

Keywords: Hybrid Petri Net, Hybrid Biochemical Simulation, Systems Biology.

1 Introduction

Computer simulation is an essential tool for studying biochemical systems. The deterministic approach (continuous simulation) is the traditional way of simulating biochemical pathways. In this approach, reactions and their influence on the concentrations of the involved species are represented by a set of ordinary differential equations (ODEs). The changes in reactants and products are obtained through solving the resulting ODEs using numerical integration algorithms. While this approach has the advantage of a well established mathematical basis and strong documentation, it lacks to capture the phenomena which occur due to the underlying discreteness and random fluctuation in molecular numbers [Pah09],[LCP+08], especially in situations where the number of molecules is few.

Stochastic simulation [Gil76] provides a very natural way of simulating biochemical pathways, since it can successfully capture the fluctuations of the underlaying model. Furthermore it deals correctly with the problem of extremely low number of molecules [ACT05]. In stochastic simulation, species are no longer represented as continuous concentrations which change continuously with time, instead they are represented as discrete entities such that their dynamics can be

simulated using the machinery of Markov process theory. In [SYS+02] an example is given comparing deterministic versus stochastic modeling using a simple model of the intracellular kinetics of a generic virus.

A major drawback of the stochastic simulation is that it is computationally expensive, when it comes to simulate larger biological models [Pah09],[LCP+08], [ACT05], especially when there are a large number of molecules of some chemical species. The reason behind this problem comes from the fact that we have to simulate every reaction event when we use stochastic simulation to simulate biological systems [LCP+08]. This drawback motivates scientists to search for other methods to enhance the capability of the stochastic approach. Hybrid simulation is one of these methods.

Hybrid simulation [ACT05],[Kie+04],[Rue+07] of biochemical system using both deterministic and stochastic approaches has been recently introduced to take the advantage of capturing the randomness and fluctuation of the discrete stochastic model and allows at the same time a reasonable computation time. This goal is achieved by simulating fast reactions deterministically, while simulating slow reaction stochastically. While this method provides a promising approach for simulating biochemical models, there are some open questions which need to be solved [Pah09].

Petri nets provide a very useful way of modelling biochemical pathways [RML93],[BGH+08],[HGD08],[Mat+03] since they provide an intuitive approach of transforming the biological model into a graphical representation which coincides with the qualitative description of this model. Furthermore, they can be easily transformed later for quantitative simulation.

Continuous Petri nets are used in biological modelling to introduce an easy way of modelling complex biological pathways and simultaneously hide the mathematical complexities of the underlying ODE. Contrary, in stochastic Petri nets and their simulation, transitions fire with exponentially distributed random waiting time.

Hybrid Petri nets [AD98] incorporate both continuous and discrete capabilities and can be used to model systems which contain both discrete and continuous elements. Many various of hybrid Petri nets have been introduced during the last two decades, with different modeling goals. Some examples can be found in [Mat+03],[TK93] and [PB09]. An overview of continuous, discrete and hybrid Petri nets can be found in [DA10] .

In this paper, we introduce the definition of a hybrid continuous-stochastic Petri net, HPN, and integrate it into Snoopy [HRR+08],[RMH10], a tool to design and animate or simulate hierarchical graphs, among them the qualitative, stochastic and continuous Petri nets, which incorporate the modeling capabilities of the previously introduced stochastic and continuous classes [GH06], [GHL07],[HLG+09], The new net class HPN, is intended to model biological pathways that require hybrid simulation, such that the resulting Petri net can be simulated deterministically and stochastically based on the model specification.

This paper is organized as follow: Firstly we briefly review the motivations of using continuous and stochastic Petri nets to model biochemical reactions. Then we introduce our hybrid stochastic-continuous Petri net class, by firstly presenting a formal definition as well as the connectivity rules between its elements. The illustration of the modeling capabilities of HPN to model biological systems is then demonstrated using two examples. At the end we conclude by a summary and autlook of future work.

2 Petri Net and Biological Systems

The tight analogy between Petri net and biochemical reactions makes it a natural choice to model these reactions [RML93],[HGD08]. Being bipartite, concurrency, and stochasticity are common properties shared by Petri nets and biochemical interactions. Qualitative Petri net [HGD08] can be used to analyze the biochemical systems qualitatively, while stochastic and continuous Petri nets are used to simulate them quantitatively. Before we discuss the various aspects of the hybrid stochastic-continuous Petri net, we provide a short overview of continuous and stochastic Petri nets as well as how they can be used to model biological systems. Detailed discussion can be found in [BGH+08] and [HGD08], and for a general introduction to Petri net see [DA10] and [Mur89].

Continuous Petri nets provide a way for modeling systems in which states change continuously with time. In this class of Petri nets, places contain nonnegative real values and transitions fire continuously with time. In systems biology, continuous Petri nets provide a very useful way of representing ODEs. Preplaces of the transitions represent reactants species and the marking of these places represents species' concentrations. Each transition is associated with a rate function which defines the kinetic rate. The corresponding ODE which represents the reaction which is modeled by this transition can be generated using (1) [GH06].

$$\frac{dp}{dt} = \sum_{t \in {}^\bullet p} f(t,p)v(t) - \sum_{t \in p^\bullet} f(p,t)v(t) \tag{1}$$

where $v(t)$: is the rate function and $f(t,p)$: is the weight connecting transition t with place p and ${}^\bullet p, p^\bullet$ are the pre- and post-transitions of place p, respectively. Note that place names are read as real variables.

The resulting system of ordinary differential equations of all places describes the changes with respects to time in all biochemical species. Our HPN supports the same functionality as the aforementioned continuous Petri net.

In contrast to continuous Petri nets, stochastic Petri nets preserve the discrete state description. The biochemical models are simulated stochastically by associating a probability-distributed firing rate (waiting time) with each transition. This means that there is a time which has to elapse before an enabled transition t $\in T$ fires [HLG+09], where T is the set of all stochastic transitions. The probability density function of the exponentially distributed random variable, x_t, which represents the waiting time, is given by (2)

$$f_{x_t}(\tau) = \lambda_t(m).e^{-\lambda_t(m)\tau}, t \geq 0 \quad (2)$$

where $\lambda_t(m)$ is a marking dependent kinetic rate which is associated with each stochastic transition. $\lambda_t(m)$ is equivalent to the propensity of the reaction t, $a(x_i)$, of the stochastic simulation algorithms which are presented in [Gil76].

Because of the deterministic nature of continuous Petri nets, the concentration of particular species will have the same values at each time point for repeated experiments, which is the main difference between simulation of stochastic and continuous biological models, and hence for Petri nets as well. In a typical execution of stochastic Petri nets, each transition gets its own local timer. When a particular transition becomes enabled, the local timer is set to an initial value which is computed by means of the corresponding probability distribution. The local timer is then decremented at a constant speed and the transition will fire when the time reaches zero. A race will take place in the case of conflict between more than one enabled transition.

To extend the modeling capabilities of stochastic Petri nets (SPN) in biological system, two extensions, general stochastic petri nets ($GSPN_{bio}$) and deterministic stochastic petri nets ($DSPN_{bio}$), of SPN are introduced in [HLG+09]. These extensions add inhibitor and read arcs and deterministically time-delayed transitions to stochastic Petri nets.

In the following section, we present the merging of stochastic Petri nets (using the extended version) and continuous one, to produce a hybrid continuous-stochastic Petri nets which are capable of modeling and simulating hybrid biochemical reactions.

3 Hybrid Continuous-Stochastic Petri Nets

In this section we describe the hybrid continuous stochastic Petri nets capable of modeling systems which consist of discrete and continuous parts. The discrete parts may be considered as a set of reactions which involves species with low number of molecules such that it is adequate to simulate them in a discrete way. On the other hand, continuous elements of this class can represent a set of reactions which involves species with large number of molecules, which are computationally too expensive to be simulated stochastically. Continuous and stochastic Petri nets complement each other. We get modelling power of fluctuation and discreteness, when using the stochastic simulation and at the same time we can simulate the computationally expensive parts deterministically using ODEs solvers.

Generally speaking, biochemical systems can involve reactions from more than one type of biological networks, for example regulatory, metabolic or transduction pathways. Incorporation of reactions which belong to distinct (biological) networks, tend to result in stiff systems. This follows from the fact that regulatory network's species may contain a few number of molecules, while metabolic networks' species may contain a large number of molecules [Kie+04]. In our hybrid Petri nets, reactions which involves species with a small number of molecules

are represented by discrete entities, so that they can be simulated stochastically, while reactions which include a large number of molecules are represented by continuous entities, so that they can be simulated deterministically. The connection between the discrete and continuous parts takes place using either special arcs (read, inhibitor, or equal arcs) or in some cases using the standard arcs based on the defined connection rules.

In the rest of this section, we will discuss in more detail the newly introduced hybrid continuous-stochastic Petri nets in terms of the graphical representation of its elements as well as the firing rules and connectivity between the continuous and stochastic parts.

3.1 Graphical Representation

As expected, HCSPN contains two types of places: discrete and continuous. Discrete places (single line circle) contain integer numbers which represent for example the number of molecules in a given species. On the other hand, continuous places - which are represented by shaded line circle - contain real numbers which represent the concentration of a given species. This means that we can combine the power of the previously discussed continuous and stochastic Petri nets together in one class. HCSPN contains a variety of transition types: continuous, stochastic, deterministic, immediate, and scheduled transitions [HLG+09]. Continuous transitions - shaded line square - fire continuously in the same way like in continuous Petri nets. Their semantics are governed by ordinary differential equations. Their ODEs define the changes in the transitions' pre- and post-places.

Stochastic transitions which are drawn in Snoopy as a square, fire randomly with an exponential random distribution delay. The user can specify a set of firing rate functions, which determine the random firing delay. Deterministic (time delay) transitions - black square - fire after a specified time delay, immediate transitions - black bar - fire with zero delay, and they have higher priority in the case of a conflicts with other transitions. They may carry weights which specify the relative firing frequency in the case of conflicts between more than one immediate transition. Scheduled transitions - grey square - fire at a user-specified time point or time interval.

The connection between those two types of nodes (places and transitions), takes place using a set of different arcs. HCSPN contains five types of edges: standard, inhibitor, read, equal and reset arcs. Standard edges connect transitions with places or vice versa . They can be continuous, i.e carry real value weights (or in the biochemical context stoichiometry), or discrete i.e carry non-negative integer value weights. Special arcs like inhibitor, read, equal and reset arcs provide only connection from places to transitions, but not vice versa. The connection rules and their underlying semantics are given below. Fig. 1 provides a graphical illustration of those elements. While this graphical notation is the default one, they can be easily customized using our Petri nets editing tool, Snoopy.

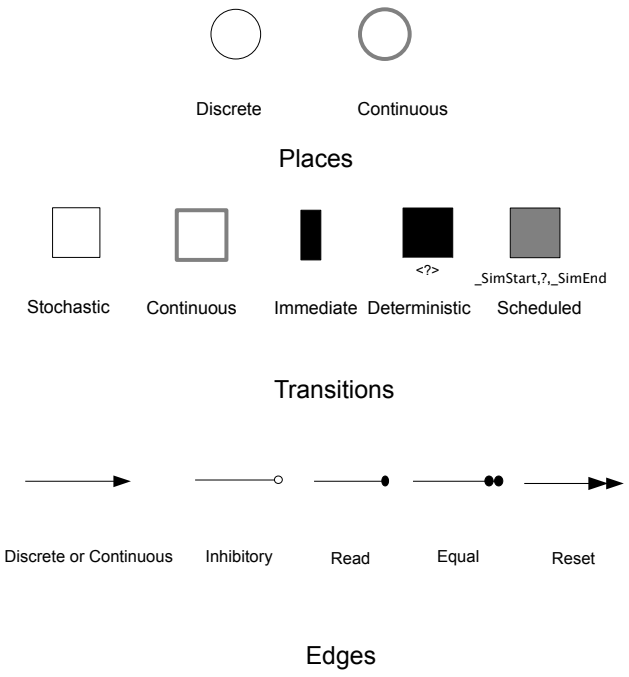


Fig. 1. Graphical representation of the HCSPN's elements

3.2 Formal Definition

HCSPN is a 5-Tuple, $HCSPN = \{P, T, A, V, m_0\}$ where: P, T are finite, nonempty and disjoint sets. P is the set of places and T is the set of transitions with:

- $P = \{P_{cont} \cup P_{disc}\}$ whereby P_{cont} is the set of continuous places to which nonnegative real values can be assigned and P_{disc} is the set of discrete places to which nonnegative integer values can be assigned.
- $T = T_{cont} \cup T_{stoch} \cup T_{im} \cup T_{timed} \cup T_{scheduled}$ with:
  1. T_{cont}, the set of continuous transitions, which fire continuously over time.
  2. T_{stoch}, the set of stochastic transitions, which fire stochastically with exponentially distributed waiting time.
  3. T_{timed}, the set of deterministic transitions, which fire with a deterministic time delay.
  4. $T_{scheduled}$, the set of scheduled transitions, which fire at predefined firing time points.
  5. T_{im}, the set of immediate transitions, which fire with waiting time zero and it has higher priority compared to other transitions.
- $A = \{A_{cont} \cup A_{disc} \cup A_{inhibit} \cup A_{read} \cup A_{equal}, \cup A_{reset}\}$, is the set of directed edges, whereby:

1. $A_{cont} : ((P_{cont} \times T) \cup (T \times P_{cont})) \rightarrow \mathbb{R}_0$: defines the set of continuous, directed arcs, weighted by nonnegative real values.
2. $A_{disc} : ((P \times T) \cup (T \times P)) \rightarrow \mathbb{N}_0$: defines the set of discrete, directed arcs, weighted by nonnegative integer values.
3. $A_{read} : (P \times T) \rightarrow \mathbb{R}^+ if P \in P_{cont}$ or $A_{read} : (P \times T) \rightarrow \mathbb{N}^+ if P \in P_{disc}$, defines the set of read arcs.
4. $A_{equal} : (P \times T) \rightarrow \mathbb{R}_0^+ if P \in P_{cont}$ or $A_{equal} : (P \times T) \rightarrow \mathbb{N}_0^+ if P \in P_{disc}$, defines the set of equal arcs.
5. $A_{inhibit} : (P \times T) \rightarrow \mathbb{R}^+ \cup \{0^+\} if P \in P_{cont}$ or $A_{inhibit} : (P \times T) \rightarrow \mathbb{N}^+ if P \in P_{disc}$, defines the set of inhibits arcs, where 0^+ means very small positive real number but not zero.
6. $A_{reset} : (P \times T_{discrete})$ defines the set of reset arcs, where $T_{discrete} = T_{stoch} \cup T_{im} \cup T_{timed} \cup T_{scheduled}$ is the set of discrete transitions.

– V is a set of functions {f,g,d,w} where :
1. $f : T_{cont} \rightarrow H_c$ is a function which assigns a rate function h_c to each continuous transition t $\in T_{cont}$, such that : $\{h_{c_t} | h_{c_t} : \mathbb{R}_0^{|^\bullet t|} \rightarrow \mathbb{R}^+, t \in T_{cont}\}$ is the set of all rates functions and $f(t) = h_{c_t}, \forall t \in T_{cont}$.
2. $g : T_{stoch} \rightarrow H_s$ is a function which assigns a stochastic hazard function h_{s_t} to each transition t $\in T_{stoch}$, whereby $\{h_{s_t} | h_{s_t} : \mathbb{N}_0^{|^\bullet t|} \rightarrow \mathbb{R}^+, t \in T_{stoch}\}$ is the set of all stochastic hazard functions and $g(t) = h_{s_t} \forall t \in T_{stoch}$.
3. $d : T_{timed} \rightarrow \mathbb{R}^+$, is a function which assigns a constant time to each deterministic transitions representing the waiting time.
4. $w : T_{im} \rightarrow H_w$ is a function which assigns a weight function h_w to each immediate transition t $\in T_{im}$, such that : $\{h_{w_t} | h_{w_t} : \mathbb{N}_0^{|^\bullet t|} \rightarrow \mathbb{R}^+, t \in T_{im}\}$ is the set of all weight functions and $w(t) = h_{w_t}, \forall t \in T_{im}$

– $m_0 = \{m_{cont} \cup m_{disc}\}$: is the set of initial marking for both the continuous (P_{cont}) and discrete places (P_{disc}), whereby $m_{cont} \in \mathbb{R}_0^{+|P_{cont}|}$, $m_{disc} \in \mathbb{N}_0^{+|P_{disc}|}$.

A critical question arises when considering the mixing between discrete and continuous elements: how are these two different parts connected with each other? Fig. 2, provides a graphical illustration of how the connection between different elements of the introduced HCSPN takes place. Note that other discrete transitions (immediate, deterministic and scheduled transitions) follow the same connection rules as stochastic transitions.

Firstly, we will consider the connection between continuous transitions and the other elements of the HCSPN. Continuous transitions can be connected with continuous places in both directions using continuous arcs (i.e arc with real value weight). This means that continuous places can be pre- and post-places of continuous transitions. These connections represent deterministic, biological interaction. According to the previous formal definition, each continuous transition takes a rate function. This rate function represents the kinetics of the deterministic reaction. Like in continuous Petri net, the firing of this transition can be represented as an ODE. The continuous transition can be connected also with

a discrete or continuous places, but only by one of the special arcs (inhibitor, read, equal). Read arcs allow to specify positive side conditions, while inhibitor arcs allow to specify negative side conditions. It is worth being mention, that the markings of the transition preplaces connected by these special arcs do not change when the transition fires. This type of connection allows a connection between the discrete and continuous parts of the biochemical model.

Discrete places are not allowed to be connected with continuous transitions using standard arcs, because the firings of continuous transitions are governed by an ODE which requires real values in the pre- and post-places. Discrete transitions (stochastic, deterministic, immediate and scheduled) can be connected with discrete or continuous places in both directions using standard arcs. However, the arc's weight should be considered, i.e the connection between discrete transitions and discrete places takes place using arcs with nonnegative integer numbers, while the connection between continuous place and discrete transitions is weighted by nonnegative real numbers. The general rule to determine the weight type of the arcs is the type of the transition's pre/post places.

The connection between continuous places and discrete transitions will result in a model like discussed in [TK93], in which the changes in the continuous places are governed by firing of stochastic transitions. Discrete transitions can also have discrete or continuous places as the transition pre-places using the special arcs.

3.3 Simulation of HCSPN

Due to the use of both stochastic and continuous parts in HCSPN, we have now two different clocks: one for the continuous parts and the other for the stochastic ones. The ODEs solver which represents the semantics of the continuous Petri net evolves deterministically with approximate time steps, while the stochastic transitions fire stochastically with exact time steps. Because we intend to use HCSPN to simulate biochemical reactions, we provide a synchronization mechanism between the stochastic and continuous Petri nets, since some species (places) may belong simultaneously to both continuous and stochastic Petri nets due to the partition of the reactions. In this part of the paper we propose a Petri net interpreted synchronization algorithm based on the algorithm presented in [ACT+04].

Many synchronization algorithms are used in the literature to synchronize between the deterministic regime and the stochastic one in hybrid simulation of biochemical reactions; some of them can be found in [Pah09,ACT05,Kie+04,Rue+07]. We opted to use the algorithm in [ACT+04], since it has a rigid mathematical basis for the synchronization of the two different clocks.

The algorithm which is presented here is based on the direct method [Gil76], see [ACT05,ACT+04] for other variations based on the first and second reaction method. The algorithm is based on the function $f(\tau|t)$. $f(\tau|t)$ will decide when we can switch from the continuous world to the stochastic one. We firstly draw an exponentially distributed random variable ξ and initialize $f(\tau|t) = 0$, then we start to simulate the continuous transitions using the ODE solvers. During the

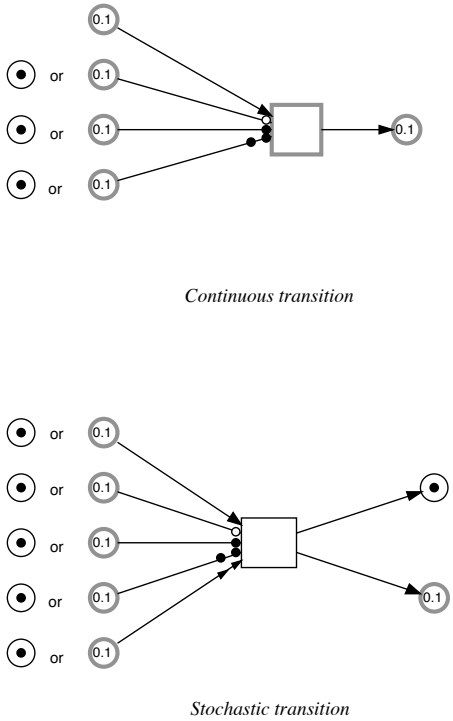


Fig. 2. Possible connections between HCSPN's elements
Continuous and stochastic transitions' connectivity with discrete and continuous places. Note that discrete places contain nonnegative integer values, while continuous places contain nonnegative real values.

continuous simulation, $f(\tau|t)$ will be increased according to the time evolution of the ODE presented in (3)

$$\frac{d}{dt}f(\tau|t) = \sum_{j \in T_{stoch}} g_j(m(\tau), \tau) \tag{3}$$

where $g_j(m(\tau), \tau)$ is the rate function, which is associated with each stochastic transition and was defined in the aforementioned formal definition of the HCSPN, and $m(\tau)$ is the current marking of the transition's pre-places. We repeat the continuous simulation until time $\tau = s$ such that $f(\tau|t) = \xi$. The mathematical derivation which is presented in [ACT+04] proves that a stochastic event will occur at time $\tau = s$, which means that we can execute the stochastic simulation at that time. Then we update the current marking according to the fired transitions using the arcs' weights which connect this fired transition with their pre-places and then we advance the simulation time. The previous steps

are then repeated until we reach the end of simulation time. In the following we present the algorithm in a more formal way.

1. Start by the initial marking m_0 and the initial time $t = t_0$;
2. Generate an exponentially distributed random variable ξ.
3. Set $g(\tau|t) = 0$ and simulate the continuous transitions using the ODE solver starting at time $\tau = t$ and progress $g(\tau|t)$ according to equation (3) Until time $\tau = s$ such that $g(\tau|t) = \xi$.
4. Perform the stochastic simulation using the discrete transitions.
5. Update the current marking $m(t)$ according to the fired transitions.
6. Repeat steps 2-5 until we reach the end of simulation time .

4 Examples

In this section, we demonstrate by examples how the HCSPN is used to model biological systems. The two examples which are presented here are: the genes operons model and the modeling of the role of LL-6R in regulation of early haematopoiesis.

4.1 Two Genes Operons

In this example, we model two genes operons using the HCSPN class. The original model can be found in [MDN+00]. The HCSPN in Fig. 3 describes the transcription of an operon containing two genes. The two genes are represented by two discrete places, $Gene1$, $Gene2$, respectively. The transcription of Gene one is represented by the transition $transcriptionG1$, which is a stochastic transition. This transition is associated with a firing rate function, which determines when this transition fires. After the transcription took place, an amount of concentration which represents the mRNA of Gene one is added to the continuous place $mRNA1$. This concentration value is equal to the rate function of the continuous transition, $transcriptionG1$, multiplied by the weight of the arc connecting transition $transcriptionG1$ with place $mRNA1$. The concentration of the mRNA of $Gene1$ can be degradated continuously, when transition $deg1$ fires, if the value of the place $mRNA1$ is greater than zero. A process called translation can take place depending on the concentration of mRNA. However this process does not change the concentration's value of the $mRNA1$ value. So we choose to connect them using a read arc.

After the translation process took place, the protein of Gene one which is represented by the continuous place $Protein1$ can be degraded, when the transition labeled $Degprotein1$ fires. A similar story can happens to Gene two after the polymerase of the RNA of Gene one into Gene two. The firing rate functions of the stochastic transitions and the rates of the continuous transitions can be specified by the user by selecting between a set of kinetic rate functions among them is the mass action kinetics. This example demonstrate by a simple way the modeling power of the HCSPN in system biology.

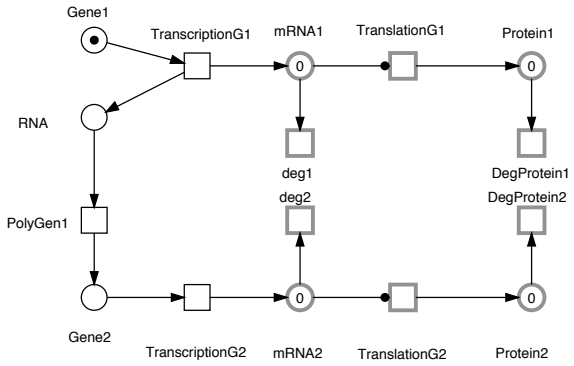


Fig. 3. Two Genes operon model

4.2 The Role of LL-6R in Regulation of Early Haematopoiesis

After we presented a simple example to illustrate the different elements of the HCSPN class, in this section we present a more realistic biological example, modeling the role of a specific cytokine, interleukin-6, in the regulation of early hoematopoiesis [TTC+06]. Fig.4 shows the modeling of this pathway using the HCSPN Petri net. Haematopoiesis is a complex phenomena beadings to the continuous production of all types of mature blood cells. The use of hybrid Petri nets to model the regulation of early haematopoiesis is motivated by the need of discrete elements for modeling the cellular evaluation, as well as continuous elements to model molecular interactions [TTC+06].

Consequently, the model of the IL-6R regulation of the early haematopoiesis consists of two submodels: the cellular submodel and the molecular one. In the former the three different cells types, equiescent, permissive, and committed cells are modeled by three discrete places, Pq, Pp, and C, respectively. Deterministic transitions are used to model the biological processes which take place between these cells types. In the later submodel, continuous places model the molecules involved in the regulation of the haematopoiesis by IL-6, while biological processes are modeled using continuous transition. The bright gray arcs represent the positive feedback loop involving the sLL-6R. Note that in the cellular submodel, arcs weight equal to one are not displayed.

The resulting hybrid Petri net model can be simulated (continuously and stochastically). Because there are no stochastic transitions in this model, the stochastic simulation is simplified to simulate the firing of the discrete Petri net submodel.

5 Conclusions and Future work

In this paper we have presented our research in progress of defining and implementing a hybrid continuous stochastic Petri net class which includes both

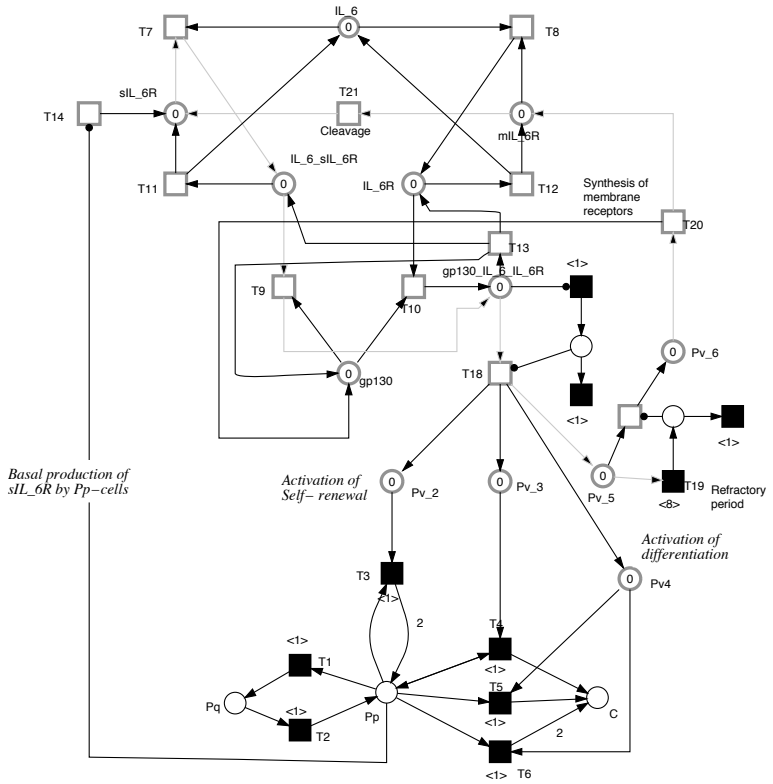


Fig. 4. HCSPN Model of role of LL-6R in regulation of early haematopoiesis

discrete and continuous modeling capabilities of biochemical interactions. The presented class is intended to model systems which are stiff, i.e contain some species with high number of molecules as well as species with low number.

Snoopy supports the export of drawn models to many other tools. For the hybrid class it can be exported to Modelica's hybrid Petri net library [PB09] for further simulation.

Our hybrid model is based on fixed partitioning of the biochemical system, i.e. the reactions are initially divided into discrete and continuous parts. Further extension of this work aims to permit the dynamic partitioning of the reactions during the simulation based on some criterias like the number of molecules in each species or the reaction propensity.

6 Acknowledgement

Mostafa Herajy is supported by the GERLS (German Egyptian Research Long Term Scholarships) program, which is administered by the DAAD in close co-operation with the MHESR and German universities.

References

ACT+04. Alfonsi, A., Cances, E., Turinici, G., Ventura, BD., Huisinga, W. : Exact simulation of hybrid stochastic and deterministic models for biochemical systems. Rr-5435, INRIA-Rocquencourt, 2004.

ACT05. Alfonsi, A., Cances, E, Turinici, G., et al :Adaptive simulation of hybrid stochastic and deterministic models for biochemical systems. In: ESAIM 14, pp 1-13. (2005).

AD98. Alla, H. and David, R.: Continuous and hybrid Petri nets. J. Circ. Syst. Comp. 8, 159-188(1998).

BGH+08. Breitling, R., Gilbert, D., Heiner, M., Orton, R.: A structured approach for the engineering of biochemical network models, illustrated for signalling pathways. Briefings in Bioinformatics 9, 404 - 421 (2008).

DA10. David, R., and Alla, H. : Discrete, Continuous, and Hybrid Petri Nets. Springer, 2010.

GH06. Gilbert, D., Heiner, M.: From Petri nets to differential equations - an integrative approach for biochemical network analysis. In: ICATPN, pp. 181-200. LNCS 4024 Springer (2006).

GHL07. Gilbert, D., Heiner, M., Lehrack, S. : A Unifying Framework for Modelling and Analysing Biochemical Pathways Using Petri Nets. In: 5th International Conference on Computational Methods in Systems Biology pp. 200-216, Springer, Edinburgh(2007).

Gil76. Gillespie, D.T.: A General method for numerical simulation of the stochastic time evolution of coupled chemical reactions. J. Comput Phys 22, 403-437(1976).

HGD08. Heiner, M., Gilbert, D., Donaldson, D.: Petri Nets for Systems and Synthetic Biology. In: Bernardo, M., Degano, P., Zavattaro, G. (eds.): SFM 2008, LNCS, vol. 5016, pp. 215-264, Springer, Heidelberg (2008).

HLG+09. Heiner, M., Lehrack, S., Gilbert, D, Marwan, W.: Extended Stochastic Petri Nets for Model-Based Design of Wetlab Experiments. T. Comp. Sys. Biology 11: 138-163 (2009).

HRR+08. Heiner, M., Richter, R., Rohr, C., Schwarick, M.: Snoopy - A Tool to Design and Execute Graph-Based Formalisms. [Extended Version]. Petri Net Newsletter 74, 8-22(2008).

Kie+04. Kiehl, T., et al: Hybrid Simulation of cellular behavior, J. Bioinformatics 20, 316 -322 (2004).

LCP+08. Li, H., Cao,Y., Petzold, L., Gillespie D.: Algorithms and Software for Stochastic Simulation of Biochemical Reacting Systems, Biotechnology Progress, 24, 56-61(2008).

Mat+03. Matsuno,H., et al: Biopathways Representation and Simulation on Hybrid Functional Petri Net. In Silico Biol. 3, 389-404(2003).

MDN+00. Matsuno, H., Doi, A., Nagasaki, M., Miyano, S. : Hybrid Petri net representation of gene regulatory network. Pac Symp Biocomput., 341-52(2000).

Mur89. Murata, T. : Petri Nets: Properties, Analysis and Applications. Proc. the IEEE 77, 541-580(1989).

Pah09. Pahle, J.: Biochemical simulations: stochastic, approximate stochastic and hybrid approaches. Briefings in Bioinformatics 10, 53-64 (2009).

PB09. Pross, S., Bachmann, B.: A Petri Net Library for Modeling Hybrid Systems in OpenModelica. In: 7th Modelica Conference, pp. 454-462. Italy(2009).

RMH10. Rohr, C., Marwan, W., Heiner, M.: Snoopy - a unifying Petri net framework to investigate biomolecular networks. J. Bioinformatics 26, 974-975(2010) .

RML93. Reddy, V., Mavrovouniotis, M., Liebman, M.: Petri Net Representations in Metabolic Pathways, In: ISMB-93, MIT Press, 328 - 336(1993).

Rue+07. Ruediger, S., et al : Hybrid Stochastic and Deterministic Simulations of Calcium Blips J. Biophysical 93, 1847 - 1857(2007)

SYS+02. Srivastava, R., You, L., Summers, J., Yin, J.: Stochastic vs. Deterministic Modeling of Intracellular Viral Kinetics. J. theor. Biol. 218, 309 -321 (2002).

TK93. Trivedi, K.S., Kulkarni, V.G. : FSPNs: Fluid Stochastic Petri Nets. In: 14th International Conference on Application and Theory of Petri Nets, pp 24-31,(1993).

TTC+06. Troncale, S., Tahi, F., Campard, D., Vannier, JP., Guespin, J. : Modeling and simulation with hybrid functional Petri nets of the role of Interleukin-6 in human early haematopoiesis. Pacic Symp Bio- comput 11:427-438(2006).

IDD-MC - a model checker for bounded stochastic Petri nets

Martin Schwarick

Department of Computer Science, Brandenburg University of Technology
Postbox 10 13 44, 03013 Cottbus, Germany
ms@informatik.tu-cottbus.de

Abstract. IDD-MC is a symbolic analysis tool for bounded stochastic Petri nets with extended arcs. Its engine is based on Interval Decision Diagrams and facilitate the validation of standard Petri net properties, model checking the Computation Tree Logic (CTL) and the Stochastic Continuous Logic (CSL). In this paper we give an informal overview of the currently implemented analysis techniques and report on the most recent extension: the evaluation of rewards. We present some experimental results which show the efficiency of our implementation.

1 Introduction

Stochastic Petri Nets (SPN) are an established formalism for the modeling and analysis of systems, among them biological networks [GHL07,HGD08,HDG09]. Assuming bounded Petri nets, interesting qualitative properties as reversibility and liveness can be determined by applying graph theoretic methods to the reachability graph. Although the complexity of these methods is linear in the size of the reachability graph, its size can grow over exponentially [PW03]. This calls for dedicated techniques as partial order reduction or symbolic state space representation. Here we consider techniques based on an efficient state space encoding using Interval Decision Diagrams (IDD), a generalization of Binary Decision Diagrams (BDD).

In a biological setting tokens often represent molecules or concentration levels. Thus the state space explosion is caused both by concurrency and by a high boundedness degree. The use of IDDs addresses especially the latter issue.

The quantitative semantics of a stochastic Petri net is described by a Continuous time Markov Chain (CTMC). Assuming a net without parallel transitions, the associated CTMC is a graph isomorphic to the reachability graph, but arcs are labeled by firing rates, given by the possibly state-dependent rate functions of the Petri net transitions. In general the rates are given by a sparse matrix indexed by the reachable states. CTMC theory offers a variety of numerical methods for an exhaustive analysis [Ste94]. But in practice either an infinite or a hugh state space of the investigated models are often a very strict limitation. In these cases simulative or approximative methods represent possible alternatives [HRSS10]. However, certain properties or the demand for a high accuracy of the

results require an exhaustive exploration, which in turn calls for an efficient representation of the CTMC and the adaption of the established algorithms. So did we by implementing our SPN analysis engine in IDD-MC which we will present in the next sections.

2 Related Work

There are several tools which offer similar functionality as IDD-MC as for instance the probabilistic model checker PRISM [HKNP06], the SMART tool [CJMS06], Möbius [GKL+09] or GreatSPN [BBC+09] . The most similarities exist regarding to PRISM. Its symbolic engine is based on Multi Terminal Binary Decision Diagrams (MTBDD) which turn out to be not the best solution when dealing with biological networks [SH09] and thus inspired the implementation of IDD-MC's stochastic analysis features.

3 IDD-MC

IDD-MC is a tool for symbolic state space based analysis; first for bounded Petri nets with extended arcs, recently also for bounded stochastic Petri nets. The symbolic engine is based on Reduced Ordered Interval Decision Diagrams (ROIDDs), IDDs for short, which represent a canonical representation for interval logic functions used to encode sets of states (markings) of bounded Petri nets; see [Tov08] for a detailed discussion. It offers a very efficient implementation of IDDs and related operations, among them dedicated operations for the firing of Petri net transitions.

3.1 Overview

Upon this IDD engine the following qualitative analysis features have been realized:

Efficient state space generation. Three state space generation algorithms have been implemented; common breath-first-search, transition-chaining and saturation. The latter algorithm is highly efficient concerning runtime and memory consumption by exploiting the locality of transition firing.

Basic Petri net properties. The tool allows to check for reversibility and liveness of transitions. Therefor efficient decomposition of the strongly connected components has been implemented.

CTL model checking. Given a bounded Petri net N and a Computation Tree Logic (CTL) formula φ (see [CGP01] for an introduction), the model checking problem is to decide whether φ holds in the initial state of N. The classical CTL model checking algorithm [CES86] can be adapted to solve the problem using a symbolic representation of the reachable states of N. It determines for each subformula ψ of φ the set of states fulfilling ψ starting from the innermost formulas. Then it proceeds such that when processing a formula, the set of fulfilling states have been determined for all its subformulas. A CTL subformula

can either be a state formula or a path formula containing a temporal operator. While a state formula can be evaluated locally in a state, a path formula requires to evaluate the paths starting in a state. In a symbolic setting this can be solved by fixpoint computations. The top-formula φ is *true* if the initial state is contained in its associated set.

In addition to qualitative analysis of bounded Petri nets IDD-MC offers the following quantitative analysis techniques for bounded stochastic Petri nets:

Transient analysis. Transient analysis is the computation of the probability distribution at a certain time point τ starting with a certain initial probability distribution. One of the standard techniques is uniformization. The basic idea is to embed a special discretization of the CTMC into a Poisson process which has the same probability distribution at time τ. Its computation reduces to truncate an infinite sum of matrix-vector multiplications. For a detailed description and further techniques see [Ste94]. IDD-MC realizes transient analysis by applying an on-the-fly multiplication algorithm, which does not require an explicit encoding of the CTMC's rate matrix.

Steady State analysis. Continuous time Markov Chains often reach finally a stable probability distribution, which is called the steady state. It can be interpreted as the transient probability distribution for infinite time. Computing the steady state probabilities means to solve a linear system of equations. Iterative methods as Jacobi and Gauss-Seidel are the favored techniques. IDD-MC offers Jacobi, Gauss-Seidel and Pseudo-Gauss-Seidel [Par02] solver based on our on-the-fly multiplication.

CSL model checking. The qualitative analysis techniques, transient and steady state analysis are the needed ingredients to realize model checking of the Continuous Stochastic Logic (CSL) introduced in [ASSB00]. Beeing an stochastic adaption of CTL, the basic model checking procedure is similar. The evaluation of path formulas with time-bounded temporal operators can be achieved by applying transient analysis as proposed in [BHHK00] and implemented in our tool. In [BHHK00], CSL has been extended by a special steady state operator and temporal operators without time bounds. Untimed operators require to solve a linear system of equations based on the Embedded Markov Chain of the original CTMC applying one of the iterative methods.

3.2 Efficiency issues

There are several aspects which are significant for the efficient implementation of the mentioned analysis techniques.

Variable order. It is well known that the variable order affects the size of the decision diagram (DD) and thus has a significant impact on the runtime and memory consumption of the DD implementation. What this means in practice can be seen in [SH09]. IDD-MC uses heuristics [Noa99] based on the Petri net structure to compute static variable orders which produce small-sized DDs in most cases. Furthermore, the tool makes use of so-called *Shared IDDs* which means that several IDD-instances reference to the same set of IDD nodes. This allows, for instance, to check for equality in constant time.

CTMC representation. One of the most challenging problems with quantitative analysis of an SPN is the representation of the rate matrix of its induced CTMC. There are established symbolic techniques as MTBDDs or Kronecker products implemented in existing tools [MP04]. But these techniques may fail under special conditions. For instance, a MTBDD representation suffers from an high amount of distinct non-zero values in the matrix and from a high number of BDD variables caused by an high boundedness degree of the model [SH09].

IDD-MC's numerical engine is based on an on-the-fly approach, which has also been considered for an explicit state space representation in [DSS97] and symbolically in [Sch08,SH09]. In our case the CTMC is represented by the Petri net structure, the reachable states encoded as an IDD, and the rate functions of the transitions. A multiplication of the matrix and a vector is realized by traversing the IDD for all transitions of the net. The traversation simulates the firing of all enabled transitions for all states. Therefor we consider the pre- and post conditions to compute the index of the source and target state and to collect the arguments for the possibly state-dependent rate functions. To achieve this, the IDD has been augmented with certain index informations; see [ST10] for more information. To prevent from unnecessary recomputations, we use the caching strategy from [Par02] adapted to our special settings. The multiplication operation *multiply(StateSet S, TransitionSet T, Vector argument, Vector result)* is implemented configurable concerning the set of considered states S and transitions T. This allows, for instance, to parallelize a single multiplication given a suitable state space partition. In [HRSS10] we present results were we achieved a speedup close to the number of physical cores of current multi-core work stations. Recently we extended our tool by reward structures. In the next section we briefly sketch how to incorporate rewards and their analysis.

4 Rewards

Rewards (also interpretable as costs) define additional measures for probabilistic models and can be associated to states and transitions. They are specified by possibly state-dependent reward functions. A reward for a state will be accumulated and weighted with the time the system remains in it. A transition reward is acquired each time a transition fires.

Conform to the probabilistic model checker PRISM rewards can be added to a model by specifying a reward structure, a collection of state and transition reward items.

A state reward item consists of a guard defining a set of states and the reward function. A transition reward item defines for a certain Petri net transition a reward function. Additionally a guard may restrict the set of enabling states of the transition for which the reward should be considered.

Each state reward item defines a vector which assignes to all states satisfying the associated guard the reward value computed using the reward function. Each transition reward item defines a matrix where all entries representing a state transition caused be the associated Petri net transition and starting in

states satisfying the guard. Each reward structure thus defines a vector representing the sum of all reward vectors defined by the state reward items and a transition reward matrix representing the sum of the reward matrices defined by the transition reward items of the structure. These vectors and matrices are in the dimension of the reachable states and introduce the same problem as the representation of the CTMC. In PRISM reward structures are encoded by MTBDDs.

Since IDD-MC follows a matrix free strategy, we use our on-the-fly multiplication to compute the rewards when needed. A reward item is represented by a set of states and an additional implicit Petri net transition. When adding a reward structure to a stochastic Petri net, the tool adds a new transition for each defined reward item. For a state reward, the rate function of the new implicit transition is the reward function itself. For a transition reward the reward function multiplied by the rate function of the referenced Petri net transition becomes the rate function of the new implicit transition. This enables the computation of an entry-wise matrix product which is required by the analysis of transition rewards [KNP07].

For a state reward, the assigned set of states is the subset of the reachable states satisfying the specified guard. For a transition reward, the assigned set of states are the reachable enabling states of the referenced Petri net transition satisfying the specified guard. Pre- and post conditions of these new transitions are configured in that way that the transition is enabled in every reachable state and its firing does not change the system state but allows to compute the reward.

For the analysis of a reward-augmented model we extend (again conform to PRISM) CSL by the special reward operator R and four new state formulas. Given a reward structure r, a real valued reward bound b and an operator $\bowtie \in \{>, \geq, <, \leq\}$ we define the following formulas:

1. $R\{\text{“}r\text{”}\}_{\bowtie b}[C_{\leq t}]$ The expected cumulative reward within the first t time units is $\bowtie$ than b.
2. $R\{\text{“}r\text{”}\}_{\bowtie b}[I_{=t}]$ The expected state reward at time t is $\bowtie$ than b.
3. $R\{\text{“}r\text{”}\}_{\bowtie b}[F\phi]$ The expected cumulated reward until the first time a state is reached satisfying the state property ϕ is $\bowtie$ than b.
4. $R\{\text{“}r\text{”}\}_{\bowtie b}[S]$ The expected long-run average reward is $\bowtie$ than b.

Because of the given space limitations we can not discuss the evaluation of all these formulas and refer to [KNP07]. However the basic step is to compute initially a single vector of reward values in the size of the state space. In the case of the cumulative operator $C_{\leq t}$, for instance, the vector represents the sum of the state reward vector and a vector containing the row sums of a matrix achieved by an entry-wise multiplication of the CTMC rate matrix and the matrix representing the transition rewards. This pre-computation step can be simply realized using our transition-based reward representation and our multiply operation.

To compute the state reward vector of a reward structure we apply the multiplication successively to all implicit transitions representing state reward items. The argument vector is initialized with 1 in every entry. Since transitions do not change the state by firing, this means to extract the elements of the diagonal of

the matrix, belonging to rows defined by a state set satisfying the guard. The computed rate of these implicit state transitions are the actual state rewards and are added to the result vector. Actually we are computing the row sums of a matrix where only one element per row is non-zero and add these sums to our result vector.

Obviuosly, we will not set the result vector to zero after a multiplication. We similarely compute the entries of transitions reward matrix just considering the transitions representing transition reward items.

For the mentioned case of the cumulative operator, we start off with a zero result vector and apply the multiplication sequentially for all implicit transitions defined by the reward structure, state rewards as well transition rewards.

4.1 Experimental results

To demonstrate the efficiency of our tool we present some experimental results. We run IDD-MC and PRISM−3.3.1 on a 8×2.26 GHz MAC Pro with 32 GB RAM with eight physical cores (with hyper-threading 16 logical cores). We consider a stochastic Petri net model of the Mitogen activated protein kinase cascade (MAPK) [HF96]. The Petri net model has been created with our tool Snoopy [RMH10]. The PRISM model and the reward structure were taken from the PRISM case study suite. Snoopy implements an export mechanism to create PRISM models including the computation of the same static variable orders as IDD-MC. Thus we used for our experiments Snoopy also to create another model description in the PRISM language. The used CSL formula $\mathbf{R}\{\text{“}time\text{”}\}_{=?}[\mathbf{F}(kpp = N)]$[1] was taken from [KNP08]. This formula requires the generation of the Embedded Markov Chain. To solve the linear system of equations we used the Jacobi method, the default in both tools. In our experiments we varied the number of tokens on certain places using parameter N of the scalable model, which comprises 22 places and 30 transitions. The number of reachable states represents the dimension of the rate matrix, the number of transitions represents the number of non-zero matrix entries. We used IDD-MC with one and with 16 threads.

The figures in Table 1 show that a good variable order reduces the analysis time significantly. Using all logical cores of our test system allows a speed up about factor seven. Except the very small state space for $N = 2$ and $N = 4$ IDD-MC outperforms PRISM for the given model and formula also without multi-threading, and using the same variable order.

5 Conclusion

We have presented IDD-MC, a symbolic tool for qualitative and quantitative analysis of bounded stochastic Petri nets. We summarized its main features, especially the analysis of reward structures. Experiments show that the tool

[1] To use '$_{=?}$' means to compute the reward for the initial state

N	states	transitions	IDD-MC$_1$	IDD-MC$_{16}$	PRISM$_{org}$	PRISM$_{go}$
2	2,172	13,608	1s	2s	0.7s	0.2s
4	99,535	910,872	23s	5s	10min15s	18s
6	1,373,026	15,015,264	4m26s	40s	8h48m49s	33m41s
8	10,276,461	125,012,862	30m18s	4m30s	–	6h26m32s
10	52,820,416	690,183,846	164m22s	22m30s	†	–

– means, that we aborted the experiment after 24h.

† menas, that we skipped the experiment.

Table 1. CTMC sizes for different initial markings of the MAPK cascade and the model checking times to evauluate the CSL formula $\mathbf{R}\{\text{“}time\text{”}\}_{=?}[\mathbf{F}(kpp = N)]$ with IDD-MC and PRISM.

outperforms the PRISM model checker for the considered case study. We intend to support Generalized stochastic Petri nets (GSPN) and to realize out-of-core techniques. Furthermore there are several aspects for performance optimization as improving variable ordering and parallelization.

References

[ASSB00] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model checking continuous-time Markov chains. *ACM Trans. on Computational Logic*, 1(1), 2000.

[BBC+09] S. Baarir, M. Beccuti, D. Cerotti, M. De Pierro, S. Donatelli, and G. Franceschinis. The GreatSPN tool: recent enhancements. *SIGMETRICS Perform. Eval. Rev.*, 36(4):4–9, 2009.

[BHHK00] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model checking continuous-time Markov chains by transient analysis. In *Proc. CAV 2000*, pages 358–372. LNCS 1855, Springer, 2000.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Systems*, 8(2):244–263, 1986.

[CGP01] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2001.

[CJMS06] G. Ciardo, R. L. Jones, A. S. Miner, and R. Siminiceanu. Logical and stochastic modeling with SMART. *Performance Evaluation*, 63(1), 2006.

[DSS97] Daniel Deavours, William H. Sanders, and William H. S. On-the-Fly solution techniques for stochastic petri nets and extensions. In *IEEE Transactions on Software Engineering*, pages 132–141, 1997.

[GHL07] D. Gilbert, M. Heiner, and S. Lehrack. A unifying framework for modelling and analysing biochemical pathways using Petri nets. In *Proc. CMSB 2007*, pages 200–216. LNCS/LNBI 4695, Springer, 2007.

[GKL+09] S. Gaonkar, K. Keefe, R. Lamprecht, E. Rozier, P. Kemper, and W. H. Sanders. Performance and dependability modeling with Möbius. *SIGMETRICS Perform. Eval. Rev.*, 36(4):16–21, 2009.

[HDG09] M. Heiner, R. Donaldson, and D. Gilbert. *Petri Nets for Systems Biology, in Iyengar, M.S. (ed.), Symbolic Systems Biology: Theory and Methods*. Jones and Bartlett Publishers, Inc., to appear, 2009.

[HF96] C. Huang and J. Ferrell. Ultrasensitivity in the mitogen-activated protein kinase cascade. *Proc. Natl. Acad. Sci.*, 93:10078–10083, 1996.

[HGD08] M. Heiner, D. Gilbert, and R. Donaldson. Petri nets in systems and synthetic biology. In *SFM*, pages 215–264. LNCS 5016, Springer, 2008.

[HKNP06] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. TACAS 2006*, pages 441–444. Springer, LNCS 3920, 2006.

[HRSS10] M. Heiner, C. Rohr, M. Schwarick, and S. Streif. A comparative study of stochastic analysis techniques. In *Proc. CMSB 2010*. Springer, 2010.

[KNP07] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of *LNCS (Tutorial Volume)*, pages 220–270. Springer, 2007.

[KNP08] M. Kwiatkowska, G. Norman, and D. Parker. Using probabilistic model checking in systems biology. *ACM SIGMETRICS Performance Evaluation Review*, 35(4):14–21, 2008.

[MP04] A. Miner and D. Parker. *Validation of Stochastic Systems: A Guide to Current Research*, chapter Symbolic Representations and Analysis of Large Probabilistic Systems, pages 296–338. LNCS 2925. Springer, 2004.

[Noa99] A. Noack. A ZBDD Package for Efficient Model Checking of Petri Nets (in German). Technical report, BTU Cottbus, Dep. of CS, 1999.

[Par02] D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems*. PhD thesis, University of Birmingham, 2002.

[PW03] L. Priese and H. Wimmel. *Theoretical Informatics - Petri Nets (in German)*. Springer, 2003.

[RMH10] C. Rohr, W. Marwan, and M. Heiner. Snoopy–a unifying Petri net framework to investigate biomolecular networks. *Bioinformatics*, 26(7):974–975, 2010.

[Sch08] M. Schwarick. Transient Analysis of Stochastic Petri Nets with Interval Decision Diagrams. In *Proc. 15th German Workshop on Algorithms and Tools for Petri Nets (AWPN 2008)*, volume 380 of *CEUR Workshop Proceedings*, pages 43–48. CEUR-WS.org, September 2008.

[SH09] M. Schwarick and M. Heiner. CSL model checking of biochemical networks with interval decision diagrams. In *Proc. CMSB 2009*, pages 296–312. LNCS/LNBI 5688, Springer, 2009.

[ST10] M. Schwarick and A. Tovchigrechko. IDD-based model validation of biochemical networks. *Theoretical Computer Sience*, 2010.

[Ste94] W.J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton Univ. Press, 1994.

[Tov08] A. Tovchigrechko. *Model Checking Using Interval Decision Diagrams*. PhD thesis, BTU Cottbus, Dep. of CS, 2008.

Simulative CSL model checking of Stochastic Petri nets in IDD-MC

Christian Rohr

Magdeburg Centre for Systems Biology (MaCS), Otto von Guericke University
and Max Planck Institute for Dyn. of Complex Tech. Syst. Magdeburg, Germany
rohr@mpi-magdeburg.mpg.de

Abstract. IDD-MC is a symbolic analysis tool for bounded Stochastic Petri nets. The restriction regarding the boundedness can be circumvented by a simulative approach. Besides that, the simulation is going to be capable of handling extended Stochastic Petri nets. In this paper we report on the integration of a multi-scaling stochastic simulation engine into IDD-MC. We present some experimental results which show the efficiency of our implementation.

1 Introduction

Stochastic models are becoming more and more popular in Systems Biology and their size increases while the understanding of the modelled networks grows. Often, the analysis of these systems with exact numerical algorithms is not feasible anymore. Apart from that, stochastic models with an unbounded state space can not be analyzed with such techniques at all. Both restrictions can be circumvented by stochastic simulation.

For modelling biological systems we are using stochastic Petri nets ($\mathcal{SPN}$). So we can optimize the quantitative analysis using structural information [HGD08]. The semantics of a stochastic Petri net is defined by a Continuous Time Markov Chain (CTMC).

Generalised stochastic Petri nets ($\mathcal{GSPN}$) extend $\mathcal{SPN}$ by introducing immediate transitions. These transitions have a higher priority then stochastic transitions. This means, if an immediate and a stochastic transition are enabled at the same time, the immediate transition has to fire. One can model very fast reactions or switch like behavior with such transitions. The semantics of generalised stochastic Petri nets can be reduced to CTMC.

The class of extended stochastic Petri nets ($\mathcal{XSPN}$) is based on $\mathcal{GSPN}$, but introduces deterministic and scheduled transitions [HLGM09]. Both transition types have the same priority, which is higher than the priority of the stochastic transitions but lower than the priority of immediate transitions. These transitions are used, e.g, to model external influences, taking place at certain time points. The use of deterministically timed transitions in extended stochastic Petri nets destroys the Markovian property [Ger01]. The stochastic simulation can be adapted in a way that it can handle $\mathcal{XSPN}$.

2 Stochastic Simulation

We implemented the stochastic simulation algorithm (SSA) introduced by Gillespie in [Gil77]. The algorithm creates a single finite path through the possibly infinite CTMC. The computation of such a simulation run (trajectory, path) needs only to store the current state. The basic idea is as follows.

Given the system is at time point τ in state s. The probability that a transition $t_j \in T$ will occur in the infinitesimal time interval $[\tau, \tau + \Delta\tau)$ is given by:

$$P(\tau + \Delta\tau, t_j \mid s) = h_j(s) \cdot e^{-E(s)\cdot\Delta\tau} \tag{1}$$

For each transition t_j, the rate is given by the propensity function h_j, where $h_j(s)$ is the conditional probability that transition t_j occurs in the infinitesimal time interval $[\tau, \tau + \Delta\tau)$, given state s at time τ. So, the enabled transitions in the net compete in a race condition. The fastest one determines the next state and the simulation time elapsed. In the new state, the race condition starts anew.

Algorithm 1 Stochastic simulation algorithm.

Require: $\mathcal{SPN}$ with initial state s_0, time interval $[\tau_0, \tau_{max}]$
Ensure: state s at timepoint τ_{max}

```
 1: initRand(seed)
 2: time τ := τ_0
 3: state s := s_0
 4: while τ < τ_max do
 5:     draw random numbers r_1, r_2, uniformly distributed on [0, 1)
 6:     r_1 := getURand()
 7:     r_2 := getURand()
 8:     Δτ = − ln(r_1) / E(s)
 9:     e := 0
10:     for all transitions t_j ∈ T enabled at s do
11:         e := e + h_j(s)
12:         if e > r_2 · E(s) then
13:             s := s + Δt_j
14:             break
15:         end if
16:     end for
17:     τ := τ + Δτ
18: end while
```

The SSA simulates every transition firing (basically by using Eq. (1)) one at a time, and keeps track of the current system state. To determine the time increment $\Delta\tau$ and to select the next Petri net transition to fire requires to generate two random numbers $(r1, r2)$ uniformly distributed on $(0, 1)$. Different trajectories of the CTMC are obtained by different initializations of the random number generator (line 1). Reliable conclusions about the system behaviour require many simulations due to the stochastic variance.

The simulative processing of immediate, deterministic and scheduled transitions is rather straightforward, see [Ger01]. In short, the Algorithm 1 needs to be extended in two ways.

- After every firing of a Petri net transition (line 13), it needs to be checked whether immediate transitions got enabled. If so, these have to be processed until no more immediate transitions are enabled. This possibly leads to a time deadlock, if there exists a cyclic path of immediate transitions.
- Having calculated the next time step (line 8), it needs to be checked whether a deterministic or scheduled transition gets enabled in the time interval $[\tau, \tau + \Delta\tau]$. If yes, the one closest to τ is processed and the simulation time will be set to the value of this transition.

2.1 Model checking.

We use the Continuous Stochastic Logic (CSL) introduced by [ASSB00]. In principle the stochastic simulation algorithm allows to check any unnested, time-bounded CSL formula without the steady state operator [YS02]. The ratio of the number of fulfilling and total number of runs leads to an approximation of the desired probability.

To achieve an appropriate accuracy of the results, one has to determine the required amount of simulation runs. The method of our choice is the confidence interval as described in [SM08]. The confidence interval contains the property of interest with some predefined probability, called confidence level. This confidence level has usually values of 90%, 95%, or 99%. Assuming 95% and an accuracy of the results of 10^{-5} leads to $\approx 38,000,000$ runs.

2.2 Parallelization.

Such a high amount of independent simulation runs requires parallelization. Because of the independence of the individual simulations runs, parallelization is straightforward. It basically requires a master, which distributes the work load on n identical slaves and collects the results.

This scenario can be realized in two different ways:

Multithreading is the method of choice, if the program is meant to run on a symmetric multiprocessing (SMP) computer, where all processors have access to the main memory. The master thread creates n worker threads and sets the required work load for each of them. In the end each worker sends the results back to the master thread.

Multitasking plays its strength in a distributed memory environment like computer clusters, where not all memory is available to all processors. In our implementation we use the Message Passing Interface (MPI). That provides special communication patterns. In the beginning, n processes are created and the master uses the "broadcast" operation to distribute the work load on the processes. When the simulation is finished, the results are collected from the processes. For that purpose the "gather" operation is used.

3 Case Study

We use the abstract circadian clock model of Barkei and Leiber, introduced in [BL00]. It shows circadian rhythms which are widely used in organisms to keep a sense of daily time. More background information can be found in [VKBL02].

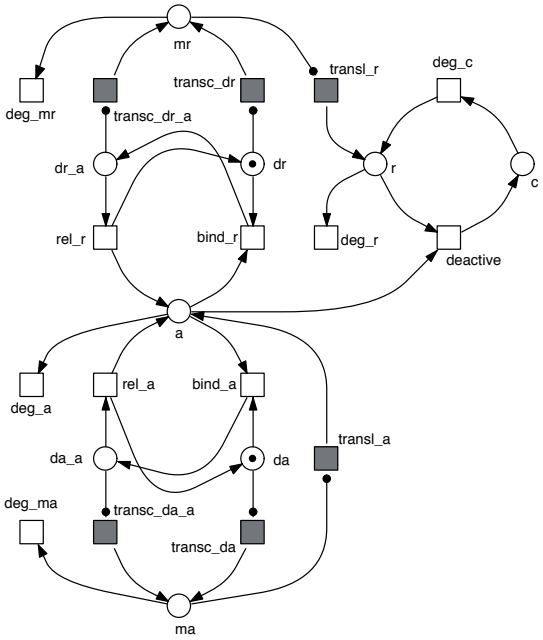


Fig. 1. $\mathcal{SPN}$ of the abstract circadian clock model

We modeled it as a stochastic Petri net (Fig. 1) containing 9 places and 16 transitions. All transitions use mass-action kinetics and the parameters were taken from [VKBL02]. The Petri net is unbounded, because once a transition with prefix "trans" got enabled (there are 6 of them) it could create an endless amount of token on its post place.

The second case study is a gene-regulation network. The Lactose-Operon model [Wil06] models the transport and metabolism of lactose in bacteria. This $\mathcal{XSPN}$ (Fig. 2) taken from [HLGM09] contains the scheduled transition "Intervention", which increases the amount of "Lactose" by 10,000 each 50,000 time units. The Petri net contains 11 places and 17 transitions and is unbounded too. All stochastic transitions use mass-action kinetics with parameters from [Wil06].

Both case studies are modeled with the generic graph editor Snoopy [RMH10]. In order to show the scalability of our implementation we decided to generate averaged traces until time point $\tau = 100$ for the circadian clock model and $\tau = 130,000$ for the lac-operon model.

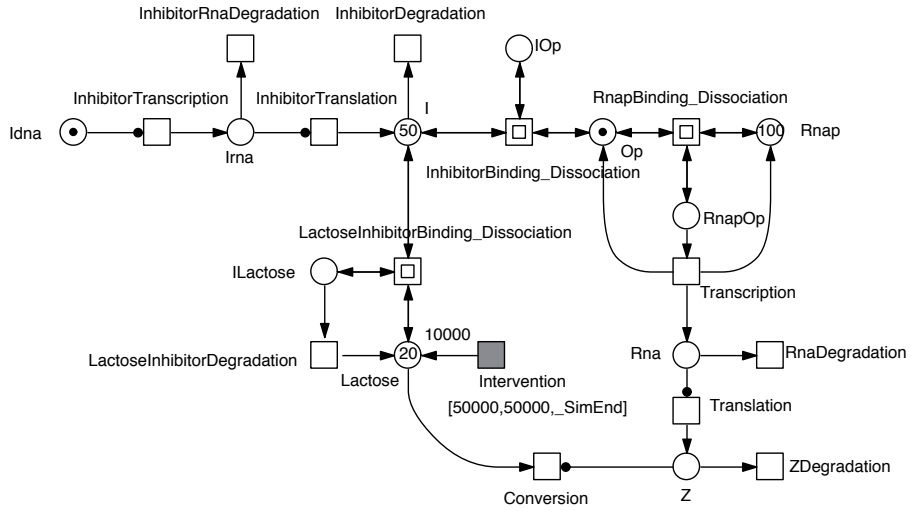


Fig. 2. $\mathcal{XSPN}$ of the lac operon model

Table 1. Comparison of the runtime for n different workers for the multi-threaded (MT) and the MPI version. The speedup is given in braces behind the time value. We created 10,000 simulation runs until $\tau = 100$ for the circadian clock model (index 1) and 1,000 simulation runs until $\tau = 130,000$ for the lac operon model (index 2).

	Circadian clock		Lac operon	
n	MT_1	MPI_1	MT_2	MPI_2
1	15m43s (1×)	20m27s (1×)	5m34s (1×)	7m12s (1×)
2	7m52s (2×)	9m55s (2.1×)	2m49s (2×)	3m33s (2×)
4	4m05s (3.8×)	5m08s (4×)	1m32s (3.6×)	1m50s (3.9×)
8	2m50s (5.5×)	2m33s (8×)	59s (5.7×)	56s (7.7×)
12	2m05s (7.5×)	1m42s (12×)	45s (7.4×)	37s (11.7×)
16	1m42s (9.2×)	1m18s (15.7×)	40s (8.4×)	31s (13.9×)

The experiments considering the multi-threaded version of the simulation engine were done on a 2.26 GHz Apple Mac Pro with 32 GB RAM and eight physical (with hyper-threading 16 logical) cores. IDD-MC was build on Mac OS X 10.5.8 as 64-bit application. The experiments with the MPI version were done on a cluster with 96 cores distributed on 24 nodes. It was build on CentOS 5.5 as 64-bit application.

Table 1 shows the result of our experiments. The runtime decreases well with an increasing number of workers. The scaling of the multi-threaded version is correlating with the number of workers up to $n = 4$, after that it goes down a bit. This is due to the 2 processors, each with 4 cores and 8 threads. The MPI version scales linearly over all settings.

4 Conclusion

We extended the analysis capabilities of IDD-MC [SH09] by implementing a stochastic simulation engine. We verified the scalability of the parallelized versions, using multithreading; and multitasking.

For the time being the stochastic simulation only provides transient analysis and generation of averaged traces. In the future we intend to support unnested time-bounded CSL formulas without steady state operator. We plan to closer investigate the possibilities of computing the steady state using stochastic simulation.

IDD-MC is available for non-commercial use [IDD10]; it includes the MT version. The MPI version is available on request.

References

[ASSB00] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model checking continuous-time Markov chains. *ACM Trans. on Computational Logic*, 1(1), 2000.

[BL00] N. Barkai and S. Leibler. Biological rhythms: Circadian clocks limited by noise. *Nature*, 403:267–268, 2000.

[Ger01] R. German. *Performance analysis of communication systems with non-Markovian stochastic Petri nets*. Wiley, 2001.

[Gil77] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340 – 2361, December 1977.

[HGD08] M. Heiner, D. Gilbert, and R. Donaldson. Petri nets in systems and synthetic biology. In *SFM*, pages 215–264. LNCS 5016, Springer, 2008.

[HLGM09] M. Heiner, S. Lehrack, D. Gilbert, and W. Marwan. Extended Stochastic Petri Nets for Model-Based Design of Wetlab Experiments. pages 138–163. LNCS/LNBI 5750, Springer, 2009.

[IDD10] IDD-MC Website. A model checker for the Continuous Stochastic Logic (CSL) of stochastic Petri nets. BTU Cottbus, http://www-dssz.informatik.tu-cottbus.de/software/iddcsl/latest/iddcsl.html, 2010.

[RMH10] C. Rohr, W. Marwan, and M. Heiner. Snoopy–a unifying Petri net framework to investigate biomolecular networks. *Bioinformatics*, 26(7):974–975, 2010.

[SH09] M. Schwarick and M. Heiner. CSL model checking of biochemical networks with interval decision diagrams. In *Proc. CMSB 2009*, pages 296–312. LNCS/LNBI 5688, Springer, 2009.

[SM08] W. Sandmann and C. Maier. On the statistical accuracy of stochastic simulation algorithms implemented in Dizzy. In *Proc. WCSB 2008*, pages 153–156, 2008.

[VKBL02] J. Vilar, H.-Y. Kueh, N. Barkai, and S. Leibler. Mechanisms of noise-resistance in genetic oscillators. *Proc. National Academy of Sciences of the United States of America*, 99(9):5988–5992, 2002.

[Wil06] D.J. Wilkinson. *Stochastic Modelling for System Biology*. CRC Press, New York, 1st Edition, 2006.

[YS02] H. Younes and R. Simmons. Probabilistic verification of descrete event systems using acceptance sampling. In *Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 223–235. LNCS 2404, Springer, 2002.

Re-Thinking Process Mining with Agents in Mind

Nils Erik Flick, Lawrence Cabac, Nicolas Denz and Daniel Moldt

Department of Informatics, University of Hamburg,
Vogt-Kölln-Straße 30, 22527 Hamburg, Germany
http://www.informatik.uni-hamburg.de

Abstract. Being able to keep business processes flexible and adapting to changing demands requires knowledge of the processes. The algorithmic generation of explicit models is a step towards making the relevant processes clearly comprehensible and exposing them for reflection.
We believe that processes which occur in practice, notably in organizational and inter-organizational contexts, possess a structure best described by systems of interacting agents.
We intend to carry out a mining of special reference nets which are structured in a multi-agent-like fashion. Process mining of reference nets might allow for structuring beyond mere model aggregation. One possible application of an algorithm providing such interpretations lies in spotting their weaknesses automatically with the aim to support of streamlining of organizational processes.

Keywords: agent technology, business process support, process mining, reference nets, workflow mining

1 Introduction

This paper is a position paper and a research agenda rather than a compendium of results. It serves to disseminate some ideas about a possible future of process mining at an early stage.

Process mining starts from the assumption that a log of activities contains interesting patterns. These patterns are deemed to be generated by a system whose structure must be recovered. In our opinion, this structure can be regarded as a multi-agent system.

Process mining should therefore be conducted from an explicitly agent-oriented perspective. In most cases, an agent-oriented view is either neglected or only implicitly taken in the current literature.

The remainder of the paper is structured as follows: In Section 2, we will position the present research within the emerging process mining landscape.

We will note the aspects we would like to concentrate on, because we perceive them as under-represented to date. These aspects include region theory and the organizational perspective of process mining.

We will work within the framework of reference nets, as defined in Kummer's monograph [1]. These higher-level Petri nets not only allow for the concise expression of concurrent processes (as do other Petri net formalisms) but lend themselves to formalizing multi-agent systems (MAS). Mining reference nets and mining agents are two sides of a medal, if the concept of MAS fits the target system: general reference nets might be too complex to be simply reconstructed from execution traces; agents must be formalized as nets when the goal is to offer an integrated net mining mechanism. Several essential implications of this multi-level, organizational view are examined in Section 3.

2 Relations to Prior Work

First, we note the existence of an algebraic approach for Petri net synthesis, the theory of regions, and its uses in process mining. Next, some aspects of the relationship of process mining to multi-agent systems are discussed and relevant publications noted.

2.1 Process Mining and Region Theory

Region theory has been put forward in its earliest form by Ehrenfeucht and Rozenberg [2]. Put simply, it allows the synthesis of Petri nets from their reachability graphs. This objective is closely related to what (part of) process mining strives to accomplish. Unsurprisingly, it has been turned into a promising avenue for traditional process mining, as judged by the growing number of publications.

The methods of van der Aalst et al. [3], van Dongen et al. [4], Carmona et al. [5], and Bergenthun et al. [6] all rely on converting an event log into a transition system in the first step. This automaton is an abstraction of the control flow represented in the log. In the second step, region theory is applied to turn the automaton into a more compact and possibly concurrent Petri net.

Region theory has to our knowledge only been applied to process mining in the control flow perspective. An interesting question is how region theory can contribute to other mining perspectives and which modifications and extensions of the original formalism might be necessary.

2.2 Agent Communities and Organizational Process Mining

Remarkably, the process mining community has, so far, treated the sequence and causality aspect more thoroughly than the semantic content of actions, maybe because the resulting structures are mostly limited to simple P/T nets.

In reality, *agents* espousing *roles* actually accomplish the actions in most cases. Arguably the agent metaphor applies in all cases, since even an organization can be modeled as an agent. Logs also commonly identify agents as originators of tasks.

Existing algorithms and implementations can mine the causal dependency between activities and also construct a social network. Minseok Song's work (especially in [7]) already shows the relevant data organized into a graph, a useful

decision which points in the direction of greater integration. Beyond that, the ProM framework [8] provides plugins for several algorithms aiming at organizational mining and putting the roles back into the processes (Bozkaya, Gabriels and Werf [9], Song and van der Aalst [10]), including role hierarchy mining.

Despite these possibilities, the results mined from different perspectives are seldom linked to provide better heuristics for structuring a log. Contributions like [11] and our own previous work [12] are a first step towards aggregating multiple perspectives in a common model. Another important property induced by the agent metaphor is often neglected as well: Current process mining techniques hardly pay attention to *locations* where actions take place. One exception is the work presented in [13] where physical places are considered.

Beyond that, our vision is a holistic approach that integrates these different perspectives in the system-theoretical viewpoint of multi-agent systems, based on the formalism of reference nets.

3 Where to Go from Here

Existing process mining methods have one shortcoming which is due to the concentration on the process-centered view *at the exclusion* of other relevant perspectives (see Cabac, Knaak, Moldt and Rölke [12] for one possible classification). There are at least three main points, all unified in a systems-theoretical view of multi-agent systems, where we realize the need for a new approach.

3.1 Agents as Coherent Entities

Certain processes are best expressed in terms of agents, not only because the agents execute the actions, but more pertinently because the concept of agents can be important for *structuring* purposes.

An agent, similar to an object in object-oriented programming, is a system characterized, among other properties, by a degree of coherence/cohesion and persistence of information attached to it. This correspondence can be used to uncover not directly observable dependencies by structuring actions according to (known or unknown) agents.

3.2 Agents as Situated Entities

Agents exist in an environment that is often defined in terms of distinct logical or physical locations with paths the agents can move on (e.g. [14]). Such topologies influence the behaviour of agents in several respects. Generally, an agent's behaviour might depend on its location. The ability to perceive and act might be restricted to a local radius (behavioural locality, e.g. [15]). Locations might serve as side conditions for the synchronization of agent behaviour as in rendez-vous synchronization (see e.g. [16]). The situatedness of agents is naturally represented in the reference net formalism.

As discussed above, locations and locality are seldom regarded in current process mining techniques. In our opinion, the focus on a nets-within-nets formalism can bring forward the handling of location-related information in process mining. On the one hand, locations and their properties might be reconstructed from a log based on hints of characteristic agent behaviour. On the other hand, available information about locations might provide heuristics to improve the reconstruction of process and organizational models.

3.3 Incorporation of Agents into Mining

Since actions are linked to agents, there is much to be gained by incorporating the agent side directly. If this information is ignored, one misses out on potentially crucial clues to understand the process. As an illustration, when a set of agents $\{A_i\}_{i \in I}$ for some $I \subseteq 2^A$ are involved sequentially in a process in certain roles, there must have been a connection between them.

When we observe that an action must happen in a certain sequence with other actions, this means that the agents participating are linked by a network of connections. Thus, the existence of dependencies between actions may establish, or betray, the fact that information must have been propagated.

3.4 Recovering Agents

The agent is 'defined' by its surroundings, at least in that it must reflect its relationships internally. Relationships also shape the ways in which it may interact. Recovering the agents could mean either of several things: (1) Building an operational model of the agent, (2) finding their relationships without trying to build such an agent, and (3) recovering a priori missing/hidden information (which will not be possible without further hints from the logs).

There is clearly a possible path for improving on methods which are blind to agents and only consider named activities, and on the existing approaches to role recovery. One could conceivably start from region theory and generalize regions further, as at present they ignore agents completely.

3.5 Recovering the Organization

Organization means that interdependencies and hierarchy exist. Some would argue that strongly linked subsystems form organizational units. It is unclear how well a 'strength of connections' to hierarchical clustering approach really helps in structuring the unknown domain of agents.

The intuition of clustering by separating weekly clusters can be misleading. Distance measures must be chosen with care with respect to the intended definition of 'similarity'. The agent metaphor provides several hints for similarity including similar behaviour, knowledge, frequency of communication, etc.

One can imagine a number simple examples: two secretaries from different companies who have a logical platform in common and frequently communicate

with each other, are closely linked but at the same time belong to separate organizations. System boundaries thus depend on semantic decisions.

The converse argument that such a naïve analysis can lead to a prejudice-free analysis of the situation is also valid. In a business setting, such a clustering result is useful as far as the goal is to examine the processes and the performance of the existing system on the technological/software side.

Opaque role annotations do not, per se, mean that one understands the relationships existing between the participants. Rather, mining this would be a step forward in doing *meaningful* process understanding (cf. roles as objects). Detecting the kind of relationship is possible because of the patterns generated.

Usually, the process perspective captures the dynamic behavior of the system, whereas the mining of more static, *structural* properties is also of value. There are prototypical kinds of relationship between such entities (see e.g. Jennings and Wooldridge [17]), which leave characteristic traces in the execution log.

4 Conclusions and Outlook

It must be stated that process mining is currently only at the beginning and many aspects have not yet been formally explored.

We have discussed the importance of considering agents as parts of processes in process mining. We argued in favor of integrating the mining of processes, the interrelated network of agents generating them, and the environment in which the agents are located. The agent metaphor naturally leads to the addressing of different perspectives beyond control flow mining as claimed in [18] and [19].

Reference nets might make the right kind of structuring possible. In our reference net-based MAS architecture Mulan [20], protocols, agents, and platforms are all nets, and as such first-order entities that can be reasoned about explicitly in process mining. They can represent other supplementary information as well, such as physical places, logical platforms and object flow.

Our prime objective at this point is to ascertain the possibility of recovering agents and mining static as well as dynamic relationships between them from sufficiently detailed logs. In case of success, further questions arise. As noted before in [21], dealing with noise, i.e. incorrectly logged information, may be a major issue and it is still not clear how to achieve robustness; this must certainly be addressed in further work. There is certainly a conflict between the uncompromising exactness of algebraic methods, the desire to detect even uncommon process variants and the quality of log data available to the analyst (many activities taking place outside of logs).

References

1. Kummer, O.: Referenznetze. Logos Verlag, Berlin (2002)
2. Ehrenfeucht, A., Rozenberg, G.: Partial (set) 2-structures. Acta Informatica **27** (1990) 343–368

3. van der Aalst, W.M.P., Rubin, V., van Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: A two-step approach using transition systems and regions. BPM Center Report BPM-06-30, BPMcenter. org (2006)
4. van Dongen, B.F., Busi, N., Pinna, G., van der Aalst, W.M.P.: An iterative algorithm for applying the theory of regions in process mining. In: Proceedings of the workshop on formal approaches to business processes and web services (FABPWS'07). Siedlce: Publishing House of University of Podlasie. (2007)
5. Carmona, J., Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: A symbolic algorithm for the synthesis of bounded Petri nets. Applications and Theory of Petri Nets (2008) 92–111
6. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. Business Process Management (2007) 375–383
7. van der Aalst, W.M.P., Song, M.: Mining Social Networks: Uncovering interaction patterns in business processes. Business Process Management (2004) 244–260
8. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A., van der Aalst, W.M.P.: The ProM framework: A new era in process mining tool support. Applications and Theory of Petri Nets 2005 (2005) 444–454
9. Bozkaya, M., Gabriels, J., Werf, J.M.: Process Diagnostics: A Method Based on Process Mining. In: International Conference on Information, Process, and Knowledge Management, 2009. eKNOW'09. (2009) 22–27
10. Song, M., van der Aalst, W.M.P.: Towards comprehensive support for organizational mining. Decision Support Systems **46** (2008) 300–317
11. Rozinat, A., Mans, R.S., Song, M., van der Aalst, W.M.P.: Discovering simulation models. Inf. Syst. **34** (2009) 305–327
12. Cabac, L., Knaak, N., Moldt, D., Rölke, H.: Analysis of multi-agent interactions with process mining techniques. In: Multiagent System Technologies. 4th German Conference, MATES 2006 Erfurt, Germany. Proceedings. Volume 4196 of Lecture Notes in Computer Science., Berlin Heidelberg, Springer-Verlag (2006) 12–23
13. Reijers, H., Song, M., Jeong, B.: On the performance of workflow processes with distributed actors: Does place matter? In Alonso, G., Dadam, P., Rosemann, M., eds.: Business Process Management. Volume 4714 of Lecture Notes in Computer Science. Springer Berlin/Heidelberg (2007) 32–47
14. Meyer, R.: Bewegungsgraphen – Ein Konzept für die räumliche Modellierung der Umgebung in der individuen- und agenten-basierten Simulation . In Wittman, J., Bernard, L., eds.: Simulation in Umwelt- und Geowissenschaften – Workshop Münster, Aachen, Shaker (2001) 47–60
15. Klügl, F.: Multiagentensimulation – Konzepte, Werkzeuge, Anwendung. Agententechnologie. Addison-Wesley, Munich (2001)
16. Jessen, E., Valk, R.: Rechensysteme: Grundlagen der Modellbildung. Studienreihe Informatik. Springer-Verlag, Berlin Heidelberg (1987)
17. Jennings, N.: Agent-oriented software engineering. Multi-Agent System Engineering (1999) 1–7
18. van der Aalst, W.M.P., Weijters, A.: Process mining: a research agenda. Computers in Industry **53** (2004) 231–244
19. Rembert, A.J., Ellis, C.S.: An initial approach to mining multiple perspectives of a business process. In: TAPIA '09: The Fifth Richard Tapia Celebration of Diversity in Computing Conference, New York, NY, USA, ACM (2009) 35–40
20. Rölke, H.: Modellierung von Agenten und Multiagentensystemen: Grundlagen und Anwendungen. PhD Thesis, Faculty of Informatics, University of Hamburg (2004)
21. van der Aalst, W.M.P.: Discovering coordination patterns using process mining. In: Workshop on Petri Nets and Coordination, Bologna, Italy, Citeseer (2004)

Helper Agents as a Means of Structuring Multi-Agent Applications

Kolja Markwardt and Daniel Moldt

University of Hamburg, Department of Informatics,
Vogt-Kölln-Str. 30, D-22527 Hamburg
http://www.informatik.uni-hamburg.de/TGI

Abstract The Paose methodology of software engineering uses Multi Agent Systems as its main way of structuring applications. However as systems get larger and more complex, additional layers of abstraction are needed. Therefore we propose the HeRA-system (short for HElper and Resource Agents) to structure agent systems. In this paper we introduce the main concepts of HeRA and illustrate via a small example the usage of its prototypical implementation.

1 Introduction

Petri Nets provide a powerful formalism for modelling and implementing distributed concurrent systems. The Paose methodology (Petri net based Agent- and Object-Oriented Software Engineering [1]) uses software agents and Multi Agent Systems to develop distributed systems with reference nets [6]. Applying our approach showed problems when developing larger systems. The need for additional abstraction and structuring was identified in this context. First proposal have been made on this topic in [8,7]. Here we now show the final result as a consolidation of the former attempts.

When developing larger Multi Agent Systems (MAS) the question needs to be answered what kind of functionality we have to assign to an agent. To ease this, here we propose to use two types of agents to implement in the system. Doing so should give some hints how they should interact in order to achieve the intended goal of the system. We have experienced that having a type of agent eases the modeling of a system.

One way to distinguish elements of an MAS is between active and passive entities. Usually agents are considered active components. In [11] artifacts are introduced as another type of element in MAS, that agents can use and interact with. As similar approach based on Petri nets but not covering enriched social concepts has been proposed in [13] with the term of units.

The tools and materials approach (T&M) [14] for object-oriented software engineering distinguishes tools and materials as different artifact types which users can interact with in a software system. HeRA tries to adopt and extend ideas from the former three proposals for the creation of distributed user-centered agent systems. In the HeRA-system a user can access functionality by using

helper agents, that can act like tools to work on resource agents, who in turn can act like materials.

In the following, section 2 will give an overview over the PAOSE methodology and the MULAN/CAPA MAS that is used in the development of the HERA-system. Section 3 will then go on to describe the different concepts used in the HERA-system, which types of agents it consists of and how they interact with each other.

Section 4 explains these concepts by means of a simple example application built on HERA. Finally in section 5 we draw a conclusion and give an outlook on future work.

2 Developing applications with the PAOSE methodology

The PAOSE methodology of software engineering uses reference nets [6] for the modelling and implementation of software. Using interacting nets, the MULAN MAS [12] has been used for agent-oriented software development for years now.

In MULAN an MAS consists of agent platforms, which are connected with a communication infrastructure. Agents reside on platforms. A platforms manages the creation and deletion of agents and the communication between agents on a platform as well as between platforms. The behaviour of agents is determined by protocols and decision components within the agents. All these components are implemented as reference nets, interacting over synchronous channels.

A number of different modelling techniques are used in this approach, describing the system from a number of different yet linked perspectives [1]. The main focus in developing MAS applications with this approach is describing the different agents and agent roles within the system, the internal processes within the agents as well as the interactions between them, and the ontology used for representing concepts in the system. Interactions, internal processes and ontologies are implemented directly in petri nets (features structure nets for the ontology, reference nets for everything else).

The HERA-system now aims at further establishing an application-oriented perspective. By focussing on domain objects and supporting the users of the system, we hope to improve the usability and overall quality of software systems.

3 Helper and Resource Agents

In order to use an MAS application, users can interact with other agents within the MAS. This is often accomplished by means of a user agent, which represents the user within the system and usually provides the user with some kind of user interface that translates his input into agent activity. If new functionality is added to the system, the user agent, too, needs to be augmented, so that the user can access it.

In dynamic distributed systems, where new functionality is added frequently, this can be quite challenging. And if a typical user only needs a fraction of

the functionality, it is advisable to provide some kind of extension or plug-in mechanism that allows easy integration of new functionality on demand [2,3,4]. In HeRA we use helper agents, that plug into the user agent to provide new functionality where needed. The basic concepts of the HeRA system have been introduced in [8,7].

3.1 Overview

Figure 1 shows the different types of agents and platforms in the HeRA-system. A user connects to the system using his two-part user agent. The GUI is used for user interaction, while the agent part represents the user within the MAS. The user agent is connected to a number of helper agents that provide functionality and resource agents that represent resources and documents the user can work with using his helper agents.

The configuration of helper and resource agents a user has on his agent platform represents his personal workplace. He can use helpers to communicate and exchange resources with other users within his greater work environment. Agents can also meet and interact with each other on collaboration platforms, which represent e.g. location, places or groups within the system. Service platforms host agents that provide services, like the helper factory which is used to create new helper agents. Other examples could be a workflow management system or a security subsystem.

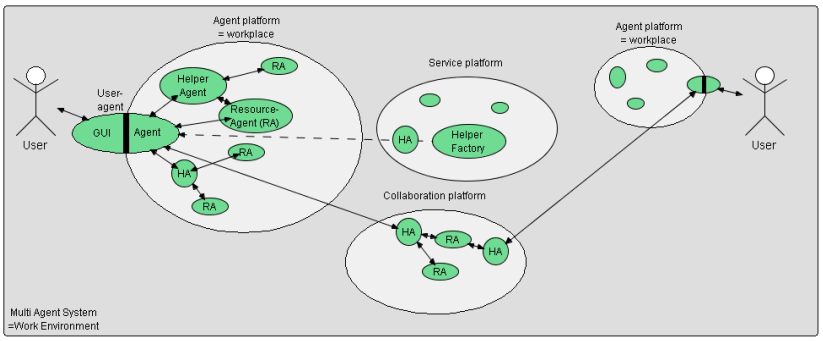


Figure 1. Agents and Platforms in HeRA

3.2 Useragent

The user agent consists of two parts, the GUI and the agent. The agent part is a Mulan agent that knows all the interactions necessary for handling helper agents and uses an RMI connection to communicate with the GUI part. The

GUI can therefore be located on the same or on a different computer than the agent part.

The GUI displays to the user a list of available helper agent types in the MAS, which he can choose to request from the helper factory (see below). If a new helper agent is registered with the user agent, it sends a description of his own user interface, which can then be integrated into the user interface of the user agent. That way the generic user interface of the user agent can be enhanced in any way needed to provide the functionality of the helper agent.

3.3 Helper Agents

In the T&M approach, users use tools to interact with materials and their environment. HERA turns these tools into agents, who can actively support a user in their work. This is reflected by the name helper agent, which emphasizes the more active role of the agent.

Helper agents are responsible for providing any kind of functionality to the user. They can provide a service all by themselves, encapsulate a legacy application, interact with other helper agents or display and manipulate resource agents.

3.4 Helper Factory

The helper factory is an agent used for creating new helper agents. It holds a list of helper agent types which it offers to user agents to choose from. On request it gathers all the information needed for creating the new helper agent and orders the agent management system to create it. The new agent can be customized for the user, for example only including functionality that the user can access according to his user permissions.

3.5 Resource Agents

Resource agents represent materials and resources in the work environment of a user. Instead of adding new concepts to the agent platform, resources are modelled as active components as well. A resource agent acts pretty much like an object, which can be handled by a helper agent, but it can also enforce its own rules. For example a material can decide which helper agents can access it or decide conflicts in concurrent access.

Helper and resource agents need to understand the same usage patterns, it is not possible to use a hammer to fasten a screw for example. These usage aspects represent a m:n mapping between different helper and resource types. As long as a helper understands the usage protocols of a resource, it can use that resource. Here the work on service manuals [5] can be used to improve the mutual dependencies with respect to the behavior.

4 Example: a Whiteboard Application

To illustrate these concepts, we provide an example in the form of a simple whiteboard application. A whiteboard is a common medium used by multiple people to communicate about ideas etc. While multiple people can read the content of the whiteboard, only one person at a time can write on it.

The whiteboard itself is modelled as a resource agent. It manages the content of the board and accepts requests for changes to this content. Helper agents can register with the whiteboard material agent to receive updates on the content whenever it changes.

The whiteboard helper allows a user to use the whiteboard application. The user requests a new whiteboard helper from the helper factory, which creates the agent for him. After registration the user agent loads the GUI extension for the helper (see figure 2) and connects it to the helper agent.

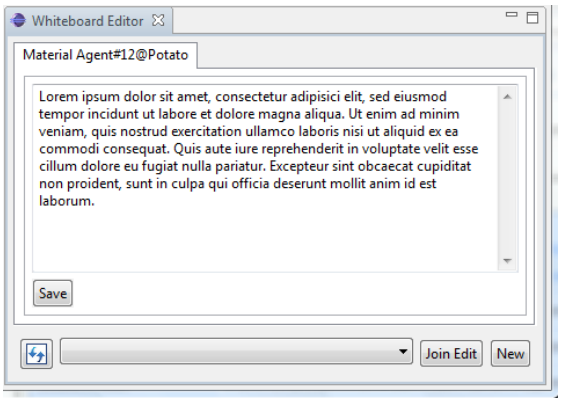


Figure 2. Whiteboard application

Using the agent by means of the GUI extension, the user can create new whiteboards or access a list of whiteboards already existing in the system and subscribe to them. He can then edit the content of the resource agent, which results in updates to all connected helpers. In some way this supports an event driven perspective.

The whiteboard example illustrates one of the possible arrangements for the collaboration platform in Figure 1. In general resource agents should be placed on such platforms if they do not belong to a single agent. Implicitly these platforms become collaboration platforms if several agents use the resource.

5 Conclusion and Outlook

Petri net based Multi Agent Systems can be used to structure net-based application development. In this paper we presented further structural elements within

MAS by introducing the concepts of helper and resource agents. These concepts provide the possibility to design applications that are more focussed on providing functionality to individual users collaborating within a distributed system. Additionally work objects can be modelled explicitly as first-order objects within the system.

In [10] it has been shown how these concepts can be used to leverage agent-based workflow management systems [9]. Future work focusses on combining these aspects further into an application development platform for complex distributed systems.

What has not been discussed here is the possibility of "feeding" agents with roles, goals, obligations etc. This allows for declarative style of programming which is nicely integrated due to the nature of agents. So also the use of social models that are currently discussed for the organization of agents can be applied to improve the overall architecture. Again this is inherently covered by MAS in general and hence can also be used in our helper and user agents.

References

1. Lawrence Cabac. *Modeling Petri Net-Based Multi-Agent Applications.* Dissertation, April 2010. http://www.sub.uni-hamburg.de/opus/volltexte/2010/4666/.
2. Lawrence Cabac, Michael Duvigneau, Daniel Moldt, and Heiko Rölke. Multi-agent concepts as basis for dynamic plug-in software architectures. In *AAMAS 2005*, pages 1157–1158, 2005.
3. Lawrence Cabac, Michael Duvigneau, Daniel Moldt, and Benjamin Schleinzer. Plugin-agents as conceptual basis for flexible software structures. In *Multi-Agent Systems and Applications V. CEEMAS'07, Leipzig.*, volume 4696 of *LNCS*, pages 340–342. Springer, 2007.
4. Michael Duvigneau. *Konzeptionelle Modellierung von Plugin-Systemen mit Petrinetzen.* Dissertation, October 2009.
5. Kathrin Kaschner, Peter Massuthe, and Karsten Wolf. Symbolische Repräsentation von Bedienungsanleitungen für Services. In *AWPN'06*, September 2006.
6. Olaf Kummer. *Referenznetze.* Logos Verlag, Berlin, 2002.
7. Kolja Lehmann, Lawrence Cabac, Daniel Moldt, and Heiko Rölke. Towards a distributed tool platform based on mobile agents. In *MATES'05*, volume 3550 of *LNAI.* Springer, September 2005.
8. Kolja Lehmann and Vanessa Markwardt. Proposal of an agent-based system for distributed software development. In *MOCA 2004*, Aarhus, Denmark, 2004.
9. Kolja Markwardt, Lawrence Cabac, and Christine Reese. A process-oriented tool-platform for distributed development. In *MSVVEIS 2009*, pages 44–52, 2009.
10. Kolja Markwardt, Daniel Moldt, and Thomas Wagner. Net agents for activity handling in a wfms. In *AWPN 2009, Karlsruhe, Germany*, 2009.
11. A. Omicini, A. Ricci, and M. Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, 2008.
12. Heiko Rölke. *Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen*, volume 2 of *Agent Technology – Theory and Applications.* Logos Verlag, Berlin, 2004.
13. Volker Tell and Daniel Moldt. Ein Petrinetzsystem zur Modellierung selbstmodifizierender Petrinetze. pages 36–41, 2005.
14. Heinz Züllighoven. *Object-Oriented Construction Handbook.* dpunkt Verlag, 2004.

PyTri, a Visual Agent Programming Language

Jochen Simon and Daniel Moldt

University of Hamburg, Department of Informatics
http://www.informatik.uni-hamburg.de/TGI/

Abstract. PyTri is a Python based visual agent programming language which has been designed top-down to utilize the possibilities of graphical representation of control flow by amending the concepts of Petri nets. Its main inspiration, Mulan, which is based on Java nets, originated bottom-up from a powerful formalism, which allows modeling and programming multi-agent systems. The here presented PyTri vision uses multiple types of transitions and places, with a specialized Python-based inscription language, in order to offer a rich semantics that allows expressive and compact representation of executable code. GUI widgets can be directly embedded into the nets and can infuse them with tokens upon user interaction.
PyTri aims to support coarsening mechanisms, instead of the net within net paradigm, and includes a special place class that allows representation of independent control flow streams. It strives to enable modularization of complex multi-agent applications as one huge flat structure, not requiring them to be separated into discrete layers. Integral to PyTri is a future tool for visualization and manipulation which is tailored to the needs of programmers using the language.

Keywords: PyTri, Renew, graphical visual programming language, agents, Petri nets, formalism, reactive programming, Python

1 Introduction

Visual Languages offer more modalities[5] for representing and incorporating the semantics of a program, compared to traditional textual languages. They directly express the control flow of the processes inside an application, unlike line-based text files with procedural jumps. In recent functional languages a plethora of different ways to compose functions emerged, resulting in arrows which are more general than monads. However, understanding the underlying mechanisms and being able to utilize them, requires at least basic proficiency in category theory.

Petri nets offer a very intuitive way of representing arbitrary computation processes by decomposing them into atomic actions and intermediate chunks of data, incorporated by transitions and places. By embedding external triggers into a net, reactive programming can be done without the hassle of splitting up the code into steps.

The human mind in general is very good at memorizing structures visually, grouping related objects by spatial proximity. There are several types of UML

diagrams which are a popular method to visualize the interrelations of the classes and objects of an application. By modeling an application with a Petri net like net formalism in one single huge structure, the big picture can be shown as a direct map of the internal processes, possibly even in three dimensions. Of course, mechanisms of coarsening have to be used, in addition to modularizing the code into organ-like fragments, or none would be able to cope with the complexity.

In [10] several existing visual languages were looked at, some general criteria were investigated and PyTri, a visual agent programming language based on Python was devised. Amongst the investigated languages were YAWL[9], AgentSheets[8], AUML[2], LabVIEW[6] and Quartz Composer[1]. PyTri is heavily inspired by the Java net formalism of Renew, which is based on reference nets, whose theoretical foundations were laid down in [7]. Java nets allow modeling and execution of complex multi-agent based distributed applications[3], through a sophisticated tool set[4]. Java nets have been designed bottom-up, starting with the reference net formalism, and are especially suited for modeling.

In Section 2 the mission statement for PyTri, the aims of its semantics, and the most important properties of its formalism are given. Section 3 introduces the idea of specialized types of places and transitions, a mixed class of elements, PyTri's focus on coarsening and refinement, and tool support. Section 4 discusses the results of the design of PyTri and future work.

2 Goals of PyTri

PyTri aims to be an expressive and compact language, condensing common programming tasks to a clean and descriptive representation. PyTri has been designed from the start with the goal of making the creation of multi-agent based applications as comfortable as possible. Essential to PyTri is an extensive graphical editor displaying the net code, designed for developing, debugging and manipulating. The concepts of PyTri have been designed top-down with a focus on programming, not modeling. PyTri aims to employ a rather complex semantics, and is targeted at experienced programmers willing to learn its large vocabulary. It still tries to be intuitive, without dumbing down, in order to be accessible to novices, given a set of good tutorials, gradually introducing the concepts.

The Java net formalism has some assembler like aspects to it, because one has to perform a lot of bookkeeping by hand to perform basic tasks. The formalism of PyTri attempts to take graphical programming to a higher level of abstraction, retaining a semantics that is very similar to that of Petri nets, but differing in several important ways, still including elements that act very much like traditional places and transitions. Rather than having one kind of place and transition that is used for everything, the formalism is based on specialized elements that are the basic primitives of the language and can be compared to statements in textual languages, like `if`, `while`, `try`, `def` or `return`.

In order to allow compact branching of control flow, the PyTri formalism deviates from the Petri net semantics by allowing transitions to dynamically decide to suppress the donation of tokens to specific target places. This means

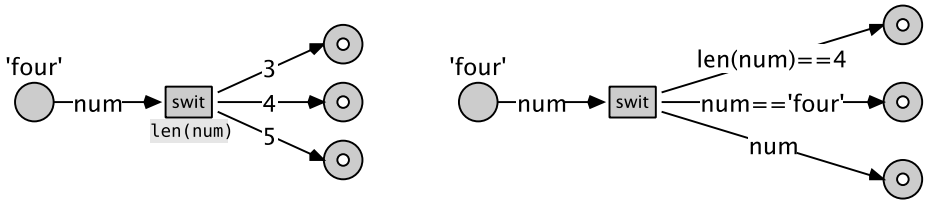


Fig. 1. Two variants of the switch transition used to decide control flow

that the actual control flow is no longer apparent from the arc connections alone, but dependent on the inscriptions of the transitions. Such conditional branching would suffice, and may be the best solution in some cases, but in other cases having a specialized transition type, with a different inscription semantics, can be useful syntactic sugar and convey the intention of the code in a better way. An example is the Switch transition shown in Figure 1, which can be used to evaluate an expression and donate a token only through the arc whose inscription matches, or can evaluate a set of arcs inscribed with boolean expressions and independently donate tokens through all those that are true.

PyTri's inscription language is based on Python, which was chosen because it is a very compact and powerful high-level language and offers a great standard library. It is attempted to allow almost arbitrary Python code in the inscriptions, while amending the inscription language with the required new concepts, pythonically. It uses pythonic duck typing, where Java nets guarantee that only certain types of objects can traverse an arc. A lot of the expressiveness of Java nets comes from unification, which allows fully dynamic connections between elements. PyTri aims for a simpler and easier to understand semantics, which introduces other mechanisms to cope with the loss of the prolog-like power.

3 Complex Elements and Tool Support

Place Types. Just like in Petri nets and Java nets, places are passive and do not initiate actions by themselves, however, they are preconditions to the transitions that accept tokens from them. Instead of having one general kind of place which is used universally to store data, PyTri offers several types. Usually, places are either used to store a single token or to store a collection of tokens, rarely both.

When storing single tokens, places are usually used either for signifying and determining the control flow, or as a single variable holding intermediate or persistent data objects, which leads to the place types Flow and Variable.

When storing a collection of tokens, one usually wants it to behave like one common abstract data type, depending on the context. By incorporating the required behavior in a place type, Queue or Stack for example, the programmer can be spared of the otherwise necessary manual bookkeeping. Also, special place types can exist simply to provide helpful eye candy to the developer, or simplify debugging of agents by easing the assignment of the states an agent can be in.

Transition Types. Many abstract mechanisms integral to programming are not atomic, but are composed of multiple steps. By having basic elements that execute multiple atomic phases when needed, while storing internal data, the language becomes more compact in its expressiveness. Those elements of this kind, which in essence are transition-bordered components, can be considered similar enough in concept to transitions to justify raising them to their level. Through this the concept of transitions is extended to a set of basic graphical statements building the core language. Since the individual phases are still atomic, complex multi-phase transitions should be decomposable to nets of places and atomic transitions. This decomposition however can also be virtual, since the behavior may be implemented through pure Python objects with a certain interface.

Some transition types require special interactions with certain types of places in order to achieve their purpose. Some transitions work like components built out of elements connected to other transitions via virtual arcs. PyTri has been designed from the start with the aim of creating a language tailored to the creation of multi-agent systems. Amongst the basic transitions are elements that allow sending messages to other agents, receiving messages, or a combination.

Gadgets. In order to directly embed GUI components into the nets, a new general type of element was devised. Gadgets are the representation for mixed-border components, which can perform proactive actions, but also expose their internal data to connected transitions, like a place. The internal virtual elements of a gadget are connected to the elements around it, based on the aliases of the connecting arcs. Gadgets have a consistent semantics that enables them to directly interact with the surrounding net by donating tokens, acting as a precondition, and receiving tokens. For GUI components, the behavior can be encapsulated in pure Python proxy objects that perform all required bookkeeping transparently. In Figure 2, a dialog is visible and tangible to a local user, as long as a token is inside the leftmost place. When one of the two buttons is pressed, the dialog hides, consumes the first token and donates a new token through the matching arc. As a result, either the user name is copied into the net, or the agent terminates.

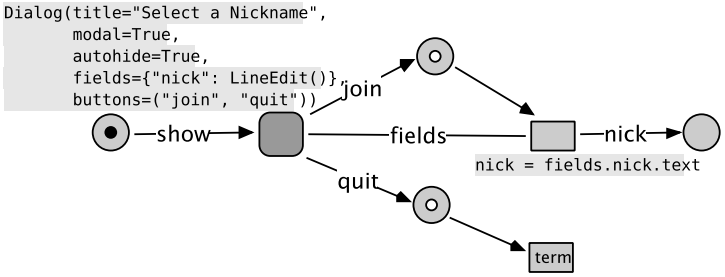


Fig. 2. A GUI gadget directly embedded in a net, interacting with a local user

Coarsening. The main mechanism of modularization in JAVA NETS is the nets within nets approach, since places can contain net objects. Contained nets can interact, through their own internal transitions, with outer transitions connected to their place, using synchronous channels. However, this approach introduces a sharp partitionment of the code into layers, resulting in a break of continuity.

PyTri favors a flat structure where all elements are on one universal layer, and other mechanisms for partitioning are used instead. Some control flows need to be executed in multiple independent instantiations, for example protocols that control interactions between agents. Instead of instantiating protocol nets as is done in MULAN, such code can be modeled through a new class of place. Parallel places transparently manage all bookkeeping necessary to flatten the behavior into a single structure, where tokens move in independent streams, separating the structure into virtual instances. Such places can also give an informational summary of all independent streams to the developer, during execution. This mechanism can be enhanced to allow piecing together net structures that look like AUML diagrams.

Coarsening and refinement of nets is a very powerful mechanism to make big structures controllable. By allowing to collapse and expand collections of elements forming a component, one can choose the optimal level of detail of the representation of the code. This can be used to modularize the code and the resulting segments can be coarsened to look just like normal elements. An interface for such a segment can be defined by specifying which of the contained elements are accessible through certain aliases. Depending on whether only places, transitions, or both are accessible, segments are represented as such an element, in the mixed case as a gadget.

Tool Support. An integral part of PyTri is having tool support for the language, namely a Monipulator that allows creation, manipulation, monitoring and debugging of net code. In order to allow developers to separate big nets into manageable chunks, they can be partitioned into fragments, which are joined into a flat whole net, maintaining the original semantics.

The preliminary maxim of PyTri, which arose due to the global interpreter lock of Python, is that every agent is its own process. There is no global simulator process that contains, manages and controls agents. Memory is not shared between agents, which have to rely on interprocess-interaction, facilitated by a process present on every machine of a distributed environment.

4 Conclusions

Programing complex systems remains a challenge. This paper provides a vision in form of a new graphical programming language to address this problem. PyTri overcomes some drawbacks of the successful extension of Java in the form of reference nets with agent concepts from MULAN. It does so by amending a different language, namely Python, and adding several new or adapted constructs and concepts to the language with respect to the graphical underpinning. While it is

strongly influenced by Petri nets, its semantics is adapted to the requirements of programming and explores a lot of different possible programming mechanisms, in order to investigate the realm of possibility. This allows for powerful expression of behavior through new constructs and concepts. PyTri's concepts, the prototypical implementation, written in Stackless Python, using PyQt, and the whole background of the language has been intensively discussed in [10]. Currently the main features are: Python as the inscription language, a powerful semantics with concepts and constructs to express important agent features, and an available prototype for some of the central concepts.

Our current direction of research is now to provide a proper Petri net semantics for the current concepts and constructs. Concepts and experiences from reference nets, Renew and Mulan/Capa will give further directions of development beside the direct evaluation of the current PyTri tool set. The main goal is to keep and extend PyTri as a powerful modeling and programming language while trying to regain the analytical power of Petri nets, which is currently lost. In order to truly seize the power of a language which directly expresses concurrency, a different implementation base for the execution core and the inscription language has to be considered, possibly based on PyPy or C++. Besides PyTri as a language, the tools around PyTri will also be subject to several different investigations. In the future, a series of prototypes, no longer based on Stackless, which gradually implement more and more concepts of the language, will be created. Here concrete, at the beginning rather artificial applications will further help to find the right modeling and programming concepts.

References

1. Apple Developer Connection – Quartz Composer. http://developer.apple.com/graphicsimaging/quartzcomposer/ (2009)
2. Cabac, L.: Modeling Agent Interaction Protocols with AUML Diagrams and Petri Nets. Diploma thesis, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg (Dec 2003)
3. Cabac, L.: Modeling Petri Net-Based Multi-Agent Applications. Logos Verlag, Berlin (2010)
4. Cabac, L., Dörges, T., Rölke, H.: A monitoring toolset for Petri net-based agent-oriented software engineering. In: Valk, R., van Hee, K.M. (eds.) 29th Petri net conference, Xi'an, China. LNCS, vol. 5062, pp. 399–408. Springer-Verlag (Jun 2008)
5. Jensen, K.: Coloured Petri Nets : Basic Concepts, Analysis Methods and Practical Use. Volume 1 (Monographs in TCS. An EATCS Series). Springer-Verlag (1992)
6. Johnson, G., Jennings, R.: LabVIEW graphical programming. McGraw-Hill Professional (2006)
7. Kummer, O.: Referenznetze. Logos Verlag, Berlin (2002)
8. Repenning, A., Ioannidou, A.: Agent-based end-user development. Commun. ACM 47(9), 43–46 (2004)
9. Russell: Workflow control-flow patterns: A revised view. http://www.workflowpatterns.com/documentation/documents/BPM-06-22.pdf (2006)
10. Simon, J.: Design of a Python Based Visual Agent Programming Language and its Prototypical Implementation. Bachelor thesis, University of Hamburg, Department of Informatics (2009)

Optimised Calculation of Symmetries for State Space Reduction

Harro Wimmel

Universität Rostock, Institut für Informatik

Abstract. Since the state space explosion is a common problem when analysing Petri nets several ways to deal with this problem leading to a smaller – reduced – state space have been invented. One of them is finding symmetries, an equivalence relation on places and transitions of a Petri net, and only evaluating one object from each equivalence class. All other objects in the same class are then known to yield the same information. Finding symmetries by brute force is known to be expensive, it is even unclear if it can be done in polynomial time due to the inclusion of the graph isomorphism problem. While a first few symmetries need to be found by brute force, later ones might also be generated. We show how to generate new symmetries from known ones efficiently, how to tell if the brute force algorithm enters a branch not containing symmetries, and how to reduce the symmetries themselves to move towards orthogonality.

Keywords: Petri Net, State Space, Symmetry.

1 Introduction

Tools for state space exploration like LoLA [Wol10] use many techniques to reduce the state space before exploring it to answer a question. This is due to the well-known problem of state space explosion, where the size of the state space can grow exponentially or worse with the size of the system to be analysed. If a system contains components that are indistinguishable from each other (by the structure of the system and the question asked about the system) it is obviously sufficient to analyse one such component only. The result can just be mapped to an equivalent component then. A mapping maintaining the relation between components is called a symmetry.

With Petri nets as systems, components are just the places and the transitions. A symmetry therefore maps each place to some place (possibly itself) and likewise for transitions, keeping the edges, i.e. a mapped pair of place and transition has an edge between them if and only if the original pair has that edge. A symmetry is thus not more than a structure-preserving permutation on the places and transitions of a net.

Unluckily, the number of symmetries (the size of the *automorphism group*) of a Petri net (or any system) can be much larger than the system itself. It is necessary to find a small set of symmetries (called *generator set*) from which all

other symmetries may be derived. If the components are ordered and numbered (say from 1 to n), a generator set consists of n levels and each level i contains one symmetry for each possible image j of the component i with $j \geq i$. Components with a number less than i are mapped to themselves. Take the Petri net from Fig. 1 as an example.

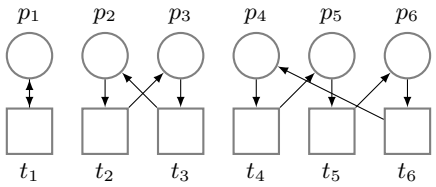


Fig. 1. An example Petri net N

If a place p_i is mapped to p_j then the transition t_i is mapped to t_j due to the structure of the net (each place has only one successor transition). If we thus use just the number of a place/transition instead of its name, we may obtain the following generators: the identity for level 1, (2 3) and the identity for level 2, (4 5 6), (4 6 5) and the identity for level 3. Only components mapped to different components are shown explicitly in this notation, and each one is mapped to the next inside the parentheses (the last to the first). For level 2 there are two replacement candidates for (2 3): (2 3)(4 5 6) and (2 3)(4 6 5). Any one of these three is sufficient. It can be shown (see e.g. [Sch02]) that any symmetry σ can be written as the composition of some generators g_i, one from each level i, by $\sigma = g_1 \circ g_2 \circ \ldots \circ g_n$.

While LoLA seems to implement an algorithm (called *Refine*/Define* in [Sch02]) that runs in polynomial time in practical cases, this cannot be guaranteed due to the inclusion of the graph isomorphism problem for which membership in P is unknown. Therefore, it is important to rely on such an algorithm (that builds generators from scratch) as seldom as possible and use already known generators instead to derive new ones when possible. To a certain extent, LoLA already does this by composing generators with themselves, building powers, and checking whether these powers can fill the gaps where generators are still missing. In the following, we show how this can be improved.

2 Basic Definitions

We assume Petri nets, formally a tuple (P, T, F) with F: $(P \times T) \cup (T \times P) \to \mathbb{N}$, to be known to the reader. We also expect some knowledge about linear algebra, especially the definition of a group, and start now by defining symmetries.

Definition 1 (Symmetry). *Given a net (P, T, F), a* symmetry σ *is a map σ: $P \cup T \to P \cup T$ with $\sigma(P) = P$, $\sigma(T) = T$, and $F(\sigma(x), \sigma(y)) = F(x, y)$ for all $x, y \in P \cup T$.*

We might also be interested in a (initial or final) marking m, in which case $m(\sigma(p)) = m(p)$ must hold additionally for all $p \in P$. A symmetry σ is written in the style $(a_{11} \dots a_{1j_1}) \dots (a_{n1} \dots a_{nj_n})$ where $\sigma(a_{im}) = a_{i,(m \bmod j_i)+1}$.

In the following we assume a fixed Petri net $N = (P, T, F)$, a fixed bijection b: $P \cup T \to \{i \in \mathbb{N} \mid 1 \leq i \leq |P \cup T|\}$ and identify the places and transitions of N with their images under b.

Definition 2 (Generator, level, orbit). *A symmetry g* for level *$lev(g) = i$ ($1 \leq i \leq |P \cup T|$) is a symmetry with $g(k) = k$ for all $k < i$. The number $g(i)$ is called the* orbit *of g. Each level i also has* orbits*, numbered from i to $|P \cup T|$. An orbit k of level i is* consistent *if there is a symmetry g for level $lev(g) = i$ with an orbit $g(i) = k$, otherwise the orbit is called* inconsistent *or* empty*. Define G_i to be the* group of all symmetries for level *i (with $G_i \supseteq G_{i+1}$). On the other hand, a* generator set *G consists of one symmetry, called* generator*, g_{ij} for each level i and each consistent orbit j on that level.*

Corollary 1 ([Sch02]). *Let G be a generator set. Each symmetry g for level i can be expressed as a consecutive composition of one generator $g_j \in G$ from each level $j \geq i$.*

Corollary 2 ([Sch02]). *There is an algorithm* Refine*/Define *taking a level i and an orbit k as input and producing a symmetry g for level i with $g(i) = k$ if such a symmetry exists. Otherwise, the algorithm terminates with the result "inconsistent".*

3 Inheriting Inconsistency

Inconsistencies are obviously the worst result that can be obtained from the *Refine*/Define* algorithm. They waste time and do not even produce a generator. Once we know of an inconsistent orbit for some level, we may use this information to find other inconsistent orbits without the *Refine*/Define* algorithm. An equivalence relation for places/transitions can be helpful here.

Definition 3 (Equivalence of components). *Let G be a generator set and i some level. For $x, y \in P \cup T$ we define an equivalence relation $x \equiv_i y \iff \exists g \in G_i$: $g(x) = y$.*

Note that $\equiv_i$ being an equivalence relation follows from the fact that G_i is a group with identity, an inverse, and composition as group operation.

Lemma 1 (Inheritance). *For level i, let k be an inconsistent orbit and n be a consistent orbit. Then, $n \not\equiv_i k$.*

Proof. Assume $n \equiv_i k$, then there is $g \in G_i$ with $g(n) = k$. Let $g' \in G$ be the generator for level i with $g'(i) = n$. Then, $g(g'(i)) = k$ and $g' \circ g \in G_i$ has the orbit k. A contradiction, as no symmetry g'' for level i with $g''(i) = k$ exists.

Looking from the other side, this means every orbit m with $m \equiv_i k$ must be inconsistent. It is therefore unnecessary to call the *Refine*/Define* algorithm for orbits equivalent to k. In our example from the introduction, all orbits $k > 1$ on the first level are inconsistent, since p_1 can only be mapped to itself by any symmetry. If we know that the orbits 2 and 4 are inconsistent, we conclude from the generators (2 3) and (4 5 6) (from levels 2 and 3) that $2 \equiv_1 3$ and $4 \equiv_1 5 \equiv_1 6$. We save three calls to *Refine*/Define*.

4 Building Products

LoLA so far takes a newly acquired generator g for level i and calculates the powers $g^2 = g \circ g$, g^3, g^4, and so on, until $g^n(i) = i$ holds. If one of the powers has an orbit for which no symmetry has been found so far, the power is saved as the new gnerator for that orbit. Further powers do not yield anything new as g^{n+1} has the same orbit as $g^1 = g$.

Instead, we propose building compositions of any new generator with any generator found so far until no new generators are derived anymore. This looks like a losing approach at first as there are by a linear factor more such products than powers. But note that in the powers approach $O(n)$ (with $n = |P \cup T|$) powers must be calculated until $g^n(i) = i$ holds and each composition done also needs $O(n)$ (the size of the map). The powers approach therefore looks quadratic.

Lemma 2 (Complexity of product testing). *Let $g \in G_i$ and $g' \in G_j$ with $j \geq i$. A test if the composition $g \circ g'$ leads to an orbit for which no generator has been found so far can be done in $O(1)$.*

Proof. We check if there is a generator for level i with orbit $g'(g(i))$. This takes $O(1)$ time. If there is none, $g \circ g'$ will be the new generator for level i and orbit $g'(g(i))$.

Note that this simple test is useless for the powers. We would test and then calculate the composition anyway, independently of the test's result, when we need the next, higher power.

What is important here is that a newly found generator for level i is composed only with generators of a higher level. To guarantee this, the levels have to be filled with generators from highest to lowest. This is the way it is done in LoLA anyway: LoLA uses a recursion from easier to harder problems, and higher levels represent the easier problems (as more elements are mapped to themselves).

If we try to calculate a complexity for our approach and (falsely) assume that the size of a generator set G is roughly equal to $|P \cup T|$ we obtain $O(n^2)$ tests (for pairs of generators in G) with complexity $O(1)$ each and $O(n)$ compositions with complexity $O(n)$ each. This would suggest a quadratic complexity just like for the powers' calculation. There are examples where the size of the generator set is much higher as well as those where it is much lower than $|P \cup T|$, so the real complexity comparison is much more difficult.

In general, the product approach will produce more new generators in a single call than the powers approach, but this depends on the structure of the automorphism group. If the *order* of generators (the lowest power yielding the identity) is lower than the number of orbits on some level, it is impossible to fill all orbits in a single call of the powers approach. Since products are iterated, from them the whole subgroup spanned by all known generators could be computed. This can mean an exponential gain compared to the powers, e.g. in groups with p^n elements (p prime) where g^p is the identity for all symmetries g. By powers at most $p-1$ new generators can be built from each call to *Refine*/Define*, while with products the group spanned from the known generators increases by a factor of p for each call.

5 Minimizing the Carrier

Our last optimisation does not deal with the finding of generators but with the size of their representation. While a smaller size might reduce execution times there may be other benefits. Let two generators g, g' be *orthogonal* if $g(i) \neq i \implies g'(i) = i$, then g and g' can be composed without effort. The set $\{i \mid g(i) \neq i\}$ is called the *carrier* of g. Conclusions drawn about the carrier of g cannot be influenced by g' and are thus valid for $g \circ g'$. While orthogonality is unreachable in general, we may still try to minimize the carrier of any generator g. Candidates that may have a smaller carrier are easy to find:

Corollary 3. *If g is a generator for level i and orbit k and $n > 0$ is the smallest integer with $g^n(i) = i$, then, from all powers of g, exactly the g^{jn+1} (for $j \in \mathbb{N}$) are generators for level i and orbit k.*

Take e.g. the cycle representation of one of the generators for the Petri net from Fig. 1: $g = (2\ 3)(4\ 5\ 6)$. If we take g as generator for level 2 and orbit 3 then $(2\ 3)$ is the *orbit cycle* of g (it contains level and orbit). Its length is $n = 2$, i.e. $g^2(2) = 2$. Thus, reduction candidates are $g^{2+1} = (2\ 3)$, $g^{4+1} = (2\ 3)(4\ 6\ 5)$, $g^{6+1} = g^1$ and so on. Since $2 + 1 = 3$ is divisible by the length 3 of the second cycle $(4\ 5\ 6)$, this cycle is eliminated in g^3 ($g^3(i) = i$ for $i = 4, 5, 6$ and identities are not shown in cycle representation). In general, the following holds.

Lemma 3. *Let g be a generator with an orbit cycle $o = (o_0\ \ldots o_{i-1})$ and another cycle $c = (c_0, \ldots, c_{m-1})$. Let k be the greatest divisor of m such that i and k have a greatest common divisor (gcd) of one. Then, in g^{k+qm} (for $q \in \mathbb{N}$) the cycle c is replaced by k cycles of length m/k and there is no power of g having the same orbit cycle and shorter replacement cycles for c.*

Proof. Just note that $g^k(c_{(n+\ell k) \bmod m}) = c_{(n+(\ell+1)k) \bmod m}$ and for $\ell = \frac{m}{k} - 1$ for the first time $g^k(c_{(n+\ell k) \bmod m}) = g^k(c_{(n+m-k) \bmod m}) = c_{n+m \bmod m} = c_n$ holds. If we choose k a greater divisor of m, there can be no j such that k divides $ji+1$, since the *gcd* of k and i also divides ji (and not $ji+1$). Since a divisor of k also divides $k + qm$, $k + qm = ji + 1$ is impossible, i.e. if $gcd(i, k) > 1$, g^{k+qm} will not preserve the orbit cycle.

Thus, for $k + qm = ji + 1$ the cycle c is reduced as far as possible and at the same time the orbit cycle stays intact. It is unnecessary though to solve this equation as the proof of lemma 3 already tells us how the cycle c will be modified.

6 Experimental Results

The approach of building powers of generators can already be optimal, as it happens e.g. with the dining philosophers and reader/writer systems. Experiments for these examples show a worst case slow down of $1 - 2\%$ in execution times from the powers approach to our new optimisations. This supports our earlier assumption that building products is not (much) slower than building powers.

A good example to show the difference between old and new approach is the hypercube ECHO [Rei98], a grid-like network with d dimensions and n communicating agents per dimension. The value of detecting inconsistencies can be shown with the nets N_i, where $N_3 = N$ from Fig. 1, and each higher index adds a new ring with i places and transitions each. Execution times for computing the symmetries are shown in Table 1.

ECHO	#symm	#gen	Old	New
3/3	48	10	0.2s	<0.1s
3/5	48	10	4.6s	2.9s
3/7	48	10	29s	16s
4/3	384	21	3.5s	1.6s
4/5	384	21	244s	96s
5/3	3840	41	82s	23s

N_i	#gen	Old	New
N_{10}	45	<0.1s	<0.1s
N_{15}	105	0.4s	0.2s
N_{20}	190	2.4s	1.3s
N_{25}	300	8.2s	4.4s
N_{30}	435	25s	13s
N_{35}	595	66s	34s

Table 1. Results for ECHO d/n with numbers of symmetries and computed generators as well as execution times for old and new approach and nets N_i consisting of i cycles of lengths from 1 to i

7 Conclusion

Theoretical observations and experimental results have shown that finding and using symmetries can be optimised beyond the current state of affairs represented by LoLA. Using products and an equivalence to detect inconsistencies leads to a clear speed up in not already optimal cases.

References

[Sch02] K. Schmidt: *Explicit State Space Verification*, Habilitation Thesis, Humboldt Universität zu Berlin, 2002.

[Wol10] K. Wolf: *LoLA – A low level analyzer*, http://www.informatik.uni-rostock.de/~nl/wiki/tools/lola, 2010.

[Rei98] W. Reisig: *Elements of Distributed Algorithms*, Springer Verlag, 1998.

Reachability Analysis via Net Structure

Harro Wimmel, Karsten Wolf

Universität Rostock, Institut für Informatik

Abstract. Exploitation of the structure of a Petri net is widely believed to be an inefficient approach to solving the reachability problem. We show that structure analysis can be combined with integer programming and partial order reduction to obtain a fast reachability solver.

Keywords: Petri net, reachability problem, integer programming, structure analysis.

1 Introduction

The *reachability problem* for Petri nets, i.e. if a final marking can be reached in a given net from the initial marking, is known to be decidable [May84,Kos82,Lam92] but EXPSPACE-hard [Lip76]. Efficient tools exist, but they cannot solve all instances of the problem (at least not in a lifetime). Model checkers, symbolic [CMS06] or with partial order reduction [Wol10], have been used successfully to solve quite large reachability problems.

Here we present an approach that is a mixture of several methods, the main ones being integer programming and structure analysis. The *marking equation*, a linear system of integer equations, is known to be a necessary condition for reachability. When a solution of the marking equation is found, it may represent the parikh vector of a firing sequence solving the reachability problem or not. If it does, the firing sequence needs to be found, otherwise the marking equation can be constrained to discriminate the found solution. The needed constraints are found by analysing the net structure. Consider the example net in Fig. 1.

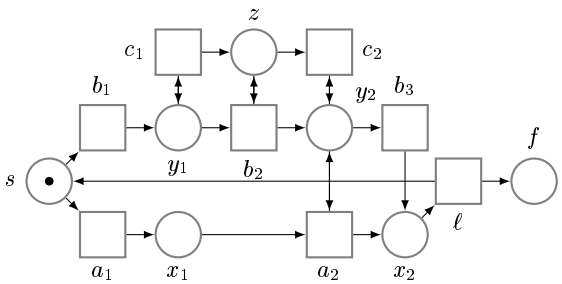


Fig. 1. An example Petri net N with initial marking s and final marking $s + 3f$

The marking equation $m_0 + Cx = m_f$ (where $m_0 = 1s$, $m_f = 1s + 3f$ are initial and final marking and C is the incidence matrix of the net N) can be solved by any integer programming tool. The smallest solution is $3a_1 + 3a_2 + 3\ell$, i.e. each of the three transitions should fire three times (in some unknown order). If we notice that a token is needed on y_2 to fire a_2, but none of the three transitions produces such a token, we might add a constraint "a token should be produced on y_2" to our marking equation to discriminate our first solution. A still viable solution is now $2a_1 + 2a_2 + b_1 + b_2 + b_3 + 3\ell$, as b_2 can produce that token. But now a token on z is missing. If required by a similar constraint, we come to $2a_1 + 2a_2 + b_1 + c_1 + b_2 + c_2 + b_3 + 3\ell$ but the token on y_2 is still not there when needed for a_2. Requiring a higher token production on y_2 will finally lead to the solution $3b_1 + c_1 + 3b_2 + c_2 + 3b_3 + 3\ell$ and now we "only" have to find a correct firing sequence, e.g. $b_1c_1b_2b_3\ell b_1b_2b_3\ell b_1b_2c_2b_3\ell$.

2 Some Basic Definitions

We expect the reader to be familiar with the basic Petri net terminology and some knowledge in linear algebra. All Petri nets here are general ones, i.e. they may have arbitrary multi-arcs including loops. Vectors are sometimes written as finite sums (multisets) over the vector's domain.

Definition 1 (Reachability problem). *A* reachability problem *is the question, when given a tuple (N, m, m') of a Petri net $N = (S, T, F)$ and two markings $m, m' \in \mathbb{N}^S$, whether m' can be reached from m, i.e. if $\sigma \in T^*$ with $m[\sigma\rangle m'$ exists.* The *reachability problem then is the set* $\mathsf{RP} = \{(N, m, m') \mid N = (S, T, F)$ *is a Petri net,* $m, m' \in \mathbb{N}^S$, $\exists \sigma \in T^*$: $m[\sigma\rangle m'\}$. *A reachability problem (N, m, m') is also called an* instance *of* RP*, to which the answer is "yes" if $(N, m, m') \in \mathsf{RP}$ and "no" otherwise.*

The reachability problem is decidable [May84] and making it solvable for as many instances as possible is our goal. It is well-known that a necessary condition for a positive answer to a reachability problem is the feasibility of the marking equation.

Definition 2 (Marking equation). *For a Petri net $N = (S, T, F)$ let $C \in \mathbb{N}^{S \times T}$, defined by $C_{s,t} = F(t, s) - F(s, t)$, be the* incidence matrix *of N. For two markings m and m' the system of linear equations $m + Cx = m'$ is the* marking equation *of N for m and m'. A vector $x \in \mathbb{N}^T$ fulfilling the equation is called a* solution.

For a firing sequence σ the *Parikh vector* $\wp(\sigma)$: $T \to \mathbb{N}$ is defined by $\wp(\sigma)(t) = \#_t(\sigma)$, where $\#_t(\sigma)$ is the number of occurrences of t in σ. If x is a solution of the marking equation $m + Cx = m'$, any firing sequence σ with $\wp(\sigma) = x$ and $m[\sigma\rangle$ positively solves the instance (N, m, m') of the reachability problem. From linear algebra the following is known:

Theorem 1 (Solution space). *For any marking equation $m+Cx = m'$ over a net $N = (S, T, F)$ there are finite sets of base vectors $B \subseteq \mathbb{N}^T$ and period vectors $P \subseteq \mathbb{N}^T$ such that all and only the solutions can be expressed as $b + \sum_i n_i p_i$ with $b \in B$, $p_i \in P$, and $n_i \in \mathbb{N}$.*

In other words, the solution space is a semilinear set over nonnegative integer vectors. Period vectors are nonnegative T-invariants where for a firing sequence σ with $\wp(\sigma)$ to be a T-invariant, $m[\sigma\rangle m$ must hold for all markings m enabling σ. Adding a T-invariant to a solution of the marking equation will produce another solution. Apart from multiples, our example net from Fig. 1 contains only one nonnegative T-invariant: $c_1 + c_2$. The base vectors take the form $i(a_1 + a_2) + (3 - i)(b_1 + b_2 + b_3) + 3\ell$ with $0 \leq i \leq 3$.

3 Traversing the Solution Space

Integer Programming (IP) solvers come in two flavors. Some can compute the whole solution space at once, but are too slow for practical purposes, others can only compute one solution. The latter, like *lp_solve* [BEN10], can be directed to compute a minimal solution (with respect to the sum over all values, leading to shortest firing sequences). Constraints can be used to discriminate a solution and force the IP solver to produce another (greater) one until no more solutions exist.

Definition 3 (Constraints). *We define two forms of constraints, both being linear inequations over transitions:*

- *a* jump constraint *takes the form $t < n$ with $n \in \mathbb{N}$ for a transition t.*
- *an* increment constraint *takes the form $\sum_{i=1}^{k} n_i t_i \geq n$ with $n_i \in \mathbb{N}$, $n \in \mathbb{N}$, and transitions t_i.*

Assume a linear system with a minimal solution b with $b(t) = n$, then an additional jump constraint $t < n$ discriminates b and leads to an incomparable solution. An increment constraint on the other hand may enforce a greater solution, adding some T-invariant to b. This idea is depicted in Fig. 2 where dashed arrows represent jumps and normal arrows the adding of T-invariants.

Since the solution space is semilinear we need jumps to get to other solution cones and increment constraints to go upwards in a cone. Jump and increment constraints can contradict each other, but it is possible to remove a jump constraint. Assume we have a solution a before and a solution b after adding a jump constraint. We construct one increment constraint per transition t with $t \geq b(t)$. This enforces at least the solution b. Removing the jump constraint now will not lead to an old (smaller) solution, especially not to a.

4 Building Constraints

Let us first argue that for a marking equation, any of the minimal solution vectors in B can be obtained by using jump constraints. For two solutions b and

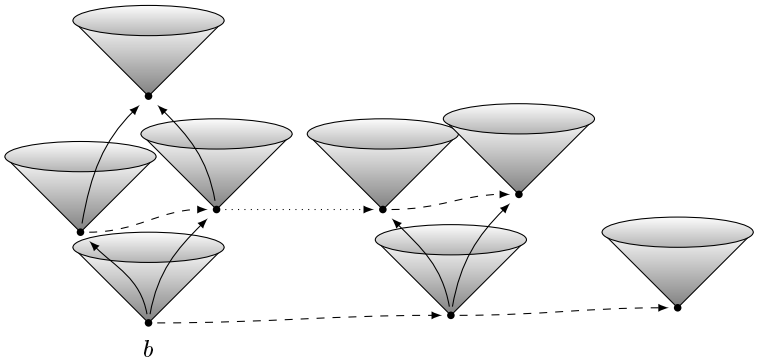


Fig. 2. Paths from the minimal solution b to any solution. Black dots represent solutions, cones stand for linear solution spaces over such solutions, which may or may not intersect or include each other. Normal arrows increment a solution by adding a T-invariant, dashed arrows are jumps to greater solutions. Such jumps can also occur on higher levels of linear solution spaces, shown by the dotted arrow

b' let $b \prec b'$ if we can change the solution our IP solver produces from b to b' by adding new constraints to a system consisting of the marking equation plus some (old) constraints.

Lemma 1 (Jumps to minimal solutions). *Let $b, b' \in B$ ($b \neq b'$) be base vectors of the solution space of the marking equation $m + Cx = m'$ plus some set of constraints $\mathcal{C}$. Assume b to be the minimal solution of the system. Then, we can obtain b' as output of our IP solver by consecutively adding jump constraints of the form $t_i < n_i$ with $n_i \in \mathbb{N}$ to $\mathcal{C}$.*

Proof. $b \prec b'$ and since b' is a minimal solution, $b \not\leq b'$. Thus, $\exists t \in T\colon b'(t) < b(t)$. Add the constraint $t < b(t)$ to $\mathcal{C}$, then b is not a solution anymore. Assume b'' to be our IP solver's new solution. As b' fulfills $t < b(t)$ it is still a solution, so from $b' \neq b''$ we conclude $b'' \prec b'$, and the same argument as above holds. Termination is guaranteed since there are only finitely many solutions $b'' \prec b'$.

Non-minimal solutions may not be reachable this way, since the argument "$b'(t) < b(t)$ for some t" does not necessarily hold. To determine those, increment constraints are necessary, and the latter can be obtained from partial solutions.

Definition 4 (Partial solution). *A* partial solution *of a reachability problem (N, m, m') is a tuple $(\mathcal{C}, x, \sigma, r)$ of*

- *a family of (jump and increment) constraints $\mathcal{C} = (c_1, \ldots, c_n)$,*
- *the $\prec$-smallest solution x fulfilling the marking equation of (N, m, m')* and *the constraints of $\mathcal{C}$,*
- *a firing sequence $\sigma \in T^*$ with $m[\sigma\rangle$ and $\wp(\sigma) \leq x$,*
- *a remainder r with $r = x - \wp(\sigma)$ and $\forall t \in T\colon (r(t) > 0 \Longrightarrow \neg m[\sigma t\rangle)$.*

The vectors x and r are included for convenience only, they can be computed from $\mathcal{C}$, σ, $\prec$, and the problem instance.

A full solution *is a partial solution $(\mathcal{C}, x, \sigma, r)$ with $r = 0$. In this case, σ is a firing sequence solving the reachability problem (with answer 'yes').*

If we obtain a partial solution with $r \neq 0$ there are not enough tokens on some places to fire the transitions in the remainder r. An underapproximation of tokens necessary can be computed from a graph G containing the transitions in r and the undermarked places, an edge from place s to transition t if there are not enough tokens on s to fire t, and an edge the other way if firing t increases the token count on s. From any strongly connected component SCC in G without incoming edges (= tokens produced by other components, a *source SCC*) we compute the minimal number k of tokens needed to activate any of its transitions. Our constraint now states that the number of tokens produced on the component's places should be increased by at least that number k. The tokens produced on a place p can be expressed as $\sum_{t:\, C(p,t)>0} C(p,t) \cdot x(t)$ for a solution vector x, so the constraint takes the form $\sum_{p \in SCC} \sum_{t \notin SCC:\, C(p,t)>0} C(p,t) \cdot t \geq k + \sum_{p \in SCC} \sum_{t \notin SCC:\, C(p,t)>0} C(p,t) \cdot x(t)$. Note that we sum up over transitions outside SCC only, as the transitions inside cannot produce tokens until the first of them gets activated, which is the aim of this constraint.

In our example net from Fig. 1 we start with the solution $x = 3a_1 + 3a_2 + 3\ell$. None of the transitions can fire three times, this leads us to the graph G: $y_2 \to a_2 \to x_2 \to \ell \to s \to a_1 \to x_2 \to a_2$. The only source SCC of G consists of just y_2 leading to the constraint $1 \cdot b_2 \geq k + 1 \cdot x(b_2) = k = 1$. Now the smallest solution becomes $x' = 2a_1 + 2a_2 + b_1 + b_2 + b_3 + 3\ell$ and only b_1 can fire as often as wanted. Our constructed graph G' now has z as its only source SCC, from which we obtain the constraint $1 \cdot c_1 \geq 1 + 1 \cdot x'(c_1) = 1$, leading to $x'' = 2a_1 + 2a_2 + b_1 + b_2 + b_3 + c_1 + c_2 + 3\ell$. After the sequence $b_1 c_1 b_2 c_2 b_3 \ell a_1$ we find a remainder $r = a_1 + 2a_2 + 2\ell$ and get again the graph G, but now with a constraint $1 \cdot b_2 \geq k + 1 \cdot x''(b_2) = 1 + 1 = 2$. After the next solution $x''' = a_1 + a_2 + 2b_1 + 2b_2 + 2b_3 + c_1 + c_2 + 3\ell$ with the graph $y_2 \to a_2 \to x_2 \to \ell \to s$ (a_1 not being in the remainder anymore) the constraint is increased to $1 \cdot b_2 \geq k + 1 \cdot x'''(b_2) = 1 + 2 = 3$. We now obtain the final solution of $3b_1 + 3b_2 + 3b_3 + c_1 + c_2 + 3\ell$. Note that the first step can also be done by a jump $a_1 < 3$, but this is impossible for the second step since $x'' \geq x'$.

5 Finding Partial Solutions

Finding maximal firing sequences σ for a solution x produced by the IP solver to obtain partial solutions can be done by a brute force tree search. The execution time may grow exponentially with the size of the solution, though. Partial order reduction, e.g. the stubborn set method [KSV06], can be applied to reduce the execution time. Other reductions can be thought of; even with stubborn sets a marking may appear more than once, either on a single path (the firing sequence is not minimal) or on permutated paths. Finding such spots allows to avoid going through subtrees unnecessarily.

6 Conclusion

The algorithm has been implemented in a tool named Sara [Wim10] and tested well with some small examples and a challenge posed by H. Garavel [Gar03] in 2003 so far. Four other tools, using different approaches, managed to solve the challenge, with run times from about 10 minutes to more than an hour (in 2003). A proof to the challenge consists of nearly 800 firing sequences of different length; those tools giving such a proof provided (much) longer firing sequences than our algorithm. Our implementation takes about 20 seconds on a simple linux PC and twice that much on a standard Windows PC under Cygwin (in 2010).

Due to the underapproximation of needed tokens when building constraints, our algorithm runs into trouble when the Petri net has high arc weights. A comparison to LoLA [Wol10] suggests an exponential loss with growing arc weights for some specialised examples.

Overall, we have the hope that our implementation is able to compete with other tools using state space exploration or symbolic model checking instead of structure analysis. Of course, more tests are necessary before a stable statement can be made. The algorithm presented here only tries to make a semi-decision for the positive case, but we are working on an extension to also make correct negative semi-decisions in many cases.

References

[BEN10] M. Berkelaar, K. Eikland, P. Notebaert: Lp_solve Reference Guide, `http://lpsolve.sourceforge.net/5.5/`, 2010.

[CMS06] G. Ciardo, R. Marmorstein, R. Siminiceanu: The saturation algorithm for symbolic state space exploration, *Software Tools for Technology Transfer* **8**:1, pp.4-25, 2006.

[Gar03] H. Garavel: Efficient Petri Net tool for computing quasi-liveness, `http://www.informatik.uni-hamburg.de/cgi-bin/TGI/pnml/getpost?id=2003/07/2709`, 2003.

[Kos82] S.R. Kosaraju: Decidability of reachability in vector addition systems, *Proceedings of the 14th Annual ACM STOC*, pp.267–281, 1982.

[KSV06] L.M. Kristensen, K. Schmidt, A. Valmari: Question-guided Stubborn Set Methods for State Properties, *Formal Methods in System Design* **29**:3, pp.215–251, Springer, 2006.

[Lam92] J. Lambert: A structure to decide reachability in Petri nets, *Theoretical Computer Science* **99**, pp. 79–104, 1992.

[Lip76] R.J. Lipton: *The Reachability Problem Requires Exponential Space*, Research Report 62, Yale University, 1976.

[May84] E. Mayr: An algorithm for the general Petri net reachability problem, *SIAM Journal of Computing* **13**:3, pp.441–460, 1984.

[Wim10] H. Wimmel: *Sara – Structures for Automated Reachability Analysis*, `http://www.informatik.uni-rostock.de/~nl/wiki/tools/download`, 2010.

[Wol10] K. Wolf: *LoLA – A low level analyzer*, `http://www.informatik.uni-rostock.de/~nl/wiki/tools/lola`, 2010.

Decidability Issues for Decentralized Controllability of Open Nets

Karsten Wolf

Universität Rostock, Institut für Informatik

Abstract. We sketch an undecidability result concerning the decentralized controllability problem of open nets and discuss some consequences.

1 Introduction

During the last years, we have used open nets (Petri nets with distinguished interface places) as a formal model for the behavior (protocol) of services [Wol09b]. Open nets can also be used as a tool in divide-and-conquer approaches to Petri net verification [Zai06,OWW10]. For this class of nets, it is interesting to study the interaction with their environment [Wol09a]. To this end, various notions of *controllability* [RW87] are in the main focus. So far, we concentrated on *centralized controllability* which asks whether there is a (monolithic) environment that can be composed in such a way to the given open net that the overall system is deadlock free or deadlock free and livelock free. In both cases, we were able to come up with decision procedures [LW10a] in the case where the given net has finitely many states while we showed undecidability for the case that the given open net is unconstrained [MSSW08].

Decentralized controllability is applied to an open net that has a partitioned interface. The question is: is there a tuple of environments, each communicating with one part of the interface and not directly communication with each other, such that the overall system is deadlock free or deadlock and livelock free? In [Wol09a], we gave a procedure only for the case that N does not run into cycles, i.e. never visits states twice. Decentralized controllability is a very useful notion as several practically relevant problems can be reduced to it:

- *Adaptibility* [GMW10]: Given a service S and a specification of elementary activities that specify the general space of actions of any adapter, is there another service R such that S and R can be made compatible by a mediating adapter? In this setting, the service R and the control unit of the adapter form a decentralized controller of a system consisting of S and the implementation of the possible elementary activities of the adapter.
- *Realizability* [LW10b]: Given a choreography (i.e. a language of event sequences to happen on the wires), are there services that interact in such a way that the observed communication fits to that language? In this case, the choreography can be transformed into an open net which has the tuple of realizing services as controller.

The notion of decentralized controllability must not be mixed up with the one of *autonomous controllability*. In the latter notion, the question is whether a decentralized

controller can be found in such a way that the different parts are not even coordinated at build time.

In this paper, we consider decentralized controllability for the setting where we want to enforce deadlock freedom and livelock freedom on the overall system. We further rule out cotrollers where, at any stage of communication, more than a given number k of tokens is pending on a message channel and we rule out unbounded open nets as these would be covered by the undecidability result of [MSSW08]. These restrictions ensure that the overall system is finite state and the undecidability result of [MSSW08] does not cover the setting.

To our best knowledge, undecidability of the mentioned problem has not been shown before while similar problem have been studied. Tripakis shows [Tri04] undecidability of decentralized controllabilty in a different setting. For his problem, the actual correctness property to be enforced by the synthesized controller is a parameter that is part of the input (in the shape of a regular language of events) while we consider the fixed setting of deadlock and livelock freedom. His undecidability result relies on coding Post's Correspondance Problem (PCP) into the mentioned language. Bontemps and Shoppens show [BS07] undecidability of the distributed realizability of a Life Sequence Chart (LSC) specification. Peculiarly, their setting permits communication between the distributed agents to be synthesized. In our situation, permission of direct communication between the distributed parts of the environment would immediately permit the conclusion that an open net is decentralized controllable if and only if it is centralied controllable. As we know that centralized controllability is decidable in our case, we conclude thet the setting of [BS07] includes some, maybe implicite, assumptions that are not compatible to our setting. In their argument, however, they as well use a reduction of PCP as their main argument.

Consequently, it is not surprising that our proof relies on a reduction from PCP as well. However, the particular execution of the reduction differs significantly. While Tripakis codes the PCP instance into the language used as a correctness criterion, Bontemps and Schoppens present the distibuted agents with a challenge generated from a nondeterministically selected PCP instance that is not solvable iff the PCP instance has a solution. We generate a controlling distributed strategy from a solution of a PCP instance and use the given open net as a verifier of that instance.

2 Open nets

In this section, we introduce the notion of open nets. The notion syntactically deviates a bit from earlier presentations. It is, however, semantically equivalent.

An open net extends a usual place/transition net $[P, T, F, m_0]$ with the following ingredients:

- A set M of *message shapes*,
- An *interface* I which is a set of *pins* partioned into ports Π_i ($I = \Pi_1 \cup \ldots \cup \Pi_n$, $\Pi_1 \cap \Pi_j = \emptyset$ for $i \neq j$); a pin is a pair $[m, d]$ where $m \in M$ and $d \in \{i, o\}$,
- A set Ω of *final markings*.
- A partial *labeling* that assigns a message shape to some of the transitions.

i and o represents incoming and outgoing messages, respectively. For a pin $[m, d]$, define the notion of a *dual pin* $\overline{[m, d]}$ by $\overline{[m, i]} = [m, o]$ and $\overline{[m, o]} = [m, i]$. For the interface of an open net, we require that, for all pins $[m, x]$, $[m, x] \in I$ implies $\overline{[m, x]} \notin I$. For a port Π_i, let the *dual port* $\overline{\Pi_i} = \{\overline{[m, x]} \mid [m, x] \in \Pi_i\}$.

A set of open nets $N_1, \ldots, N_k$ is *composable* iff, for all m and x, $[m, x] \in I_i$ and $[m, x] \in I_j$ implies $i = j$ (each connection is purely bilateral), and, for each pair of pins in the same port, their dual pins belong to the interface of the same net, or none of the dual pins belongs to any interface (ports signal communication with the same partner).

For composable open nets, the composition is built by

- building the disjoint partition of the places, transitions, and arcs of the involved net (if necessary by renaming);
- introducing additional places for all those pins where both pin and dual pin occur in some interface;
- inserting an arc from a transition t labeled m to the place inserted for m if $[m, o]$ occurs in the interface of the net containing t,
- inserting an arc from the place labeled m to a transition t labeled with m if $[m, i]$ occurs in the interface of the net containing t,
- removing the labels of all such connected transitions,
- letting the new ports be exactly those ports whose pins are not matched with dual pins in other nets,
- letting the initial marking be the disjoint partition $m_0 = m_{01} \oplus \ldots \oplus m_{0k}$, i.e. $m_0(p) = m_{0i}(p)$ for the unique i with $p \in \Pi_i$,
- letting $\Omega = \{m_1 \oplus \ldots \oplus m_k \mid m_i \in \Omega_i\}$.

We study the following *decentralized controllability problem*: Given some number k, open net N with ports $\Pi_1, \ldots, \Pi_n$, do there exist open nets $N_1, \ldots, N_n$ where each N_i has $\overline{\Pi_i}$ as its only port such that $N \oplus N_1 \oplus \ldots \oplus N_n$ is a k-bounded Petri net which is weakly terminating, i.e. from each reachable marking, a final marking is reachable?

3 Undecidability of Decentralized Controllability

In this section, we present a reduction of PCP to our setting of decentralized controllability. In PCP, we are given a set $d_1 = [u_1, v_1], \ldots, d_a = [u_a, v_a]$ of pairs of words over some fixed alphabet Σ. The problem is to decide whether there exists a finite word w over $\{d_1, \ldots, d_a\}$ such that $w_{[d_1 \leftarrow u_1, \ldots, d_a \leftarrow u_a]} = w_{[d_1 \leftarrow v_1, \ldots, d_a \leftarrow v_a]}$. It is well known that PCP is undecidable.

We prove:

Theorem 1. *If our decentralized controllability problem is decidable then PCP is decidable.*

We argue by translating an instance of PCP to an open net such that the PCP instance has a solution if and only if the open net is decentralized controllable. The idea is to design the open net such that the environment must be built from a candidate solution and leads to termination of the overall system if and only if the candidate solution is indeed a solution. Our open net is designed to have two ports, U and V. From port

U, we expect a sequence $d_{i_1}u_{i_1}d_{i_2}u_{i_2}\ldots$ followed by a concluding # while from V we expect a sequence $d_{i_1}v_{i_1}d_{i2}v_{i_2}\ldots$ followed by a concluding #.Together, the sequences should form a candidate solution for the PCP. Thus, U has input pins corresponding to $\Sigma \cup \{d_1, \ldots, d_a\} \cup \{\#\}$. Correspondingly, V has input ports corresponding to $\Sigma \cup \{d_1, \ldots, d_a\} \cup \{\#\}$ as well which need to be renamed bijectively in order to meet the syntactical constraints on pins. For each port there is a single output pin representing the acknowledgment of a received message.

The constructed open net behaves as follows: Whenever it sees more than one message in any of the two ports, it immediately moves into some deadlock or trap marking. This way, the partners are forced to send their content consecutively. Second, after having processed a message, it sends an acknowledgment to the sender. This way, the partners know when to savely send the next element of their candidate string. In the core part, the constructed net verifies the presented candidate strings. Verification includes

(1) Checking whether the candidate strings indeed correspond to a sequence of pairs of the given PCP instance;
(2) Checking whether the concatenation of the u_i is equal to the concatenation of the v_i.

A finite state system cannot check both items concurrently. The reason is that, in intermeadiate steps, a substring $u_1 \ldots u_b$ may have a different length than a corresponding substring $v_1 \ldots v_b$. The length difference may exceed any finite bound even if the substrings can be complemented to a solution of PCP. In the most popular proof of undecidability of PCP, the difference between $u_1 \ldots u_b$ and $v_1 \ldots v_b$ is used to code a comple Turing machine configuration including the state of the tape.

For this reason, the idea is to start with an internal nondeterministic decision and then to check only one of the above items. As the internal decision is not communicated to the environment, the partners can only control the open net by meeting both requirements.

If the internal decision is in favour of checking (1), the first input message on both ports is the identifier of some pair. If these identifiers are different or some other message is sent, we let the open net proceed into a deadlock. Otherwise, there is a branch depending on the pair identifier d_i and the open net matches the input on U with the sequence u_i and the input on V with v_i. This is clearly doable in a finite state fashion. If after that check there is # on both ports, the open net proceeds to a final state, otherwise it returns to the state where it expects the next path identifier.

If the decision is in favour of the second item, the open net checks whether the same letters are present in both ports. Thereby, all seen pair identifiers are ignored (i.e. acknowledged for stepping to the next input). If, at some stage, there appear # on both ports, we proceed to a final state. If a # appears on only one port, we proceed to a deadlock state. Otherwise, we continue forever.

At no time, any of the distributed partners is able to make educated guesses about the outcome of the internal decision between (1) and (2). In both cases, the individual communication is a straight alternation between inputting a letter of the candidate string and an acknowledgment. Any deviation from this procedure comes at the risk of deadlock if the open net is in the opposite mode.

4 Discussion

There is a few interesting questions that appear immediately.

First, what is the difference if we present the constructed net to a centralized environment? In the previous section, one crucial argument was that the partners cannot "sense" the outcome of the internal decision of the constructed open net. It turns out that a centralized partner *is* in fact able to sense that outcome in sufficiently many cases. This ability relies on the already mentioned difference between the lengths of substrings of the u_i and the v_i. If the open net is in mode (1), it processes its input such that it synchronises along the pair identifiers. That is, assuming that the length of the sequence of the u_i is longer than the one on the v_i, it proceeds farther into U than into V. Technically, "proceeding farther" corresponds to acknowledgements sent to U earlier than acknowledgnments to V. If the open net is in mode (2), it synchronises along letters of the alphabet, i.e., it would send acknowledgments for subsequent pairs on V before finishing acknowwledgements on U. This way, the outcome of the internal decision between (1) and (2) becomes visible to the partner at least on those PCP instances where all solutions go through intermediate steps with greatly diverging lengths between the u_i and v_i sequences. Once having recognised the mode of the open net, the partner can now react with nonstandard strategies. In mode (1) the partner can finish after having sent a complete pair even if the two sequences do not match. In mode (2), the partner canm send some matching letters on U and V even if there is no corresponding PCP pair. Hence, the constructed open net would not establish a valid PCP reduction.

Second, what about using only deadlock freedom as a correctnes criterion? Again, the used constructive ides does not work. If deadlock freedom is the only criterion, the distributed partner may choose an infinite sequence of pairs where, for all finite prefixes, the sequence of the u_i matched the corresponding initial segment of the v_i. There exist PCP instances with that property which do not have solutions. They can be built from nonterminating Turing machines using the translation used in the well known reduction of the halting problem to PCP. Although the PCP has no solution, the partners provide input forever without failing in the (1) or (2) checks. It is evident, that it is impossible to add another finite state check that identifies the nontermination of the input as this would require reasoning about the halting problem of Turing machines. We conclude that decidability of decentralized controllabillity w.r.t. deadlock freedom is still open.

Third, what about the autonomous setting? It is easy to see that the two partners are build-time coordinated. If they want to control the constructed open net, they have to agree on some PCP solution (although they do not communicate with each other while feeding it into the open net). In the autonomus setting, there is no reason to believe that there is any canonical choice for the individual partners that composes to a strategy.

Gven the importance of decentralized controllability for interesting problems like adaptability of realizability, the floor is now open for proposing approximations or solutions for subclasses of open nets. As one of the core aspects in our proof was the initial hidden choice between (1) and (2), a closer look into the disclosure of internal choices to the partners may be an interesting starting point.

References

[BS07] Yves Bontemps and Pierre-Yves Schobbens. The computational complexity of scenario-based agent verification and design. *J. Applied Logic*, 5(2):252–276, 2007.

[GMW10] Christian Gierds, Arjan J. Mooij, and Karsten Wolf. Reducing adapter synthesis to controller synthesis. *IEEE T. Services Computing*, 2010. (accepted for publication in July 2010).

[LW10a] Niels Lohmann and Daniela Weinberg. Wendy: A tool to synthesize partners for services. In Johan Lilius and Wojciech Penczek, editors, *31st International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, PETRI NETS 2010, Braga, Portugal, June 21-25, 2010, Proceedings*, volume 6128 of *Lecture Notes in Computer Science*, pages 297–307. Springer-Verlag, June 2010.

[LW10b] Niels Lohmann and Karsten Wolf. Realizability is controllability. In Cosimo Laneve and Jianwen Su, editors, *Web Services and Formal Methods, 6th International Workshop, WS-FM 2009, Bologna, Italy, September 4-5, 2009, Revised Selected Papers*, volume 6194 of *Lecture Notes in Computer Science*, pages 110–127. Springer-Verlag, September 2010. (in press).

[MSSW08] Peter Massuthe, Alexander Serebrenik, Natalia Sidorova, and Karsten Wolf. Can I find a partner? Undecidablity of partner existence for open nets. *Inf. Process. Lett.*, 108(6):374–378, November 2008.

[OWW10] Olivia Oanea, Harro Wimmel, and Karsten Wolf. New algorithms for deciding the siphon-trap property. In Johan Lilius and Wojciech Penczek, editors, *31st International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, PETRI NETS 2010, Braga, Portugal, June 21-25, 2010, Proceedings*, volume 6128 of *Lecture Notes in Computer Science*, pages 267–286. Springer-Verlag, June 2010.

[RW87] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete -event processes. *SIAM J. Control and Optimization*, 25(1), 1987.

[Tri04] Stavros Tripakis. Undecidable problems of decentralized observation and control on regular languages. *Inf. Process. Lett.*, 90(1):21–28, 2004.

[Wol09a] Karsten Wolf. Does my service have partners? *LNCS ToPNoC*, 5460(II):152–171, March 2009. Special Issue on Concurrency in Process-Aware Information Systems.

[Wol09b] Karsten Wolf. A theory of service behavior. In Oliver Kopp and Niels Lohmann, editors, *Proceedings of the 1st Central-European Workshop on Services and their Composition, ZEUS 2009, Stuttgart, Germany, March 2-3, 2009*, volume 438 of *CEUR Workshop Proceedings*, pages 1–7. CEUR-WS.org, March 2009.

[Zai06] D. A. Zaitsev. Compositional analysis of Petri nets. *Cybernetics and Systems Analysis, Volume 42, 1, 2006*, pages 126–136, 2006.

On the notion of deadlocks in open nets

Richard Müller

Institut für Informatik, Humboldt-Universität zu Berlin, 10099 Berlin, Germany
richard.mueller@informatik.hu-berlin.de

Abstract. Before a system is built in Service Oriented Computing from interacting services, it is modeled and verified with regard to different behavioral correctness criteria, among others, deadlock freedom. For this purpose, open nets as a special class of Petri nets are frequently used. In this paper, we present and compare three formalizations of when two services interact deadlock freely. We specifically highlight subtle details of existing formalizations and propose a new formalization that matches the intuition in every case.

1 Introduction

In the context of *Service Oriented Computing*, a system is built from interacting *services* ([4]). A service is a well defined, self-contained component that provides standard *business functionality* which is accessible via a standardized *interface*. It interacts with other services through *communication*, while being independent of their state or context ([1]). Best engineering practice suggests a system to be modeled before it is physically built. *Open nets* ([3]) are a special class of Petri nets, which have been proven notably helpful in modeling the *behaviour* of open systems, e.g., services.

There exist well-developed methods to analyze the behavior of a service modeled as open net with regard to different correctness criteria ([6]), among others, *deadlock freedom*. Intuitively, a *deadlock* describes an unwanted situation in which further progress in relevant (but not necessarily all) parts of a system is impossible. A system reaching a deadlock is typically considered malfunctioning. This makes the absence of a reachable deadlock, i.e., deadlock freedom, desirable behavior. We call two services *deadlock free partners* if they interact deadlock freely. There exist various formalizations of deadlocks and deadlock free partners in the model of open nets ([2], [6]). In this paper, we present and analyze two existing notions and highlight deficits of these notions arising in subtle situations. Based on these insights, we propose a new formalization of deadlock free partners which overcomes these deficits. We finally compare all three notions.

The rest of this paper is organized as follows. In Sect. 2, we briefly introduce basic concepts of Petri nets and our formal model for services, i.e., open nets. Then, in Sect. 3, we present three different formalizations of deadlocks and deadlock free partners. Section 4 is devoted to the comparison of the different notions. Finally, Sect. 5 concludes the paper and gives directions for future work.

2 Preliminaries

In this section, we recall *open nets* as a Petri net model for services. We start from marked net structures, i.e. place/transition nets, with the usual meaning of the con-

stituents and the standard firing rule. We assume the standard notions of *markings*, *enabledness*, *firing*, and *reachability* in Petri nets. Otherwise, see [5] for an introduction.

Definition 1 (open net structure, partner structures, composition). *Let $N = (P,T,F)$ be a net structure and let I, O be two disjoint sets of places of N, such that ${}^\bullet I = O^\bullet = \emptyset$, and the environment ${}^\bullet t \cup t^\bullet$ of each transition t of N contains at most* one *node of $I \cup O$. Then N together with (I,O) is an* open net structure. *(I,O) is the* interface *of N, $IP(N) =_{def} I \cup O$ is the set of* interface places *of N, and nodes in $IP(N) \cup I^\bullet \cup {}^\bullet O$ are* interface *nodes of N, opposed to* inner *nodes of N.*

For $i = 1,2$ let $N_i = (P_i,T_i,F_i)$ together with the interface (I_i,O_i) be open net structures. Without loss of generality, we assume each node x shared by N_1 and N_2 be an interface place: Otherwise, replace x by different instances in both open net structures. N_1 and N_2 are partner structures *iff $I_1 = O_2$ and $I_2 = O_1$. The* composition *of N_1 with N_2 is the open net structure $N_1 \oplus N_2 =_{def} (P_1 \cup P_2, T_1 \cup T_2, F_1 \cup F_2)$ together with the interface $(I \setminus O, O \setminus I)$, where $I = I_1 \cup I_2$ and $O = O_1 \cup O_2$.*

Inner nodes model a service's business functionality and interface nodes model a service's standardized interface. The concept of partners expresses two services' capability of proper communication. On the level of open nets, communication of two partners N_1 and N_2 is the firing of an interface transition of N_1 or N_2 in their composition $N_1 \oplus N_2$. Additionally, we distinguish situations which N_1 and N_2 shall reach together, i.e., *target markings*.

Definition 2 (open net, partners). *Let N be an open net structure, let m be a marking of N with no interface place marked, and let Ω be a set of markings of N with no interface place marked. Then N together with m and Ω is an* open net*, with m the* initial marking *of N, and Ω the set of* target markings *of N.*

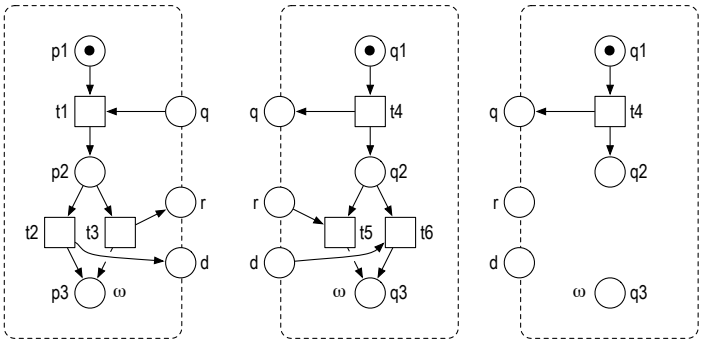
Let N_1 and N_2 be two open nets with initial markings m_1, m_2 and target markings Ω_1, Ω_2, respectively. N_1 and N_2 are partners *iff N_1 and N_2 are partner structures. The initial marking m of $N_1 \oplus N_2$ is defined by $m =_{def} m_1 + m_2$, the target markings Ω of $N_1 \oplus N_2$ are $\Omega =_{def} \{m_1 + m_2 \mid m_1 \in \Omega_1, m_2 \in \Omega_2\}$.*

We depict the interface of an open net N by drawing its interface places on a dashed rectangle which contains all inner nodes and interface transitions of N. If N has exactly one target marking m, we indicate a place p of N with ω if and only if p is marked in m. See Fig. 1 for some examples. For an open net N, its *inner structure* $inner(N)$ is defined by removing the interface places of N, and restrict its initial marking and target markings accordingly.

3 Deadlocks

In the rest of this paper we assume freely chosen partners N and P. We shall compare several deadlock notions for open nets by means of N and P being *deadlock free partners*. Intuitively, deadlock free interaction is the avoidance of unwanted situations in which further progress in relevant parts of a system is impossible. In the model of open nets, we formulate this as follows: Whenever one relevant part, i.e., N or P, of the system $N \oplus P$ stops, than there is a wanted situation, i.e., a target marking, reachable

in $N \oplus P$, or N and P will eventually communicate again. Massuthe [2] uses the notion of a *dead marking* to define a deadlock of two interacting services. This definition discriminates between target and non-target dead markings, the latter ones must not be reachable.



(a) N **(b)** P **(c)** Q

Fig. 1. The open nets N, P and Q. N and P are DF_{PM}-partners, N and Q are not.

Definition 3 (dead marking, deadlock$_{PM}$, DF_{PM}-partners). *A marking m of N is* dead *in N iff no transition of N is enabled in m. A dead non-target marking of N is a* deadlock$_{PM}$ *of N. P is DF_{PM}-*interacting *with N (synonymous to N and P are DF_{PM}-*partners*) iff no reachable marking of $N \oplus P$ is a deadlock$_{PM}$ of $N \oplus P$. We write $DF_{PM}(N)$ for the set of all open nets DF_{PM}-interacting with N, and DF_{PM} for the set of all DF_{PM}-partners.*

For example, the open net N (Fig. 1(a)) models a service which is able to receive a query (input place q), and then either replies to the query (output place r) or denies access (output place d). Afterwards, the service can terminate in the target marking $[p3]$. Typically, the decision between reply or denial is made depending on data inside the received query. As we abstract from data in open nets, we model the decision's outcome using non-determinism (interface transitions $t2$ and $t3$). The open net P (Fig. 1(b)) is DF_{PM}-interacting with N: Whether N sends a reply or denial, P is able to receive it. Thus, the only marking of $N \oplus P$ with no transition enabled is the target marking $[p3,q3]$ of $N \oplus P$, i.e., $[p3,q3]$ is no deadlock$_{PM}$. In contrast, the open net Q (Fig. 1(c)) is not DF_{PM}-interacting with N: There are two deadlocks$_{PM}$, $[p3,r,q2]$ and $[p3,d,q2]$, reachable in $N \oplus Q$.

Massuthe's definition provides a good first impression on how to formalize deadlocks in open nets. However, it is not precise enough. As an example, consider the open net R (Fig. 2(a)), which originates from Q by adding an *inner loop* (transition $t5$). R is as malfunctioning as Q: Two markings, $[p3,r,q2]$ and $[p3,d,q2]$, are reachable in $N \oplus R$, such that neither N nor R is in a target marking, and they will never communicate again. Intuitively, N and R should not be deadlock free partners. Nevertheless,

R is DF_{PM}-interacting with N, i.e., N and R are DF_{PM}-partners. This example shows a general drawback of Definition 3: *Every* open net N has at least one open net M which is DF_{PM}-interacting with N. M consists of an interface that is compatible with N's interface, and an inner loop without communication. Wolf [6] refines Massuthe's notion and introduces *responsiveness* of open nets to overcome endless loops without communication [1].

Definition 4 (responsiveness, DF_{KW}-partners). *An open net N is* responsive *iff, from each reachable marking in inner(N), a marking is reachable in inner(N) which is a target marking or which enables an interface transition of N. P is DF_{KW}-*interacting *with N iff P is responsive and no reachable marking of $N \oplus P$ is a deadlock$_{PM}$ of $N \oplus P$. N and P are DF_{KW}-*partners *iff N is DF_{KW}-interacting with P and P is DF_{KW}-interacting with N. We write $DF_{KW}(N)$ for the set of all open nets DF_{KW}-interacting with N, and DF_{KW} for the set of all DF_{KW}-partners.*

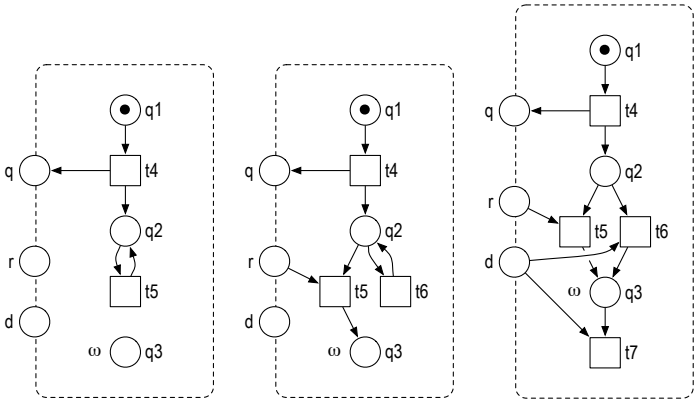


Fig. 2. Another three open nets R, S and T.

Responsiveness rules out some intuitively unwanted interacting open nets with endless loops without communication (like R in Fig. 2(a), which is not DF_{KW}-interacting with N), but unfortunately not all. Figure 2 depicts an example. The open net S (Fig. 2(b)) is responsive and DF_{KW}-interacting with N. However, we would intuitively classify the reachable marking $[p3, d, q2]$ of $N \oplus S$ as a deadlock: Neither N nor S is in a target marking, and they will never communicate again. Additionally, responsiveness is a rather strong restriction on partners. Intuitively, N and the open net T (Fig. 2(c)) are deadlock free partners: $N \oplus T$ has no reachable marking which enables transition $t7$. Nevertheless, T is not responsive and not DF_{KW}-interacting with N according to Def. 4.

[1] Additionally, Wolf restricts Def. 4 to bounded open nets and bounded communicating partners for decidability reasons [3]. As we just compare the notion of deadlocks and deadlock free partners (whether decidable or not), we do not employ this restriction.

In order to overcome those drawbacks, we propose a new, third definition of deadlocks and deadlock free partners. Our definition is based on the observation that a deadlock of two interacting services N and P is intuitively described from the view of one of the involved services, e.g., N, by three facts: (1) N is in an unwanted situation, (2) further progress in $N \oplus P$ by N alone is impossible, i.e., there is a need for interaction of P with N, and (3) P will never interact with N anymore. Therefore, we propose to describe deadlocks in open nets *locally*, i.e., from the view of one service, rather than from a global point of view on both partners. We do so by using the target markings of only one of the involved open nets, and by considering *silent* markings, which capture the absence of further communication between the partners.

Definition 5 (silent marking, deadlock$_{RM}$, DF_{RM}-partners). *A marking m of $N \oplus P$ is* silent *in $N \oplus P$ iff for all markings m' of $N \oplus P$ reachable from m holds: At most inner transitions of P are enabled in m'. A silent marking m of $N \oplus P$ such that $m_{|N}$ is no target marking of N, is a* deadlock$_{RM}$ *of $N \oplus P$. P is DF_{RM}-*interacting *with N iff no reachable marking of $N \oplus P$ is a deadlock$_{RM}$ of $N \oplus P$. N and P are DF_{RM}-*partners *iff N is DF_{RM}-interacting with P and P is DF_{RM}-interacting with N. We write $DF_{RM}(N)$ for the set of all open nets DF_{RM}-interacting with N, and DF_{RM} for the set of all DF_{RM}-partners.*

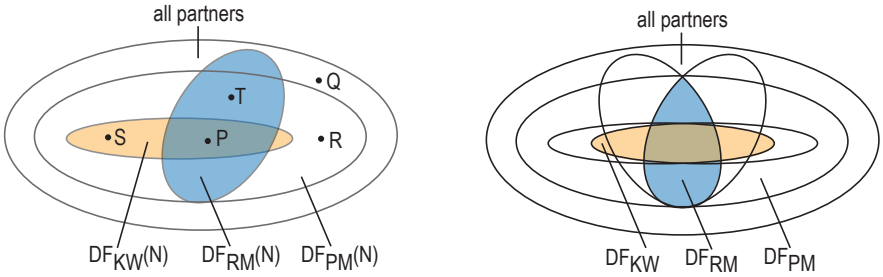
Notice the discrimination between wanted and unwanted situations by means of N's target markings instead of $N \oplus P$'s target markings like in Def. 3 and 4. Definition 5 matches our intuition in every previously mentioned example: N and P as well as N and T are DF_{RM}-partners, whereas all other partners are not.

4 Comparison

In this section, we shall compare Definitions 3, 4, and 5 in terms of the characterized deadlock free partners, and the complexity of the notions. Given an open net N, the set of open nets deadlock freely interacting with N generally differ, depending on which definition of deadlock free interaction is employed. Every open net DF_{KW}-interacting with N is DF_{PM}-interacting with N as well, the opposite is not true (see R and S in Fig. 2 for an example). An open net DF_{RM}-interacting with N can be DF_{PM}-interacting with N, DF_{KW}-interacting with N, or neither. Consider Fig. 3(a) for an illustration of above coherences as Venn diagram.

An open net DF_{RM}-interacting with N, which is not DF_{PM}-interacting with N, may seem counter-intuitive for a notion of deadlock free partners. However, Def. 5 says DF_{RM}-partners have to be mutually DF_{RM}-interacting with each other, therefore every two DF_{RM}-partners are DF_{PM}-partners as well. Again, this matches our intuition. Every two DF_{KW}-partners are DF_{PM}-partners. The sets of all DF_{KW}-partners and all DF_{RM}-partners lie diagonally: There exist DF_{KW}-partners, which are no DF_{RM}-partners (e.g. the open nets N and S in Fig. 1(a) and 2(b)), and vice versa (e.g. the open nets N and T in Fig. 1(a) and 2(c)). Figure 3(b) depicts above coherences.

All three presented formalizations differ in how complex it is to check whether two given open nets N and P are deadlock free partners or not. Checking according to Def. 4 is more complex than according to Def. 3, as responsiveness of N and P has to be checked additionally. Checking according to Def. 5 is more complex than according



(a) deadlock free interaction **(b)** deadlock free partners

Fig. 3. The sets of open nets deadlock freely interacting with N (Fig. 1(a)), and the sets of deadlock free partners.

to Def. 4, because it is more complex to check if a marking m of $N \oplus P$ is silent than to check if m is dead. To sum up, there is a trade-off between matching our intuitive deadlock notion and the complexity of a formalization.

5 Conclusion

We have presented two existing formalizations of when two services interact deadlock freely, and gave an idea of their deficits in subtle situations. Our approach for overcoming these deficits is a third formalization, describing deadlocks from a local point of view rather than globally. To the best of our knowledge, this is new. Finally, we compared all three formalizations, and highlighted their differences in complexity as well as matching our intuition. Though more complex, the new formalization matches our intuition in every case.

There exists strong theory for characterizing all deadlock free partners, based on the two existing formalizations ([2], [6]). It is part of further work to determine if those theory can be applied with the new formalization as well.

References

1. Alonso, G., Casati, F., Kuno, H.A., Machiraju, V.: Web Services - Concepts, Architectures and Applications. Data-Centric Systems and Applications, Springer (2004)
2. Massuthe, P.: Operating Guidelines for Services. Dissertation, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II; Eindhoven University of Technology (Apr 2009), iSBN 978-90-386-1702-2
3. Massuthe, P., Serebrenik, A., Sidorova, N., Wolf, K.: Can I find a partner? Undecidablity of partner existence for open nets. Inf. Process. Lett. 108(6), 374–378 (Nov 2008)
4. Papazoglou, M.P.: Service-oriented computing: Concepts, characteristics and directions. In: WISE. pp. 3–12. IEEE Computer Society (2003)
5. Reisig, W.: Petri Nets: An Introduction, Monographs in Theoretical Computer Science. An EATCS Series, vol. 4. Springer (1985)
6. Wolf, K.: Does my service have partners? T. Petri Nets and Other Models of Concurrency 2, 152–171 (2009)

A graphical user interface for service adaptation

Christian Gierds[1] and Niels Lohmann[2]

[1] Humboldt-Universität zu Berlin, Institut für Informatik,
Unter den Linden 6, 10099 Berlin, Germany
gierds@informatik.hu-berlin.de
[2] Universität Rostock, Institut für Informatik, 18051 Rostock, Germany
niels.lohmann@uni-rostock.de

Abstract. Service-oriented computing aims at composing independent services to achieve a common goal. Although very flexible, this independence may result in incompatibilities. A pragmatic approach to overcome such incompatibilities offer *adapters*. An adapter is again a service which reorganizes the message exchange in a service composition to avoid incompatibilities.
Given a set of domain-specific message transformation rules, adapters can be generated fully automatically. This paper presents a graphical user interface to support the systematic design of these transformation rules.

1 Introduction

Service-oriented computing [10] aims at replacing large monolithic systems by a composition of *services*. By abstracting from underlying technologies and implementations, it is possible to focus on the functionality of a service and to reuse it in other compositions. Consequently, services can be designed independently from the compositions they are used in, which in turn allows for faster production cycles at lower costs. A downside of this flexibility is the possible incompatibility of independently designed services. To avoid the redesign of incompatible services, an *adapter* (sometimes called mediator) can resolve incompatibilities by manipulating the communication protocol between the incompatible services. State-of-the-art techniques [1,2,3,5,9,4,11,6] allow to generate adapters automatically given a set of domain-specific message transformation rules.

So far, the design of such transformation rules was out of scope of most existing adapter generation approaches, and of course transformation rules can be formulated independently of a concrete service composition. However, it is likely that the design of such rules can be accelerated if the services to be adapted is taken into account. This paper follows this idea and presents an approach to iteratively create proposals for the designer of semantic message transformation rules. These proposals are derived by diagnosing behavioral incompatibilities. The approach is complete; that is, if services can be adapted using some rule set, then this set can be constructed.

The rest of this paper is organized as follows. The next section briefly sketches the automatic generation of adapters and introduces a running example. It further

discusses how transformation rule proposals can be derived from diagnosed incompatibilities. Section 3 presents the main contribution of this paper, a Web-based graphical user interface for the iterative construction of transformation rules. Finally, Sect. 4 discusses possible future extensions and concludes the paper.

2 Adapter generation

We shall briefly outline the basic concepts of an adapter generation algorithm and its meaning for finding transformation rules in this section.

2.1 Synthesis using message transformation rules rules

For adapting two services A and B several approaches agree on using *message transformation rules* (creating, removing, or changing messages) [1,2,3,5,9,4,11,6], which handle semantical incompatibilities.

We concentrate on the approach of Gierds et al. [6], consisting of two: They model transformation rules as an artifact called *engine* E. Then they try to synthesize a *controller* C, such that the composition of A, B, E, and C behaves according to a certain correctness criterion (e. g., deadlock freedom). The composition of E and C thus yields an adapter for A and B and ensures semantical and behavioral correctness of the two services.

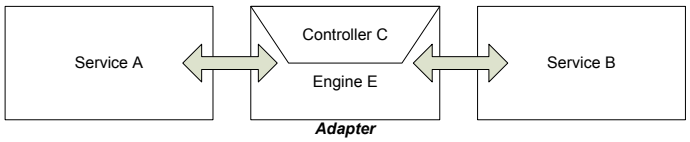
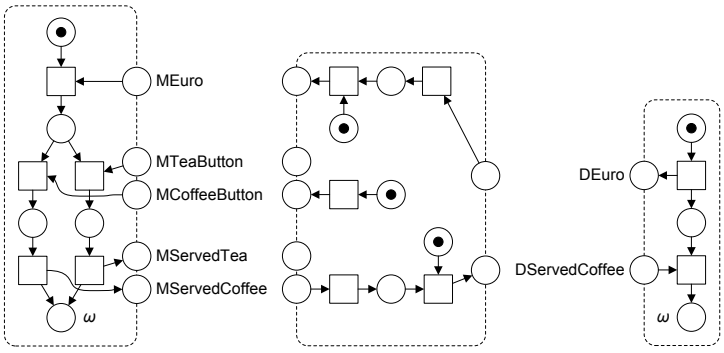


Fig. 1. Two services A and B and an adapter (engine E and controller C) in the middle

Figure 1 depicts a schema of this approach. The two services A and B communicate via the adapter in the middle. As it is indicated, an adapter comprises two parts: The engine E implements the message transformation rules and thus ensures semantically correctness. The controller C ensures correct behavior; that is, the correct order of applying rules and sending messages to the services.

Figure 2 shows a small example based on open nets [7], an extension of classical Petri nets. Interface places are positioned on the dashed border of a net. As running example, the model of a beverage vending machine is depicted on the left (cf. Fig. 2(a)). After receiving a Euro (MEuro), either the tea (MTeaButton) or the coffee button (MCoffeeButton) must be pressed. Afterward the appropriate beverage is delivered (MServedTea and MServedCoffee, resp.). On the right (2(c)), a coffee drinker provides a Euro (DEuro), then forgets to press a button, and



(a) Beverage vending machine (b) Adapter (c) Coffee drinker

Fig. 2. The two services to be adapted with an adapter

waits for its coffee (DServedCoffee). Obviously, this service is not compatible to the vending machine. To overcome this incompatibility, the adapter in Fig. 2(b) simply transforms a DEuro message to an MEuro message and MServedCoffee to DServedCoffee, which seems obvious concerning the names. Further it creates a MCoffeeButton message. Due to structural reduction, we may identify the controller part only by the initially marked places, allowing each rule to be applied exactly once.

To synthesize such an adapter automatically, the before-mentioned three rules must be provided as input to the synthesis algorithm.

2.2 Finding additional rules

As mentioned, one of the essential parts of the adapter approach is the set of message transformation rules. Although correct in isolation (in the example, there exist compatible drinker and vending machine services, resp.), two services may only be adaptable if a certain set of rules is provided. So whenever the synthesis algorithm fails to create an adapter, this is caused by missing rules.

Previous approaches almost totally rely on Semantic Web technologies for providing rules. We will briefly sketch an idea on how to extend the set of transformation rules by behavioral diagnosis. During controller synthesis deadlocks will be reached if no deadlock free controller exists. These deadlocks provide valuable information, how an additional rule might look like. The setting does not allow to change one of the services, but we are free to add as many new rules as we like. Let m be a deadlock marking, then we can analyze which messages remain in the engine, thus are *pending* and could be transformed. Further we check whether one of the services could continue if we provided a certain message, so we check which messages are *required*. A new rule then may transform pending into required messages. Consequently, m will no longer be a deadlock marking, because we can apply the newly added transformation rule now (which behaves

like a transition added to a net, which is enabled by the pending messages). This step can be repeated until we find a controller and therefore an adapter, or until we are no longer able to add meaningful transformation rules.

For our example in Fig. 2, starting with an empty set of transformation rules, our proposed algorithm will state that DEuro is pending (in Fig. 2(c) the appropriate transition is activated and thus fired), MEuro is required, and thus we may add the rule transforming MEuro to DEuro. In the next step (as shown in Fig. 3) we will see, that MTeaButton and MCoffeeButton are required. After providing a corresponding rule, we will see, that MServedCoffee is pending, and DServedCoffee is required. Finally the rule set is sufficient and we gain an adapter for our example.

In the given example the single steps are straightforward. For more complex examples, the number of deadlocks as well as the number of details for each deadlock grows significantly. Thus we need a good representation for this kind of information.

3 Using the Web as graphical user interface

Interactive approaches highly benefit from a concise way of presenting information. A user must be able to quickly access all relevant information. Graphical solutions with means to highlight or hide information based on a user's demands clearly excel console applications in this point. Marlene as single purpose tool has already been integrated into *service-technology.org/live* [8], which is our platform to demonstrate the functionality of our tools and allow a user to perform more complex tasks involving several of our tools by simply using a Web browser. The previously described interactive approach has also been integrated there and can be tested at the URL `http://service-technology.org/live/marlene`.

In an interactive approach, we do not only need to present the input and output artifacts, but also intermediate information which shall enable the user to make a next step. In our case, we have to list all possible suggestions for adding new transformation rules without showing all details at once and thus confusing the user.

Figure 3 shows the essential part for our approach: an editing field for transformation rules and below a table with information on all deadlocks, which may help in providing additional rules. Additionally, but not depicted here, the services are visualized. We divide the table in the following columns:

- *type* might either be deadlock or a livelock (in case we want an adapter ensuring also livelock freedom);
- the *pending messages*, which can be used in a rule on the left side;
- the *required messages*, of which at least one must be provided for resolving a deadlock in one of the services
- the *triangle* button, for showing additional information on a deadlock or livelock

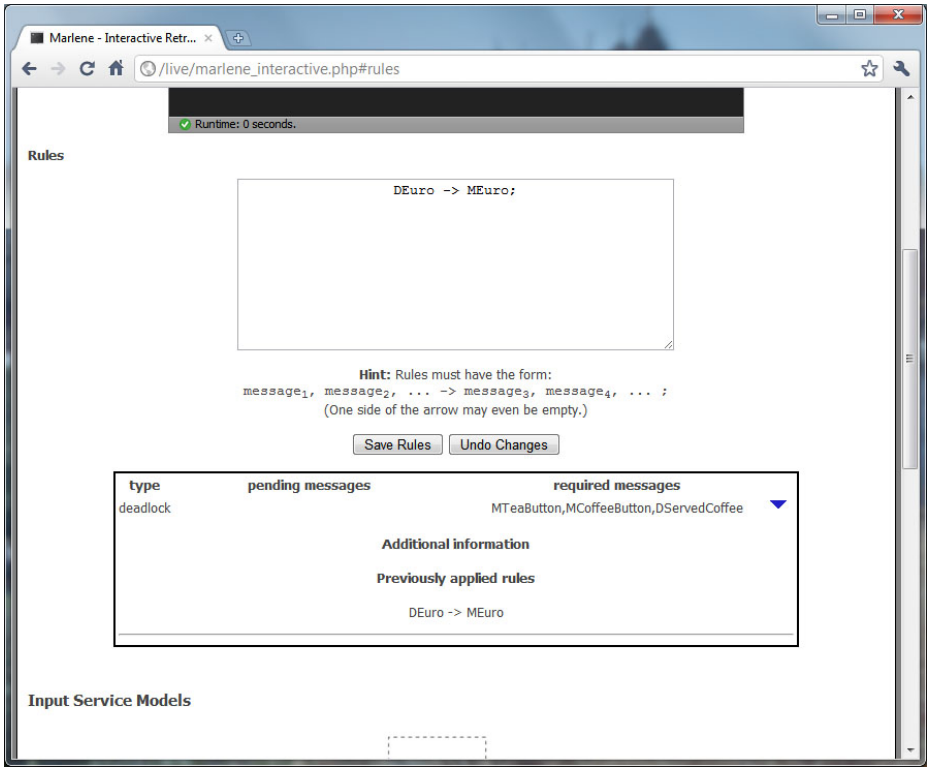


Fig. 3. Screenshot of interactive site

The additional information might state, that one of the services is already in a final state thus needing no further attention, and which rules have been applied prior to reaching a certain deadlock.

We have decided to initially show only the first line for each deadlock (the line starting with *deadlock*, thus hiding all additional information at first). As we can see in Fig. 3, providing all available information on a deadlock in a clear way requires a lot of space. Presenting a larger number of deadlock then would almost immediately require the user to scroll the page. This would clearly hamper deciding which deadlock to resolve, because a direct comparison of deadlocks would always depend on scrolling.

In our understanding, pending and required messages are the most important information for a certain deadlock. Thus showing only this piece of information should be sufficient in most case. By clicking the triangle at the end of a line the user gets additional information as described above.

The user is frfee to add and change the rules arbitrarily in the text field. By clicking *Save Rules* the page is updated and information based on the new rule set are provided. Finally, if a sufficient set of rules was added, the generated adapter is presented.

4 Conclusion and outlook

We have presented a first idea for the interactive retrieval of transformation rules in the setting of service adaptation. We have also focused on an appropriate visualization of information. First tests indicate that interaction as described here with the proposed degree of initial information offers easy access to the approach. This of course is only a first step.

First, the algorithm for finding new transformation rules has to be described in detail and we have to proof its feasibility in adapter generation (i. e., that when an adapter exists, the algorithm leads to a corresponding set of rules). Second, we have to evaluate acceptance of the Web site. Only feedback of real users playing through real-word examples will give us valuable hints on how to improve presentation of our tool.

Especially the order of different deadlocks might facilitate decision, which one to resolve — the higher the position of a deadlock, the more likely it will be considered. Here we have to find heuristics based on user behavior and its reason to prefer certain deadlocks. Also highlighting certain situations (e. g., both services are already in a final state, but superfluous messages must be removed) might help a user to pick more goal leading deadlocks.

Although not having finished the approach, yet, using a Web front-end for our prototype allows us to test our approach from the very beginning and distribute it easily, thus gaining valuable feedback from prospective users.

References

1. Benatallah, B., Casati, F., Grigori, D., Motahari Nezhad, H.R., Toumani, F.: Developing adapters for web services integration. In: CAiSE. pp. 415–429 (2005)
2. Bracciali, A., Brogi, A., Canal, C.: A formal approach to component adaptation. Journal of Systems and Software 74(1), 45–54 (Jan 2005)
3. Brogi, A., Canal, C., Pimentel, E., Vallecillo, A.: Formalizing web service choreographies. In: WS-FM'04. ENTCS, vol. 105, pp. 73–94 (2004)
4. Brogi, A., Popescu, R.: Automated generation of BPEL adapters. In: ICSOC. pp. 27–39 (2006)
5. Dumas, M., Spork, M., Wang, K.: Adapt or perish: Algebra and visual notation for service interface adaptation. In: Business Process Management. pp. 65–80 (2006)
6. Gierds, C., Mooij, A.J., Wolf, K.: Reducing adapter synthesis to controller synthesis. Transactions on Services Computing (accepted for publication) (2010)
7. Kindler, E.: A compositional partial order semantics for Petri net components. In: ATPN'97. LNCS, vol. 1248, pp. 235–252 (1997)
8. Lohmann, N.: service-technology.org/live - replaying tool experiments in a Web browser. In: BPM 2010 Demonstration Track. CEUR Workshop Proceedings, vol. 615, pp. 64–68. CEUR-WS.org (2010)
9. Motahari Nezhad, H.R., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-automated adaptation of service interactions. In: WWW. pp. 993–1002 (2007)
10. Papazoglou, M.P.: Web Services: Principles and Technology. Pearson - Prentice Hall, Essex (Jul 2007)
11. Wang, K., Dumas, M., Ouyang, C., Vayssière, J.: The service adaptation machine. In: ECOWS. pp. 145–154 (2008)

Managing test suites for services

Kathrin Kaschner

Universität Rostock, Institut für Informatik, 18051 Rostock, Germany
kathrin.kaschner@uni-rostock.de

Abstract. When developing an existing *service* further, new functionalities can be added and existing functionalities can be changed or removed. Consequently, also test cases have to be added to or removed from the existing test suite. In this paper, we present an idea how a test suite can be adjusted to these changes. Thereby, we focus on modifications concerning the *communication behavior* of a service.

1 Introduction

Testing is an effective instrument to detect errors in software. However, for thorough testing the number of required test cases increases rapidly with growing software complexity. To reduce the effort for testing, test cases are generated and executed by tool support as much as possible.

In the paradigm of *service-oriented computing* (SOC) [1], modern software systems are composed by a set of loosely-coupled and possibly geographic distributed *services*. Each service implements an encapsulated, self-contained functionality and communicates via message exchange over a well-defined interface with its *partner services*. In earlier work [2] we presented a black box testing approach to test the *communication behavior* for a single service.

The development of a service is rarely finished with its initial release. In most cases, maintenance follows. This includes both, correction (i.e., fixing of discovered bugs) and integration of enhancements and new features. To adequately test the new service version, also the test suite must be revised: New test cases have to be added and existing test cases may be removed. Using the updated test suite, all (new) parts of the new version are taken into account during testing. Further, one can be sure that a failed test case indicates a bug in the implementation and is not caused by the specified changes.

In this paper, we present an idea how a test suite, generated by our approach [2], can be updated. For that purpose, unsuitable test cases are removed directly from the test suite and test cases for new functionalities are created selectively. Consequently, it is not necessary to again generate the complete test suite for a new service version. This can be useful, because there are still some small manual steps involved which do not need to be repeated for those test cases that can stay in the test suite. Especially, if only small changes are specified and most parts of the test cases can remain in the test suite the effort for test case generation can be reduced with this idea.

The rest of this paper is organized as follows: Section 2 introduces basic formalisms. Section 3 recalls the testing approach of [2]. Section 4 describes our idea to update a test suite for a new service version. In Sect. 5, open issues are discussed, before Sect. 6 concludes the paper.

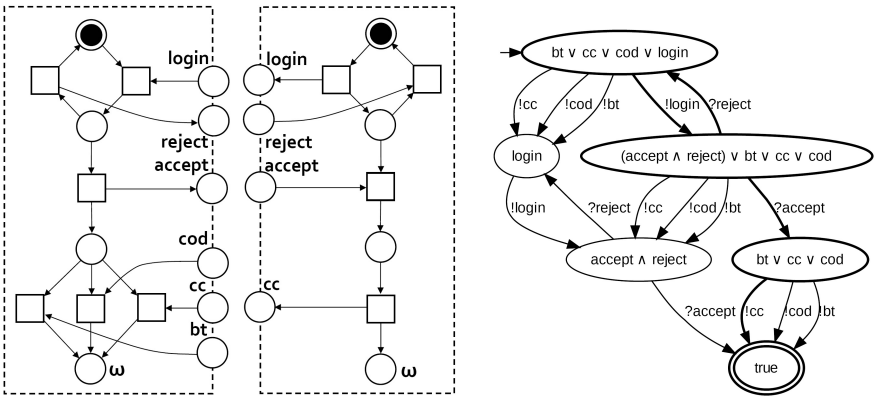
2 Formalization

Black box testing [3,4] means that test cases are created without consideration of the actual code. For an automatic generation of a test case, a formal specification is indispensable. To model the specified communication behavior of a service we use open nets [5], a class of Petri nets. Thereby, the interface is specified by special input and output places which represent the possible message types. In the model, we abstract from the content of messages and focus on sending and receiving events only. They are represented by transitions producing or consuming a token on the interface places of the open net. Furthermore, an open net has an initial and a final marking. A final marking distinguish desired end states from undesired deadlocks. To emphasize the data abstraction we also speak of an *abstract specification* S^* for a given service S.

Figure 1(a) shows the abstract specification Q^* of a simple online shop Q. The shop expects a costumer to log in and to choose one of the payment methods: cash on delivery (cod), credit card (cc), or bank transfer (bt). However, for setting the payment method the login must be accepted first, otherwise the login is rejected and the user has to retry to login. The final marking of this net is the marking $[\omega]$ which only marks the place ω: the control flow reached its end and all message channels are empty.

A *partner* P for a given service S is again a service that interacts deadlock freely with S. That means, no deadlocks (except the final state) can occur in the composition of the both services. To model the communication behavior of a partner P, we use again open nets. Since we also abstract from the data, the net models an *abstract partner* P^* of P.

As an example we consider a partner R for the service Q (cf. Fig. 1(a)). Its abstract version R^* is depicted in Fig.1(b). To check deadlock freendom between Q and R it is sufficient to compose their abstract versions Q^* and R^*. For this purpose, we only need to merge interface places with the same label. In the composed system, these merged



(a) Q^* (b) R^* (c) OG_Q

Fig. 1: The open net Q^* in (a) specifies the communication behavior of a simple online shop. In (b) an abstract partner of Q^* is shown and in (c) the operating guideline OG_Q derived from Q^* is depicted.

interfaces place become to ordinary places. In our example the composition of Q^* and R^* is free of deadlocks (except the final marking).

The set of *all* partners of a given service S can be characterized by its operating guideline OG_S [6]. Formally, an operating guideline is an *annotated service automaton*. That is a finite state automaton which edges are labeled with sending or receiving events and states are annotated with a Boolean formula. The labels determine which events of the partner may occur in a certain situation and the Boolean formulas defines which combinations thereof are allowed. Each service P belonging to the set characterized by an operating guideline OG_S fulfills the following requirements: First, the reachability graph of the abstract version P^* is a subgraph of OG_S (including an initial state). Further, each node n of the reachability graph satisfies the Boolean formula of its corresponding state in OG_S. The interested reader is referred to [6]. If a services P is characterized by an operating guideline OG_S then we also say: P *matches* with OG_S. Operating guidelines can be generated from abstract specifications using the tool Wendy[1] [7].

Figure 1(c) depicts the operating guideline OG_{Q^*} of the abstract specification Q^* (see Fig. 1(a)). In the graphical representation of operating guidelines, sending events are preceded by "!" and receiving events are preceded by "?". Initial states have an incoming arc from nowhere. Final states are double-lined. Since the reachability graph of R^* is a subgraph of OG_{Q^*} (see the bolded part of OG_{Q^*}) and the Boolean formulas of the "touched" states are fulfilled, R is characterized by OG_{Q^*}; that is, R is a partner of Q.

3 Testing Communication Behavior

To test whether the implemented service interacts correctly with its environment we create partner services as test cases. As they are derived from the specification they interact by design deadlock freely with the service under test (SUT). If, however, during testing a deadlock still occurs, we can conclude that the implementation contains (at least) one error.

The generation of test cases for a service S bases on the operating guideline OG_S. As there is usually a high number of abstract partners characterized by OG_S, it is not recommended to take every partner for testing. But among the test cases there is some redundancy, such that is possible to to select a subset without reducing the test suite's quality [2]. That means, it is still possible to find all errors, that can be found using the whole set of (abstract) partners. After we have selected the required abstract partners from the operating guideline we have to fill the message content with test data, and finally, transform them into "real" services. The latter can be done automatically by tool support, e.g. by oWFN2BPEL[2] [8] for generating a BPEL process. Data are not integrated into the concept of operating guidelines at the current state. However, data for sending messages can be generated randomly and then easily added to an abstract test case. Therefore, only the message types need to be respected. More sophisticated data have to create manually.

[1] Available at `http://service-technology.org/wendy`
[2] Available at `http://service-technology.org/owfn2bpel`

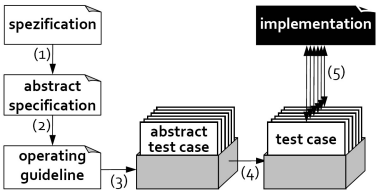


Fig. 2: A graphical overview of the general procedure for generating test cases.

Figure 2 gives again a graphical overview of the general procedure of generating test cases. First, the service specification is transformed (1) into a formal model. Based on this abstract specification the operating guideline is constructed (2) and a set of abstract partners is selected (3). To achieve executable test cases, the abstract partners are enriched with data information and retranslated into real services (4). The actual test procedure (5) can be finally sketched as follows: The service to be tested is deployed in a testing environment together with the test suite. To process the test suite, the contained test cases are executed one after the other. Thereby, each test case interacts as designed with the service under test. The testing environment then is responsible for logging and evaluating exchanged messages. We thereby assume the test environment is able to detect whether the implementation terminates properly; for instance, whether a BPEL process instance has completed successfully. Except the generation of test data in (4), all steps can be executed automatically.

4 Adjust existing test suites to new specifications

As already mentioned in the introduction, it can be required to modify an existing service from time to time. By introducing enhancements and new features the specification is changed. This entails existing test cases to become invalid. That makes the test suite inconclusive. Further, after adding new functionalities, there are parts in the new version of the service which are not taken into account by the old test suite.

To ensure a thorough testing, the test suite needs to be revised. Instead of discarding all old test cases and deriving a new test suite from the scratch, we aim to keep as much old test cases as possible. This is desirable because the procedure for generating the test case (see Sect. 3) is not fully automatic. Thus, this strategy helps to reduce the manual effort. Further, it is reasonable to use the same data in the remaining test cases. By generating test cases from scratch, these data information would be lost.

First, we describe in the following how invalid test cases can be removed from the test suite. Afterward, we suggest how the new test cases can be found.

Making the test suite conclusive. Integrating new functionalities can change the control flow and with it the communication behavior such that existing test cases become invalid. This is demonstrated by the following example. Figure 3 shows a modification Q^*_{new} for the specification in Fig. 1(a): If the credit card (cc) is chosen as payment method, a voucher is sent to the customer. Now, the composition of Q^*_{new} and R^* (see Fig. 1(b)) contains a deadlock because the voucher cannot be received from R^*. When testing the new version of the simple online shop with R the test fails even through the

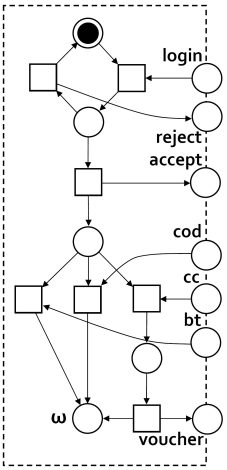


Fig. 3: An modified specification Q^*_{new} for the simple online shop, cf. Fig.1(a).

implementation is correct. In converse, if the implementation is incorrect the test can succeed.

To achieve a conclusive test suite for the new specification we discard the invalid test cases. They can be detected using the matching algorithm for operating guidelines (cf. Sect. 2). For each test case we check whether it matches with the operating guideline of the *new* specification. This can be done using the tool Cosme[3]. Non-matching test cases are invalid and have to be removed. After this procedure we have a conclusive test suite.

Adding missing test cases. To cover new parts during testing, test cases have to be added to the test suite. For a systematic approach we make the following consideration: Let OG_S be the operating guideline of a specification and $OG_{S_{new}}$ the operating guideline of the new specification. Let T_S be the set of test cases characterized of OG_S and $T_{S_{new}}$ be the set of the test cases characterized by $OG_{S_{new}}$. When calculating the difference $T_{S_{new}} \setminus T_S$ we get exactly the set of test cases for testing the new parts.

Unfortunately, the result of the difference operation cannot be represented as an operating guideline. Instead, it can be represented as an *extended annotated service automaton* (EAA for short) [9]. This kind of finite automata extends annotated service automata (cf. Sect. 2) by a global Boolean formula. It constrains the combinations of states in the EAA that have to be "touched" during matching. Further, the structure of the Boolean formulas in the states is not restricted as much as in an operating guideline.

Formally, an operating guideline can easily be transformed into an EAA by setting the global Boolean formula to $true$; that is, there are no additional requirements for the states. For EAAs all basic set operations (including the difference) are defined [9] and supported by the tool Safira[4] [10].

Thus, we are able to calculate the set $T_{S_{new}} \setminus T_S$ based from the corresponding operating guidelines. The result is an EAA that characterizes the test cases to be added.

[3] Available at `http://service-technology.org/cosme`
[4] Available at `http://service-technology.org/safira`

In the next step, the (abstract) test cases needs to extract from the EAA and transformed into executable test cases.

5 Future Work

The set of test cases to be added, can be only represented by EAAs, an extension of operating guidelines. In contrast to operating guidelines, the extraction of test cases from an EAA is a time-consuming task. In [9] we proved that this problem is NP-complete in general. This is caused by the more complex Boolean formula (in comparison to the operating guidelines). In future work, we intend to find a good heuristic such that the selection is practicable for EAAs. Thereby, it can be useful that the EAA is resulted by applying the difference operation to two operating guidelines. Thus, the boolean formulas in the result are less complex, than it could be in an arbitrary EAA.

6 Conclusion

In this paper, we presented a concept to manage test suites for services in an iterative development process. With the contained test cases the communication behavior can be tested thoroughly. We demonstrated how invalid test cases can be detected and removed from the test suite and we principle show how new test cases can be added. Thereby, most steps are supported by existing tools.

References

1. Papazoglou, M.: Agent-oriented technology in support of e-business. Commun. ACM **44**(4) (2001) 71–77
2. Kaschner, K., Lohmann, N.: Automatic test case generation for interacting services. In: Service-Oriented Computing 2008 Workshops. LNCS 5472, Springer (2009) 66–78
3. Beizer, B.: Black-box testing: techniques for functional testing of software and systems. John Wiley & Sons, Inc., New York, NY, USA (1995)
4. Black, R.: Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing. Wiley Publishing (2009)
5. Wolf, K.: Does my service have partners? LNCS T. Petri Nets and Other Models of Concurrency **5460**(2) (2009) 152–171
6. Lohmann, N., Massuthe, P., Wolf, K.: Operating guidelines for finite-state services. In: ICATPN 2007. LNCS 4546, Springer (2007) 321–341
7. Lohmann, N., Weinberg, D.: Wendy: A tool to synthesize partners for services. In: PETRI NETS 2010. LNCS 6128, Springer-Verlag (2010) 297–307
8. Lohmann, N., Kleine, J.: Fully-automatic translation of open workflow net models into simple abstract BPEL processes. In: Modellierung 2008. Volume P-127 of Lecture Notes in Informatics (LNI)., GI (2008) 57–72
9. Kaschner, K., Wolf, K.: Set algebra for service behavior: Applications and constructions. In: BPM 2009. LNCS 5701, Springer-Verlag (2009) 193–210
10. Kaschner, K.: Safira: implementing set algebra for service behaviour. In: Proceedings of the 2nd Central-European Workshop on Services and their Composition. CEUR Workshop Proceedings, CEUR-WS.org (2010) 49–56

The Petri Net API

A collection of Petri net-related functions

Niels Lohmann, Stephan Mennicke, and Christian Sura

Universität Rostock, Institut für Informatik, 18051 Rostock, Germany
pnapi@service-technology.org

Abstract. This paper introduces the Petri Net API, a C++ library of Petri net-related functions. The Petri Net API is currently used in more than a dozen Petri net tools, ranging from compilers to verification tools.

1 Introduction

Algorithms to reason about the correctness of distributed systems usually have devastating worst-case complexities. Nevertheless, experiences in the field of model checking show that the feared state space explosion can be alleviated by state space reduction techniques or symbolic representations [5]. Therefore, novel techniques to verify correctness often require a proof-of-concept implementation to conduct experiments or to demonstrate feasibility on realistic input data. We recently investigated the academic software development process [14] and claimed that single purpose tools have the right granularity to be implemented using rapid prototyping techniques. A prerequisite for this is the encapsulation of frequently recurring functionality into reusable libraries to avoid an unnecessary "reinvention of the wheel" and to minimize the amount of untested ad-hoc code.

This paper introduces with the Petri Net API such a reusable library. It was originally derived from the back-end of the compiler BPEL2oWFN [12] and offered simple net management functionality and the file output in several formats. In the last years, the functions of the Petri Net API have been revised and collected into a consistent C++ library. The main focus of the API is to *organize* Petri nets, rather than to implement verification algorithms (i.e., to build and analyze state spaces) or to provide a graphical front-end. We claim that these tasks should be part of a dedicated tool rather than a library that is designed to be shared by several tools. As of September 2010, the Petri Net API is used by 14 tools, see `http://service-technology.org/tools` for an overview.

The rest of this paper is organized as follows. The next section presents the features that are implemented by the Petri Net API. Then Sect. 3 shows how the Petri Net API can be used in novel tools, discusses its availability, and the integration in third-party tools. Section 4 presents a small case study in which the Petri Net API is used to perform some structural modifications to check a correctness notion for workflow nets. Section 5 briefly compares the Petri Net API to existing frameworks, before Sect. 6 concludes the paper.

2 Features

The Petri Net API implements the following features to organize Petri nets.

- Petri net creation and manipulation (creation, modification, deletion, copying, and search of nodes and arcs),
- file import and export of Petri nets in various file formats (PNML [9], LoLA [22] file format, Fiona [15] open net format, Woflan [21] workflow nets),
- generation of graphical representation using Graphviz dot,
- efficient application of structural reduction rules [17,19,18],
- structural checks (e.g., workflow net structure, free-choice property),
- import from automata (using the region theory tools Petrify [6] or Genet [3]),
- export to automata,
- support for open nets [23] (ports, net composition)
- organization of final markings, and
- support for role annotations.

The Petri Net API can be easily extended to new features. As of now, we only moved features from a tool into the Petri Net API when this feature is used by at least one other tool. This avoids a cluttered API full of too specific functions. At the same time, it ensures a high test case coverage of the features.

3 Usage

Integration as C++ library. The API itself is written in C++ and can be integrated into other tools with no more effort than including a header file. Listing 1.1 shows example code to read a file in PNML format, structurally reduce it using the Murata rules [17], and output a graphical representation.

Listing 1.1. Example using the Petri Net API as C++ library.

```
#include <pnapi/pnapi.h>
using namespace pnapi;

int main() {
  // read PNML net from file
  std::istream in("file.pnml", std::ios_base::in);
  in >> io::pnml >> net;

  // apply reduction rules
  net.reduce(PetriNet::SET_MURATA);

  // output the Petri net in Graphviz dot format
  std::cout << io::dot << net;

  return 0;
}
```

The Petri Net API complies with the 1998 ANSI/ISO C++ standard and can be compiled on many platforms, including Microsoft Windows, Sun Solaris, GNU/Linux, Mac OS X, and other UNIX-based operating systems.

Command-line tool. Beside the direct integration, we implemented a command-line tool `petri` (part of the Petri Net API distribution) for the most common operations. The call

```
petri *.pnml --input=pnml --reduce=murata --output=png
```

performs a similar transformation as the code in Listing 1.1, but is also able to read multiple files and to directly create graphics files. This front-end tool is very useful in shell scripts to process large libraries of nets.

Integration into third-party tools. The Petri Net API is currently indirectly (as front-end tool or library) integrated into the business process modeling tool Oryx [7], the process mining toolkit ProM [20], and the YAWL workflow editor [2]. In all tools, the Petri Net API organizes the exchange of Petri Net models in PNML format.

3.1 Availability

The Petri Net API is free open source software, licensed under the GNU LGPL 3.[1] The most recent version together with its manual can be downloaded at `http://service-technology.org/pnapi`.

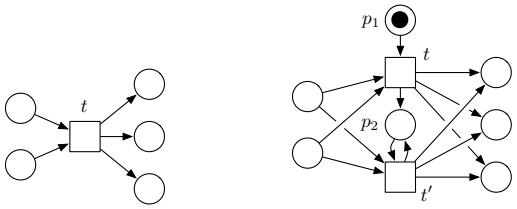
4 Case study: Checking relaxed soundness

To demonstrate the usage of the Petri net API in a realistic setting, we discuss a small case study in this section. We show how *relaxed soundness* [8], a correctness property of workflow nets [1], can be translated into a series of reachability problems that can be checked by LoLA [22].

Relaxed soundness requires for every transition of the workflow net to occur in at least one successful firing sequence from the initial marking $[i]$ to the final marking $[o]$. Dehnert and Aalst [8] provided a verification algorithm for this property in which builds the state space of the workflow net and then analyzes its runs. This algorithm is implemented in the tool Woflan [21]. It is, however, not clear whether state space reduction techniques are applicable.

Alternatively, we can reformulate the above requirement in a reachability problem as follows. Given a workflow net N and a transition t of N, we can construct a Petri net N_t which only reaches a final marking iff N reaches a final marking by a transition sequence which includes t. We create N_t by adding to N a transition t' and two places p_1 and p_2. Figure 1 illustrates the construction. Thereby, p_1 is marked as long t has not yet fired, and p_2 is marked after t has fired at least once. We then can check wether the marking $[o, p_2]$ is reachable from the initial marking $[i, p_1]$. If this check succeeds for all nets N_t, we can conclude relaxed soundness of N.

[1] GNU Lesser General Public License, `http://www.gnu.org/licenses/lgpl.html`



(a) subnet of N (b) subnet of N_t

Fig. 1. Transforming relaxed soundness into a reachability problem.

Listing 1.2. Construction of N_t using the Petri Net API.

```
#include <pnapi/pnapi.h>
using namespace pnapi;

void constructAnalysisNets(PetriNet &N) {
  // iterate transitions
  PNAPI_FOREACH(trans, N.getTransitions()) {
    // create copy of the net
    PetriNet Nt(N);

    // create analysis subnet for current transition (see Fig. 1)
    Place &p1 = Nt.createPlace();
    Place &p2 = Nt.createPlace();
    Transition *t = Nt.findTransition((*trans)->getName());
    Transition &tprime = Nt.createTransition();

    // connect analysis subnet
    PNAPI_FOREACH(p, t->getPreset()) {
      Nt.createArc(**p, tprime);
    }
    PNAPI_FOREACH(p, t->getPostset()) {
      Nt.createArc(tprime, **p);
    }
    p1.setTokenCount(1);
    Nt.createArc(p1, *t);
    Nt.createArc(*t, p2);
    Nt.createArc(p2, tprime);
    Nt.createArc(tprime, p2);

    // write nets into LoLA files
    std::ofstream o;
    o.open("N_" + (*trans)->getName() + ".lola", std::ios_base::trunc);
    o << pnapi::io::lola << Nt << std::endl;
    o << "FORMULA (" << pnapi::io::lola << Nt.getFinalCondition()
      << " AND " << p2.getName() << " = 1 )" << std::endl;
  }
}
```

Listing 1.2 shows a function implementing this construction. It assumes a Petri net N is given and writes, for each transition t of N, a Petri net N_t in LoLA file format together with a formula expressing the final marking to disk.

The implementation is straightforward and is — due the encapsulation of the Petri net functions — nearly on a pseudocode level. The Petri net model checker LoLA can read these generated files and check whether the included formula can

be satisfied by a reachable marking. The described transformation and check is implemented as service-oriented extension of the business process editor Oryx [7].

5 Related work

The idea to collect Petri net-related functions in a library or toolbox is not new (see [16,4] and the Petri Net World Website[2] listing hundreds of tools). We discuss two prominent examples.

The *Petri Net Kernel* [11] was designed as a modular kernel that is designed to integrate Petri net algorithms. Petri net types are not fixed, but can be freely defined and extended. Similarly, the *PNML framework* [10] is a reference implementation of the PNML standard. It is designed to facilitate import and export of PNML standard compliant files and provides a complex meta model to support different kinds of Petri nets. Again, its focus lies on flexibility.

In contrast, the Petri net API has a fixed feature set and new features are only added when they are also used by other tools. Furthermore, it was not originally designed as a generic Petri net framework, but was created by "outsourcing" Petri net-related code from actual tools. Finally, it is implemented in C++ due to performance considerations.

6 Conclusion

This paper introduced the Petri Net API, a library of Petri net-related functions. We observed that the Petri Net API greatly simplified rapid prototyping. The encapsulation of Petri net-related functions lead to smaller tools which could focus on their main functionality; see [13] for a discussion. After four years of development and a consolidated feature set, we claim that this API is useful to other researchers in the Petri net community. The main advantage of the Petri Net API is that its implemented functions are heavily used for several years and thus has a well-tested and justified feature set.

Acknowledgments. The authors thank Christian Gierds, Dennis Reinert, Georg Straube, Robert Waltemath, and Martin Znamirowski for their work on earlier versions of the Petri Net API.

References

1. Aalst, W.M.P.v.d.: The application of Petri nets to workflow management. Journal of Circuits, Systems and Computers 8(1), 21–66 (1998)
2. Aalst, W.M.P.v.d., Hofstede, A.H.M.t.: YAWL: yet another workflow language. Inf. Syst. 30(4), 245–275 (2005)

[2] `http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/`

3. Carmona, J., Cortadella, J., Kishinevsky, M.: Genet: a tool for the synthesis and mining of Petri nets. In: ACSD 2009. pp. 181–185. IEEE (2009)
4. Chouikha, A., Fay, A., Schnieder, E.: Konzept eines Frame- works für Petrinetz-Editoren. In: AWPN 1998. pp. 32–37. No. 694 in Research Reports, Universität Dortmund, Fachbereich Informatik (1998)
5. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)
6. Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers. Trans. Inf. and Syst. E80-D(3), 315–325 (1997)
7. Decker, G., Overdick, H., Weske, M.: Oryx - an open modeling platform for the bpm community. In: BPM 2008. pp. 382–385. LNCS 5240, Springer (2008)
8. Dehnert, J., Aalst, W.M.P.v.d.: Bridging the gap between business models and workflow specifications. Int. J. Cooperative Inf. Syst. 13(3), 289–332 (2004)
9. Hillah, L.M., Kindler, E., Kordon, F., Petrucci, L., Trèves, N.: A primer on the Petri Net Markup Language and ISO/IEC 15909-2. Petri Net Newsletter 76, 9–28 (2009)
10. Hillah, L.M., Kordon, F., Petrucci, L., Trèves, N.: PNML framework: An extendable reference implementation of the Petri Net Markup Language. In: PETRI NETS 2010. pp. 318–327. LNCS 6128, Springer (2010)
11. Kindler, E., Weber, M.: The Petri Net Kernel - an infrastructure for building Petri net tools. STTT 3(4), 486–497 (2001)
12. Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0 and its compiler BPEL2oWFN. Informatik-Berichte 212, Humboldt-Universität zu Berlin, Berlin, Germany (2007)
13. Lohmann, N., Weinberg, D.: Wendy: A tool to synthesize partners for services. In: PETRI NETS 2010. pp. 297–307. LNCS 6128, Springer (2010)
14. Lohmann, N., Wolf, K.: How to implement a theory of correctness in the area of business processes and services. In: BPM 2010. pp. 61–77. LNCS 6336, Springer (2010)
15. Massuthe, P., Weinberg, D.: Fiona: A tool to analyze interacting open nets. In: AWPN 2008. pp. 99–104. CEUR Workshop Proceedings Vol. 380, CEUR-WS.org (2008)
16. Menzel, T.: Entwurf und Implementierung eines objektorientierten Frameworks zur Petrinetzbasierten Modellierung. In: AWPN 1997. pp. 25–30. No. 85 in Informatik-Berichte, Humboldt-Universität zu Berlin, Institut für Informatik (1997)
17. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
18. Pillat, T.: Gegenüberstellung struktureller Reduktionstechniken für Petrinetze. Diplomarbeit, Humboldt-Universität zu Berlin (2008), (in German)
19. Starke, P.H.: Analyse von Petri-Netz-Modellen. Teubner Verlag (1990)
20. Verbeek, E., Buijs, J., Dongen, B.v., Aalst, W.M.P.v.d.: Prom 6: The process mining toolkit. In: BPM 2010 Demos. CEUR Workshop Proceedings Vol. 615, CEUR-WS.org (2010)
21. Verbeek, H.M.W., Basten, T., Aalst, W.M.P.v.d.: Diagnosing workflow processes using Woflan. Comput. J. 44(4), 246–279 (2001)
22. Wolf, K.: Generating Petri net state spaces. In: ICATPN 2007. pp. 29–42. LNCS 4546, Springer (2007)
23. Wolf, K.: Does my service have partners? LNCS T. Petri Nets and Other Models of Concurrency 5460(2), 152–171 (2009)

Partner datenverarbeitender Services

Christoph Wagner

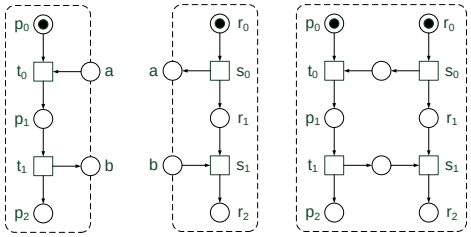
Institut für Informatik, Humboldt Universität zu Berlin,
Unter den Linden 6, 10099 Berlin, Germany
cwagner@informatik.hu-berlin.de

Zusammenfassung Offene Netze sind eine petrinetzbasierte formale Beschreibung von Services. Zwei offene Netze sind füreinander Partner, wenn ihre Komposition aus jedem erreichbaren Zustand einen Endzustands erreichen kann. Dieser Beitrag erweitert das Konzept der offenen Netze auf High-Level Petrinetze, um die Verarbeitung von Daten in Services darzustellen. Es werden exemplarisch Partner offener High-Level Netze und die in ihnen vorkommenden Ausdrucksmittel untersucht.

1 Einleitung

Service Oriented Computing (SOC) beschreibt ein Paradigma zum Aufbau verteilter Systeme aus Services. Ein *Service* ist eine eigenständige funktionale Einheit, die durch asynchrone Nachrichtenkanäle mit anderen Services kommunizieren kann. *Offene Netze* [2,3] sind ein Mittel zur formalen Beschreibung von Services. Mit offenen Netzen lassen sich Kommunikation und Komposition von Services auf natürliche Weise im Petrinetzkalkül ausdrücken.

Ein offenes Netz ist ein Petrinetz mit speziell ausgezeichneten *Sende-* und *Empfangsplätzen*, welche Puffer für asynchrone Nachrichtenkanäle repräsentieren. Abb. 1 zeigt ein offenes Netz N_1 mit einem Empfangsplatz a und einem Sendeplatz b. Wir verzichten im Folgenden auf eine formale Definition und verweisen für technische Details auf [1,2]. Zwei offene Netze N_1 und N_2 heißen (syntaktisch) *kompatibel*, wenn jeder



(a) N_1 (b) N_2 (c) $N_1 \oplus N_2$

Abbildung 1: Kompatible offene Netze und ihre Komposition

Sendeplatz eines Netzes ein Empfangsplatz des jeweils anderen Netzes ist. Kompatible offene Netze werden *komponiert*, indem gleichnamige Empfangs- und Sendeplätze

verschmolzen werden. Die verschmolzenen Plätze werden dabei zu internen Plätzen der Komposition $N_1 \oplus N_2$. Die Komposition ist ein *geschlossenes Netz* ohne Sende- und Empfangsplätze.

Für ein einzelnes offenes Netz stellt sich die Frage nach seiner *Bedienbarkeit*: Gegeben ein offenes Netz N, gibt es ein kompatibles offenes Netz S, so dass die Komposition $N \oplus S$ aus jedem erreichbaren Zustand einen ausgezeichneten Endzustand erreichen kann? Eine Ursache von Nichtbedienbarkeit können in der Komposition erreichbare Deadlocks sein. Bedienbarkeit ist ein Grundproblem der Controllersynthese und wurde im Kontext von offenen Netzen z. B. in [2,3] untersucht. Der den Analyseverfahren zugrunde liegende Formalismus berücksichtigt die in den Nachrichten enthaltenen Datenwerte bisher nicht explizit. Offene Netze, die große oder gar unendlich große Domänen verwenden, können mit rechnergestützten Methoden nur schwer oder gar nicht analysiert werden.

Dieser Beitrag verfolgt den Ansatz, Datenwerte explizit in den Formalismus der offenen Netze mit einzubeziehen. Dafür erweitern wir das Konzept der offenen Netze auf High-Level Petrinetze. Datenwerte werden in einem High-Level Petrinetz durch farbige Marken dargestellt. Da die im Folgenden behandelten Probleme weitgehend unabhängig vom verwendeten High-Level Petrinetzformalismus sind, soll die Definition hier nur kurz skizziert werden:

Ein High-Level Petrinetz besteht aus einer Menge von Plätzen P, einer Menge von Transitionen T, einer Menge F von Kanten und einer Anfangsmarkierung m_0. Jedem Platz p ist eine eine Menge von Werten (seine *Domäne*) $dom(p)$ zugeordnet. Jeder Transition t ist ein *Guard* $g(t)$ und eine Menge $var(t)$ von *Schaltvariablen* zugeordnet, wobei jede Variable x ebenfalls eine Domäne $dom(x)$ hat. Die Kanten sind mit Termen über den Schaltvariablen beschriftet. Eine *Markierung* m ist eine Funktion, die jedem Platz p eine Multimenge von Elementen aus $dom(p)$ zuweist. Eine Funktion β, die jede Variable $x \in var(t)$ einer Transition t mit einem Element $e \in dom(x)$ belegt, heißt *Schaltmodus* von t. Beim Schalten einer Transition in Modus β werden die Kantenbeschriftungen mit β ausgewertet und entsprechend Marken von Vorplätzen konsumiert bzw. auf den Nachplätzen produziert. Voraussetzung zum Schalten ist, dass der Guard für β zu $true$ auswertet.

Der folgende Abschnitt erläutert zunächst den Begriff des *Partners* und zeigt exemplarisch einige Partner von offenen High-Level Petrinetzen. Es wird untersucht, wie sich die Verwendung von Variablen in einem offenen Netz auf die Struktur seiner Partner auswirkt. Es zeigt sich, dass sich selbst bei Verzicht auf Ausdrucksmittel wie Guards, Funktionssymbole und Konstanten nicht-triviale Abhängigkeiten ergeben, die ein Partner berücksichtigen muss.

2 Partner eines datenverarbeitenden Services

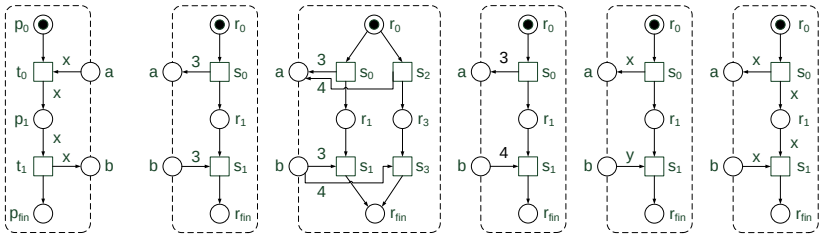
Wir betrachten im Folgenden die Partner von offenen High-Level Netzen. Zwei offene Netze mit jeweils ausgezeichneten Mengen von *Endmarkierungen* nennen wir *Partner*, wenn ihre Komposition *schwach terminiert* (vergleiche hierzu Def. 4 in [2]).

Definition 1 (Partner). *Ein (geschlossenes) Petrinetz* terminiert schwach*, wenn von jeder erreichbaren Markierung aus eine Endmarkierung des Netzes erreichbar ist. Seien*

N, S kompatible offene Netze. Wir nennen S einen Partner *von N, wenn die Komposition $N \oplus S$ schwach terminiert. Die Menge der Partner von N notieren wir mit $Partner(N)$.*

Die Endmarkierungen der Komposition ergeben sich kanonisch aus den Endmarkierungen der komponierten offenen Netze. Man beachte, dass $N \oplus S$ nicht schwach terminiert, wenn $N \oplus S$ einen Deadlock enthält. Das folgende Beispiel dient zunächst der Erläuterung des Partnerbegriffs. Die einzige Endmarkierung der offenen Netze bestehe jeweils aus einer Marke auf dem Platz p_{fin} bzw. r_{fin}. Die Domäne jedes Platzes sei entweder $\mathbb{N}$ oder $\{\bullet\}$, wobei $\bullet$ das Symbol für eine Marke ist, die keinen Wert trägt. Jede Variable habe die Domäne $\mathbb{N}$.

Beispiel 1 Das offene Netz N aus Abb. 2 setzt eine Nachrichtenweiterleitung um. Die von N auf a (durch Schalten von t_0) empfangene Nachricht wird unverändert (durch Schalten von t_1) auf b zurückgeschickt. Das offene Netz S_1 ist ein Partner von N. S_1



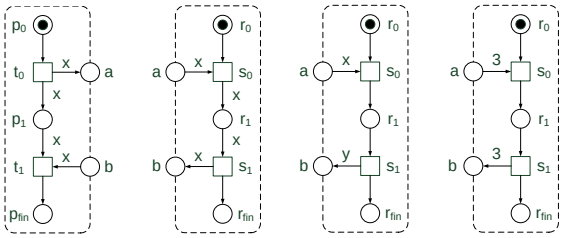
(a) N (b) S_1 (c) S_2 (d) S_3 (e) S_4 (f) S_5

Abbildung 2: Ein offenes Netz N und kompatible offene Netze

sendet zunächst durch Schalten von s_0 eine Nachricht auf a mit dem Wert 3. Da N auf b den selben Wert 3 sendet, kann auch s_1 schalten. Beide offene Netze erreichen somit ihre Endmarkierungen $[p_{fin}]$ und $[r_{fin}]$. Daher terminiert $N \oplus S_1$ schwach. Das offene Netz S_2 ist ebenfalls ein Partner von N. Es kann alternativ zum Wert 3 auch den Wert 4 senden und und diesen anschließend empfangen. Das offene Netz S_3 ist dagegen kein Partner von N. Es sendet den Wert 3 auf a, kann auf b aber nur Nachrichten mit Wert 4 empfangen. Da N auf b jedoch den Wert 3 sendet, kann s_1 nicht schalten und die Endmarkierung $[r_{fin}]$ von S_3 wird nicht erreicht. Das offene Netz S_4 ist ein Partner von N. Es wählt zunächst nichtdeterministisch eine natürliche Zahl als Belegung für x und sendet diese auf a. Anschließend empfängt es einen beliebigen Wert auf b (unabhängig davon, ob dieser gleich dem gesendeten ist) und wechselt in seine Endmarkierung $[r_{fin}]$. S_5 verhält sich auf die gleiche Weise, verlangt jedoch, dass der auf b empfangene Wert gleich dem auf a gesendeten Wert sein *muss*. S_5 ist in diesem Sinne präziser als S_4: Beide offene Netze sind gleichermaßen Partner von N, jedoch lässt die Struktur von S_5 einen genaueren Rückschluss auf die Funktionsweise von N zu. Aus S_4 kann man nicht erkennen, dass der von N auf b gesendete Wert stets gleich dem von N auf a empfangenen Wert ist.

Das nächste Beispiel zeigt ein für High-Level Netze spezifisches Phänomen auf.

Beispiel 2 Das offene Netz N' in Abb. 3 hat die gleiche Struktur wie S_2 aus Abb. 2. Es ähnelt dem offenen Netz N aus Abb. 2, jedoch sind Eingabe und Ausgabe vertauscht. Trotz dieses scheinbar geringen Unterschieds verfügt N' über zwei entscheidende Ausdrucksmittel, über die N nicht verfügt:



(a) N' (b) S'_1 (c) S'_2 (d) S'_3

Abbildung 3: Ein offenes Netz N' und kompatible offene Netze

1. t_0 ist eine *erzeugende* Transition, d. h. sie führt einen neuen Wert in das Netz ein. Die Variable x ist beim Schalten von t_0 nicht gebunden an den Wert einer Marke auf einem Eingangsplatz. Vielmehr wird x nichtdeterministisch an einen beliebigen Wert seiner Domäne $dom(x)$ gebunden. Im Unterschied zu N, wo t_0 einen auf a bereits vorhandenen Wert auf p_1 lediglich verschiebt, wird in N' auf den Plätzen p_1 und a ein Wert neu *erzeugt*.
2. t_1 führt einen *Test auf Gleichheit* aus. t_1 kann nur dann schalten, wenn auf p_1 und b Marken mit gleichen Werten liegen. Somit bestimmt der von der Umgebung auf b gesendete Wert direkt, ob N' seine Endmarkierung $[p_{fin}]$ erreichen kann oder nicht.

Über einen Partner von N' können wir folgern: Der vom Partner auf b gesendete Wert darf nicht unabhängig sein vom auf a empfangenen Wert. Vielmehr muss ein Partner Sorge dafür tragen, dass der von ihm auf b gesendete Wert gleich dem von ihm auf a empfangenen ist. Aus diesem Grund ist S'_1 ein Partner von N', S'_2 jedoch nicht. S'_3 berücksichtigt im Unterschied zu S'_2 die Gleichheit von empfangenem und gesendeten Wert, deckt aber nur einen einzigen Wert aus $dom(x)$ ab. Daher ist s_0 für die meisten auf a empfangbaren Werte nicht aktiviert und S'_3 infolgedessen kein Partner von N'.

Da $dom(x) = \mathbb{N}$ eine unendlich große Menge ist, muss jeder Partner ebenfalls über ein Ausdrucksmittel verfügen, das einen unendlich großen Wertebereich abdeckt. Diese Folgerung formulieren wir wie folgt:

Vermutung 1. Jeder Partner von N' enthält eine Kante, die mit einer Variablen beschriftet ist, deren Wertebereich unendlich groß ist.

Eine weitere entscheidende Beobachtung ist, dass ein Partner offenbar gewisse *Abhängigkeiten* zwischen gesendeten und empfangenen Werten einhalten muss. Augenscheinlicher Verursacher dieses Phänomens ist die Transition t_1. Diese ist eine *datenabhängige*

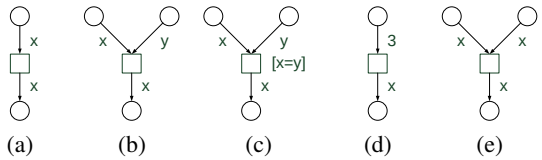
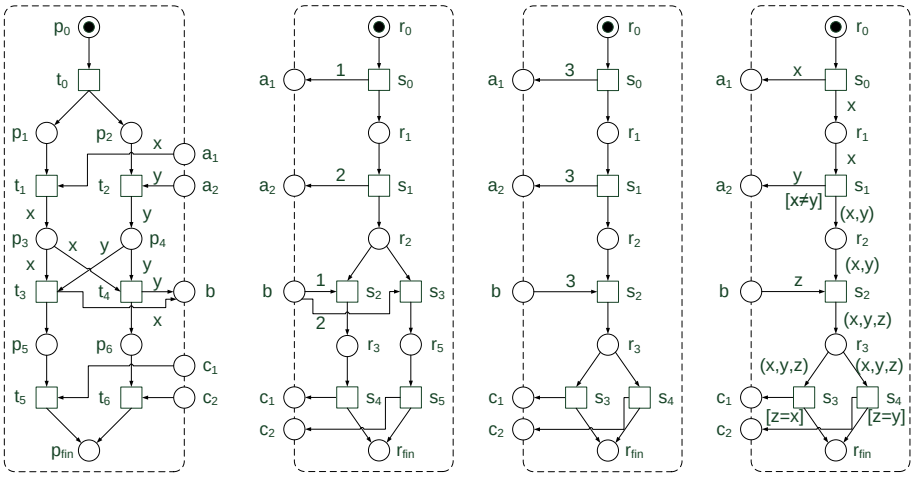


Abbildung 4: Datenunabhängige Transitionen: (a) - (b). Datenabhängige Transitionen: (c) - (e).

Transition. Wir nennen eine Transition datenabhängig, wenn die Entscheidung, ob sie schalten darf, von den Werten der Marken auf ihren Vorplätzen abhängt. Umgekehrt ist eine Transition *datenunabhängig*, wenn die Entscheidung nur von der Anzahl der Marken auf den Vorplätzen abhängt. Die Werte der produzierten Marken dürfen dagegen sehr wohl von den Werten der konsumierten Marken abhängen. Abb. 4 zeigt einige datenabhängige und -unabhängige Transitionen. Transitionen mit Guards sind im allgemeinen datenabhängig. Es ist offensichtlich, dass eine datenunabhängige Transition genau dan schalten kann, wenn sie im Skelett des Netzes schalten kann.

Das nächste Beispiel zeigt, dass jedoch auch Partner, die keine datenabhängige Transition enthalten, Abhängigkeiten zwischen Datenwerten berücksichtigen müssen. Auch dies steht, wie im vorangehenden Beispiel, in engem Zusammenhang zur Werterzeugung.



(a) N'' (b) $S_1'' \in Partner(N'')$ (c) $S_2'' \notin Partner(N'')$ (d) $S_3'' \in Partner(N'')$

Abbildung 5: Ein offenes Netz N'' und kompatible offene Netze

Beispiel 3 Das Netz N'' aus Abb. 5 empfängt zunächst zwei Werte auf a_1 und a_2 und trifft anschließend eine nichtdeterministische Entscheidung: Es wird eine Marke entwe-

der (a) auf p_5 oder (b) auf p_6 produziert. Diese Wahl wird der Umgebung kommuniziert, indem entweder (a) der auf a_1 empfangene Wert oder (b) der auf a_2 empfangene Wert auf b gesendet wird. N'' erwartet anschließend (a) eine Nachricht auf c_1 oder (b) eine Nachricht auf c_2. Ein Partner von N'' darf niemals zwei gleiche Werte auf a_1 und a_2 senden. In diesem Fall sind die beiden Fälle (a) und (b) anhand der auf b empfangenen Nachricht nicht unterscheidbar und der Partner kann nicht wissen, ob N'' eine Antwort auf c_1 oder c_2 erwartet. Der Guard $[x \neq y]$ an Transition s_1 in S_3'' ist also zwingend erforderlich, um zu verhindern, dass N'' möglicherweise einen Deadlock erreicht. Aufgrund des Guards ist s_1 eine datenabhängige Transition. Ebenfalls datenabhängig sind jeweils die Transitionen s_2 und s_3 in S_1'' sowie s_3, s_4 in S_3'', welche eine Fallunterscheidung abhängig vom auf b empfangenen Wert treffen. Wir gelangen zu der folgenden Vermutung:

Vermutung 2. Jeder Partner von N'' enthält mindestens eine datenabhängige Transition.

Diese Folgerung ist bemerkenswert, da N'' selbst weder datenabhängige noch erzeugende Transitionen enthält. Die Partner verwenden also Ausdrucksmittel, die in N'' nicht vorkommen: S_1'' verwendet Konstanten, S_3'' einen Guard. N'' verwendet weder Konstanten noch Guards.

3 Schlussfolgerungen und Ausblick

Die untersuchten Beispiele zeigen, dass selbst bei beschränkten Ausdrucksmitteln (nur Variablen, keine Guards) vergleichsweise komplexe Anforderungen an Partner eines offenen Netzes entstehen. Insbesondere zeigt Beispiel 3, dass jeder Partner eines offenen Netzes unter Umständen Ausdrucksmittel verwendet, die im offenen Netz selbst nicht vorkommen. Dies ist vor allem für das Problem der Partnersynthese wichtig. Offen ist, welche Typen von Transitionen hinreichen, um jeden Partner eines offenen Netzes, dass keine datenabhängigen Transitionen enthält, zu beschreiben. Ebenfalls offen ist, welche Techniken zum Beweis der aufgestellten Vermutungen notwendig sind.

Literatur

1. Lohmann, N., Massuthe, P., Wolf, K.: Operating guidelines for finite-state services. In: Kleijn, J., Yakovlev, A. (eds.) 28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, ICATPN 2007, Siedlce, Poland, June 25-29, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4546, pp. 321–341. Springer-Verlag (2007)
2. Massuthe, P., Serebrenik, A., Sidorova, N., Wolf, K.: Can I find a partner? Undecidablity of partner existence for open nets. Information Processing Letters 108(6), 374–378 (November 2008)
3. Weinberg, D.: Efficient controllability analysis of open nets. In: Bruni, R., Wolf, K. (eds.) Web Services and Formal Methods, Fifth International Workshop, WS-FM 2008, Milan, Italy, September 4–5, 2008, Proceedings. Lecture Notes in Computer Science, Springer-Verlag (September 2008)

Autorenverzeichnis

Blätke, Mary Ann, 42

Cabac, Lawrence, 94

Denz, Nicolas, 94

Flick, Nils Erik, 94

Gierds, Christian, 136

Heiner, Monika, 51, 66
Herajy, Mostafa, 66

Kaschner, Katrin, 142

Liu, Fei, 51
Lohmann, Niels, 136, 148

Müller, Richard, 130
Markwardt, Kolja, 100
Marvan, Wolfgang, 42
Mennicke, Stephan, 148
Moldt, Daniel, 94, 100, 106

Prüfer, Robert, 16

Rohr, Christian, 88

Schneider, Christoph, 22
Schwarick, Martin, 80
Simon, Jochen, 106
Sura, Christian, 148

Wagner, Christoph, 154
Wehler, Joachim, 22
Wimmel, Harro, 112, 118
Wolf, Karsten, 118, 124

Zaitsev, Dmitry, 1