Pi-calculus-based Resource-access Analysis of High-level Petri Nets [1]

Martin Kunštátský
e-mail: ikunstatsky@fit.vutbr.cz

Vladimír Janoušek
e-mail: janousek@fit.vutbr.cz

Brno University of Technology,
Faculty of Information Technology,
Božetěchova 1/2, 612 66 Brno, Czech Republlic
IT4Innovations Centre of Excellence

2013-Apr-15

Abstract: One of many possible uses of coloured Petri nets, is the use for modelling complex resource-accessing systems. In this article a resource-access analysis method, based on translation of Petri net model to a complex pi-calculus process, is presented.

The use of coloured Petri nets for modelling resource-accessing systems, suggests an enhancement of the Petri net formalism with a resource data type, which is also described in this work.

A correct-resource-access checking method for the pi-calculus has been known for several years This article shows, how to use this method for analysing static high-level Petri nets.

1 Introduction and Motivation

High-level Petri nets, in the sense of "individual token" nets, belong among the most used means of modelling of discrete concurrent systems.

Possibly the most prominent class of high-level Petri nets are the Coloured Petri nets (abbreviated in the rest of the article as 'CPNs') [2].

As being very strong means of modelling, high-level nets are in general very hard to analyse. In particular, CPNs are usually analysed by constructing a state space graph, which is often problematic, because in almost all practical cases, the state space is extremely large or even infinite [2].

CPNs can be used for modelling of a large class of systems. This article is concerned with analysis of (computer) systems accessing resources, such as files or hardware devices – printers, scanners, etc. The use of CPNs to model such resource accessing systems suggests a variant of CPNs enhanced with a

[1]This work has been supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), by BUT FIT grant FIT-11-1, and by the Ministry of Education, Youth and Sports under the contract MSM 0021630528.

resource data type (or resource colour set), which represents (references to) resources, together with specifying resource accessing primitives.

The method for analysis of CPNs enhanced with a resource data type, suggested in this article is, unlike most of contemporary-used methods of CPN analysis, not based on constructing a state-space graph; it is based on constructing a pi-calculus process simulating the original Petri net instead.

The goal of this article is to show, that a pi-calculus analysis method described in 2006 article by N. Kobayashi, K. Suenaga and L. Wischik *Resource Usage Analysis for the pi-calculus* [3] can be used to analyse CPNs too.

For simplicity, only non-hierachical CPNs are considered in this article. Conclusions, presented here, can be easily used for hierarchical CPN too.

The pi-calculus belongs among the most prominent process algebrae, which are in general mathematic tools for describing systems consisting of concurrent processes communicating over channels. The pi-calculus was defined by R. Milner, J. Parrow and D. Walker in the late eighties and early nineties of the 20th century [6] [5]. The most peculiar feature of the pi-calculus is that it allows channel mobility among processes.

The principle of this method of CPNs analysis is following: The CPN is translated into an equivalent pi-calculus process, which is then analysed by the pi-calculus analysis method.

The method of translating CPN into a complex pi-calculus process, introduced in this work, has a great potential to be used for other methods of analysis.

The content of this article is following: in section 2 there are definitions of the basic formalisms used in this work. Section 3 decribes the method of translation of a restricted variant of CPN to pi-calculus. Section 4 describes succinctly the pi-calculus analysis method defined in [3], its use to analyse CPNs and gives an example. Section 5 concludes.

1.1 Relation to other works

According to authors' knowledge, only a little of research on the relations of the pi-calculus and Petri nets was done so far, and no pi-calculus-based Petri net analysis method was suggested so far.

Xu and Zhang issued an article on business process modelling with both Petri nets and pi-calculus, where they introduced a method of integration of these two business process models together [9].

It has been also known for a long time, that at least a simplified form of pi-calculus (CCS) may be modelled by Petri nets [7].

It has already been mentioned, that the article by Kobayashi et al. [3] was an important source for this work.

2 Formalism definitions

In this section, succinct formal definitions of various formalisms, which are used in this work, are given. Besides Coloured Petri nets and pi-calculus, which are substantial for this work, also CCS is mentioned, which is used in the pi-calculus analysis method as a behavioural type of processes. For more detailed definitions, an interested reader should see the literature.

2.1 Coloured Petri Nets

Here, only a brief definition of non-hierarchical Coloured Petri nets is given, which is sufficient for the purpose of this work, more detailed definition, including definition of hierarchical coloured Petri nets can be found in literature (the most prominent book about CPNs is [2]).

Only structural definition of Coloured Petri nets is given here, for description of dynamical behaviour of the formalism, a reader should see the literature [2].

The definition of CPNs is based on the definition od more primitive clases of nets (place-transition nets), which are notoriously known, and whose definition is outside the scope of this article – the definition can be found in literature (e.g. [8] and hunderds of others). CPNs can be considered as an extension of place-transition nets with types, where each token is labelled with so called 'colour', which is a data value represented by the token.

Note, that the definition of CPNs given in this section (which is a standard definition of CPNs) is somewhat more complex, than that of the Petri nets variant, for which the resource-access analysis method is designed – a simplified variant of CPNs will be used as the object of the analysis.

The set of variables used in a CPN model N is denoted with V_N. A set of expressions provided by the CPN's inscription language is denoted by $EXPR$. For each expression $e \in EXPR$:

- $Type[e]$ denotes the type of the expression.
- $Var[e]$ denotes the set of free variables appearing in the expression.

$EXPR_W$, where $W \subset V$, denotes a set of inscription language expressions, such that $\forall e \in EXPR_W : Var[e] \subseteq W$.

Thus a *non-hierarchical CPN* is a 9-tuple:

$$N = (P, T, A, \Sigma, V, C, G, E, I)$$

where

- P is a finite set of places
- T is a finite set of transitions, $P \cap T = \emptyset$

- A is a set of directed arcs, $A \subseteq (P \times T) \cup (T \times P)$
- Σ is a finite set of non-empty colour sets (or types)
- V is a finite set of typed variables
- C is a colour set (or type) function, $C : P \to \Sigma$, specifying a type of each place (i.e. the type of tokens, which may be present in the place).
- G is a guard function $G : T \to EXPR$, such that $\forall t : Type[G(t)] = Bool$
- E is an arc expression function $E : A \to EXPR$, which specifies for each arc $a \in A$ the expression $E(a)$ defining the values transfered over this arc.
- I is an initialisation function $I : P \to EXPR_{\emptyset}$, specifying the initial values of each place.

Note, that these symbols can be subscribed with N, when necessary (thus for example a set of colour sets of a net N can be denoted as Σ_N).

A *marking* of a CPN is a function

$$M : P \to \Sigma_{MS}$$

mapping each place $p \in P$ into a multiset of values (called tokens). It has to hold, that $M(p) \in C(p)_{MS}$. (Note, that $_{MS}$ subscript denotes a multiset)

Also presets and postsets of places and transitions are denoted in standard way as ${}^\bullet x$ and $x^\bullet$ respectively (here x stands for the place or transition, whose preset or postset is considered).

For more detailed description of standard CPNs, which is out of the scope of this article, an interested reader should look into specialised literature (e.g. famous book *Coloured Petri Nets* by Jensen and Kristensen [2]).

2.2 Basic pi-calculus

This subsection gives a succinct definition of the basic pi-calculus, note, that the variant given in this subsection is not exactly the one, which is used for Petri net analysis – an enhancement for resource access will be given in chapter 4.

The pi-calculus belong among the most popular process algebrae. It serves for describing systems consisting of concurrent communicating processes.

There are many variants of the pi-calculus: *monadic* (that is: transfering only a single value over a channel at a time) [6], polyadic (allowing to transfer tuples of values over a channel at once) [5]. There are also many variants of the pi-calculus extended with typing information [1]. There are also many

definitions of the monadic pi-calculus, which differ from each other subtly. In this work a variant of monadic pi-calculus with replication is used.

For the purpose of high-level Petri net analysis, monadic pi-calculus is sufficient, no polyadicity is needed.

Suppose we have an infinite set of *names* $\mathcal{N}$, individual names are denoted with $a, b, \ldots x, y, z$. The purpose of names is twofold: (1) they represent channels serving for communication between processes and (2) they serve as transferred data values – this is the principle allowing channel mobility

The set of pi-calculus processes **P** is defined by following grammar:

$P ::=$

0	inert process
$\mid x(a).P$	input prefix
$\mid \tau.P$	unobservable action prefix
$\mid \overline{x}\langle a\rangle.P$	output prefix
$\mid P \mid P$	parallel composition
$\mid P + P$	summation
$\mid (\nu x)P$	restriction
$\mid * P$	replication

Note, that at the right side of each rule, there is specified the name of the rule (which usually serves as the name of the corresponding operator too).

Inert process 0 denotes a process with no behaviour(or a process, whose behaviour has finished). Final 0 is usally omitted – e.g. $x(y).\overline{y}\langle z\rangle.0$ should be written as $x(y).\overline{y}\langle z\rangle$.

Whereas *parallel composition* of processes, $P|R$, is used in all variants of pi-calculus definition for denoting a process consisting of two subprocesses running in parallel. This is not the case for *summation*: $P + R$ denotes a process, which can behave either as P or as R, with the choice between the variants being non-deterministic.

Input prefix, $x(a).P$ denotes a process which receives a value denoted with a over a channel x, and then continue behaving like P. Similarly *output prefix*, $\overline{x}\langle a\rangle.P$, denotes a process, that sends a value a over a channel x and then continue behaving like P. Since the *communication is synchronous*, both processes, preforming input and output prefix, are blocked until the value transfer is performed. *Unobservable action* τ, is an action, that cannot be observed from outside of the process (i.e. an action taking place inside the process on a hidden channel serving as a 'null' step).

Replication of a process denotes a potentially infinite number of copies of the process running in parallel. Replication of a process P is in majority of pi-calculus literature denoted with $!P$. But the exclamation mark is sometimes used for denoting the output prefix (i.e. $x!a.P$ instead of $\overline{x}\langle a\rangle.P$), which may potentially cause confusion. This is the reason why the kleene-star-like notation is used for replication in this work.

Communication is possible only between two processes runnung in parallel, one of which runs an input prefix, while the other runs an output prefix, both communication actions have to be performed over a channel, which is shared between the two processes; e.g.

$$x(a).P|\overline{x}\langle b\rangle.R$$

This process reduces in one step into a process

$$P[a \mapsto b]|R$$

where $P[a \mapsto b]$ denotes a process P, where all occurences of name a have been replaced by name b.

$(\nu x)P$ denotes a creation of a 'fresh' channel x for a process P. The creation of a channel can be also wieved as an allocation of a new channel. For the new channel, it is guaranteed, that no other process except (subprocesses of) P will interfere into communication over x.

It is possible to enhance the basic untyped calculus with typing. In this case, each name n has its type T, which is denoted as $n : T$. The typed pi-calculus may use different forms of input prefix and restriction, where the type of the name is specified.

$\mid x(a : T).P$	input prefix
$\mid (\nu x : T)P$	restriction

Structural equivalence relation of processes [2] $\equiv$ is defined as follows ($\mathbf{P}$ denotes the set of all pi-calculus processes):

1. $(\mathbf{P}, |, 0)$ forms abelian monoid:

$$P|Q \equiv Q|P \quad (1)$$
$$P|(Q|R) \equiv (P|Q)|R \quad (2)$$
$$P|0 \equiv P \quad (3)$$

2. rules for restriction

$$(\nu x)0 \equiv 0 \quad (4)$$
$$(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P \quad (5)$$
$$(\nu x)P|Q \equiv (\nu x)(P|Q) \text{ if } x \text{ not free in } Q \quad (6)$$

[2] Note, that this structural relation is sometimes defined as a quasiorder (e.g. [3]), allowing only one-directional transformations. For the purpose of Petri Nets analysis, its definition as an equivalence is sufficient.

E.g. the often problematic translation of $*P|P$ to $*P$, which is correct in untyped pi-calculus, but may cause problems in typed pi-calculus, is not a problem in pi-code obtained by translation of a Petri net (two different nodes within a single net cannot be represented by the same pi-code).

3. rule of replication

$$*P \equiv P | *P \tag{7}$$

Some additional notation:

- Let $\sum_{i\in\{1,\ldots,n\}} P_i$ be a shorthand notation for $P_1 + \cdots + P_n$.
- Let $\prod_{i\in\{1,\ldots,n\}} P_i$ be a shorthand for $P_1 | \ldots | P_n$.

So far, only *monadic* variant of the pi-calculus was described. The *polyadic variant* differs by the fact, that *tuples of names* are sent over channels, i.e. the input and output prefixes have the form

$P ::=$

$\mid x(a_1, \ldots a_n).P$	input prefix
$\mid \overline{x}\langle a_1, \ldots, a_n\rangle.P$	output prefix

The polyadic variant of the pi-calculus is mentioned here only for completeness, in the main part of this article, it is mentioned only briefly.

More precise definition of the pi-calculus can be found in [6], [5].

2.3 CCS

The *calculus of communicating systems* (CCS) can be described as a sublanguage of the pi-calculus, where there is a single 'null' name, denoted for the purpose of this article with ε, which is the only value, that can be transferred via other channels. The null name cannot be used as a channel transfering other names. All pi-calculus rules are applicable in CCS except *replication*.

Since the ε is the only value, that can be transferred in CCS, there is no need to specify the transferred value at all, i.e. $\overline{x}.P$ and $x.R$ are used to represent $\overline{x}\langle\varepsilon\rangle.P$ and $x(\varepsilon).R$, respectively.

The greatest difference from the pi-calculus is that the channels transfering other channels cannot be transfered between processes, which implies that CCS does not support *channel mobility* among processes.

The only reason, why CCS was mentioned in this section, is that it will be used as a behavioural type of pi-calculus processes in section 4.

3 Translation

In this section, the method of translation of a CPN model to an equivalent pi-calculus model is described. The pi-calculus code in fact simulates the CPN.

This method serves for creation of equivalent pi-calculus code for a restricted CPN. The obtained code can be then used for analysis of the original CPN. There are possibly many pi-calculus analysis methods applicable for

the pi-code representing the original CPN. One of these method (resource-access analysis) is decribed in the next section.

The variant of CPNs considered in this section is very restricted:

- It does not use transition guards
- The initial marking is not considered (i.e. the net is initially empty) – the lack of initial marking can be alleviated easily by a special initialising transition, which puts the 'initial' values to each place of the net.
- Each arc of the CPN is allowed to transfer only one value at a time. Since in general CPNs a multiset of values can be transfered over an arc at a time, this condition leads to the possible use of parallel arcs.

Let x be an object from a CPN (place, transition, arc, etc.). Then $\hat{x}$ denotes a sentence of pi-calculus code representing x.

Let's also denote for any node (transition or place) of a CPN:

- by ${}^*x$ the set of arcs leading into the node x (i.e. ${}^*x = \{(y,x)|y \in {}^\bullet x\}$);
- by x^* the set of arcs leading from the node x (i.e. $x^* = \{(y,x)|y \in x^\bullet\}$).

It will be also 'forgotten' for the rest of this article, that an arc is a 2-tuple, and it will be treated it as an atomic value.

The process of translation of a CPN into a pi-code can be decomposed into several parts: translation of the structure of the net and representation of colour sets – these parts will be described in following subsections, also, since no proof of equivalence between the original CPN and its pi-representation exists so far, an argumentation for correctness of the method will be given.

3.1 Net Structure

First note, that a pi-calculus channel and a Petri net arc behave in very similar manner, that is: both serve as one-directional connection for information transfer between two entities (i.e. processes in the case of pi-calculus and places or transitions in the case of a Petri net). This suggests, that arcs in Petri net can be directly and very faithfully modelled by pi-calculus channels. Petri net nodes (i.e. places and transitions) have to be modelled by special pi-calculus processes.

Since we consider only a simplified variant of the CPNs, where each arc can transfer a single value at a time, the target pi-calculus code can be strictly monadic.(It would be possible to use polyadic pi-calculus, where an arc transfering a multiset of n items would be modelled by a n-adic channel (that is a channel transfering n-tuples of names, but this would lead to complications while defining the processes representing CPN nodes).

Thus for each Petri net arc a, there is an equivalent pi-calculus channel denoted as $\hat{a}$. In cases, when it is clear whether a represents arc or channel, the 'wedge' (or 'hat') $\hat{}$ diacritic mark can be omitted.

Each place $p \in P$ can be modelled by a pi-calculus code:

$$\hat{p} = * \left(\sum_{\hat{a} \in {}^*p} \hat{a}(v) \right) . \left(Cell[v]. \sum_{\hat{b} \in p^*} \overline{\hat{b}} \langle v \rangle \right) \tag{8}$$

Where $Cell[v]$ denotes a memory cell keeping one value (v) appearing in the place. Input arcs are iterated with $\hat{a}$, output arcs are iterated with $\hat{b}$.

Whenever a new value is received from any of input channels $\hat{a}$, a new copy of $Cell[]$, keeping the value, is created. Each copy of $Cell[]$ is always followed with representations of output arcs of the place. Note, that a representation of a place keeping some values is a group of $Cell$s keeping individual values runing in parallel.

Note, that $Cell[v]$ is not needed, after the value kept in the place was substituted for v in the following code, so after the substitution, the code $Cell[]$ can be erased.

Each transition $t \in T$, can be modelled by a pi-calculus code:

$$\hat{t} = * \left(\prod_{\hat{a} \in {}^*t} \hat{a}(v_a) \right) .Copy(\dots). \left(\prod_{\hat{b} \in t^*} \overline{\hat{b}} \langle v_b \rangle \right) \tag{9}$$

Where v_a denotes a single value transfered over the arc a, $Copy(\dots)$ denotes a function computing output-arc values v_b from the input-arc values v_a. Obviously a process representing a transition can be run only if each of the processes representing input places of the transition have $Cell[v]$, where v is a bound variable.

A Petri net $N = (P, T, A)$, where $P = \{p_1, \dots, p_n\}$ and $T = \{t_1, \dots, t_m\}$ will be translated into a pi-calculus code

$$\hat{N} = \hat{p}_1 | \dots | \hat{p}_n | \hat{t}_1 | \dots | \hat{t}_m$$

3.2 Colour sets

In general, tokens of the original CPN are represented by names of the pi-calculus (which, in CPNs do not serve as channels transfering values, but note, that in modelling more dynamic variants of high-level nets like reference nets [4], the feature of channel mobility can be utilized).

There are basically two ways, how to represent colour sets:

- In the case of *finite colour sets*, the situation is very simple – all functions operating over these sets are finite and thus can be represented by table-like representation. The values of the finite colour set can be represented simply by allocating a name for each value or the colour set.

- Another possibility to represent a colour set in pi-calculus, is by means of typing. Each colour set $T \in \Sigma_N$ of the original CPN N will be represented by a pi-calculus type $\hat{T}$.

  If a place p has type (colour set) T, then its representation in typed pi-calculus is

$$\hat{p} = * \left(\sum_{\hat{a} \in {}^*p} \hat{a}(v : \hat{T}) \right) . \left(Cell[v : \hat{T}]. \sum_{\hat{b} \in p^*} \overline{\hat{b}}\langle v \rangle \right)$$

3.3 Argumentation for correctness

In this subsection an argumentation for correctness of the translation, instead of formal proof of equivalence between an original CPN and its representation in pi-calculus, is given.

Let's have a place p with $p^* = \{a_1, \ldots, a_n\}$ and $p^\bullet = \{t_1, \ldots, t_n\}$ (such that for each t_i there is $a_i \in {}^*t_i$). Presence of a token with value b in the place p is represented by pi-calculus code $Cell[b].(\overline{a}_1\langle b \rangle + \cdots + \overline{a}_n\langle b \rangle)$, where b is a nonfree variable (i.e. a name bearing the value b). Since b is a nonfree name, there is no need to keep the $Cell[b]$ piece of code, so we can simply represent the presence of the token in the place as

$$\overline{a}_1\langle b \rangle + \cdots + \overline{a}_n\langle b \rangle$$

Let's have a marking M of a net N, such that marking of a place $p \in P_N$ is $M(p) = \{b_1, \ldots, b_n\}_{MS}$. Let's also denote the piece of pi-code representing p following the $Cell[b]$ as $cont\langle b \rangle = \overline{a}_1\langle b \rangle + \cdots + \overline{a}_n\langle b \rangle$. The place p with the marking $M(p)$ is represented as

$$\hat{M}(p) = \hat{p}|cont\langle b_1 \rangle| \ldots |cont\langle b_n \rangle$$

Note that $\hat{p}$ serves only for acception of new values (i.e. reception of new tokens by the place) and thus creation of new *cont* pieces of code.

Thus if a CPN N has $P_N = \{p_1, \ldots, p_n\}$ a marking of the net M is represented by

$$\hat{M}(p_1)| \ldots |\hat{M}(p_n)$$

(Note, that for representation of the whole net, there must be also represenations of all transitions in parallel to this.)

Because transitions are the only active elements in this restricted variant of CPNs, the equivalence of transition behaviour between the original CPN and its representation in pi-calculus should be sufficient for equivalence of the whole net and its representation.

It should be clear from the definition of translation of transitions and nodes (equations 9 and 8) , that the structure of the representation of a CPN is equivalent to the original CPN.

In CPNs an occurence of a transition t removes some tokens (which are specified by bindings of its variables) from all places in ${}^\bullet t$ and adds some tokens (also specified by bindings) into all places in $t^\bullet$.

If the function *Copy* from the representation $\hat{t}$ (equation 9) of the transition is specified correctly, it should be clear that the behaviour of the representation of the transition $\hat{t}$ is equivalent to the behaviour (i.e. marking change) caused by occurence of the original transition.

4 Resource Usage Analysis

This part describes resource usage analysis of CPNs, enhanced with a resource data type. The method of the analysis is straightforward: translate the analysed CPN to an equivalent pi-calculus model (which was described in the previous section), which shall be then analysed with a method described in [3].

Note that no proof of correctness of this method is given, since the original method of pi-calculus analysis was justified enough in the original article [3].

Our models (i.e. CPNs and the pi-calculus) are enhanced with the notion of resources: the set of resources will be denoted with $\mathcal{R}$; the same symbol will be used also for the resource data type (the distinction should be clear from the context).

Each resource $r \in \mathcal{R}$ has a *set of operations* Σ_r, (also known as a set of *access labels*), which describe possible operations with the resource. Each resource has also *access protocol* (trace set) Φ, which is a regular set [3] describing possible access sequencies of operations to the resource.

Example: for a file there is $\Sigma = \{r, w, i, c\}$, denoting operations "read", "write", "initialise" and "close", respectively. And the access protocol for the file is

$$\Phi = i(r + w)^* c$$

which means, that a file have to be first initialised (that is: opened), then it can be read or written arbitrarily times in arbitrary order and in the end it must be closed.

We suppose a variant of CPNs enhanced with a resource data type $\mathcal{R}$, where each transition may perform an accessing operation ξ to a resource $r \in \mathcal{R}$, this access operation is denoted with an inscription $\mathbf{acc}_\xi(r)$. Besides resource access, there is also a need of resource allocation operation ($\mathbf{N}^\Phi r$),

[3] Note, that the method does not require the access protocol to be a regular set. In fact, the access protocol could belog to any class of languages, which it is sufficiently easy to manipulate with. Since the class of regular languages is an example of such sufficiently easy-to-manipulate language class, and at the same time it it seems to be complex enough for most practical cases, we recommend the access protocol to be always a regular set.

which allocates a new resource r with specification of its accessing protocol Φ.

This method uses a variant of the pi-calculus and CCS enhanced with primitives for creating and accessing resources (which are very similar to the the operations used within CPNs; the *set of resources* $\mathcal{R}$ can be considered a subset of names $\mathcal{N}$). This adds two new rules to the grammar for the pi-calculus process:

$P ::=$

$\mathbf{acc}_\xi(x).P$	resource accessing prefix
$\mid (\mathbf{N}^\Phi x)P$	resource restriction

In this definition $\xi \in \Sigma_x$ is an acess label and $x \in \mathcal{R}$ is a resource. Resource restriction denotes creation (or allocation) of a new resource together with the specification of its access protocol Φ. The resource restriction is similar to the common name restriction of the basic pi-calculus.

An access to the resource removes the first symbol from the trace set:

$$(\mathbf{N}^{\xi\cdot\Phi}x)\mathbf{acc}_\xi(x).P \equiv (\mathbf{N}^\Phi x)P \tag{10}$$

where $\xi \in \Sigma$ and $\Phi \in \Sigma^*$

The principle behind the pi-calculus analysis method described in [3]: each channel is labelled with so-called 'behavioural type', which is in fact a CCS sentence describing the behaviour of the receiver process, after receiving the value (or the *continuation* of the receiving process). This behavioural description is then distributed over the system, in the end the behaviour of the entire pi-calculus code is "subtracted" from the access protocol Φ at the restriction of the resource, leaving only empty set.

There are two categories of types:

- value types (of names $\mathcal{N}$)
- behavioural types (of processes and channels)

The behavioural type of a process is a CSS process approximating the behaviour of the process, where the transfered values were abstracted. The grammar of the behavioural type:

$$\begin{aligned} A \;::=\; & 0 \;\mid\; x.A \;\mid\; \overline{x}.A \;\mid\; x^\xi.A \;\mid\; \tau.A \;\mid \\ & A|A \;\mid\; A+A \;\mid\; *A \;\mid\; A\uparrow_S \;\mid\; A\downarrow_S \;\mid\; \langle x/y\rangle A \end{aligned}$$

Where S is a set of names. $x.A$, $\overline{x}.A$, $x^\xi.A$, $\tau.A$ represent input prefix, output prefix, resource accessing prefix and unobservable prefix, respectively. The meaning of $A|B$, $A+B$, $*A$ is straightforward,

$A\uparrow_S$ describes a process behaving like A, removing those actions, that work with some member of S (i.e. actions working with some member of S are replaced with τ); $A\downarrow_S$ represents a process behaving like A, where those

actions not working with some member of S are replaced with τ. $\langle x/y\rangle A$ is a type A, where all occurences of x were substituted with y.

For the behavioural types, there is defined a *subtyping relation* $A_1 \leq A_2$, which denotes, that A_2 simulates A_1 (i.e. a process, whose behaviour is approximated by A_1, can be also viewed as being approximated by A_2).

Besides process types, there are also value types of names, which are basically the same as colours in CPN. The only exception is the *channel type*: $\mathbf{chan}\langle val; behav\rangle$,which represents names representing channels transfering values (i.e. those channels representing arcs of the original CPN), where *val* is a type of the transferred value and *behav* is the behavoural type of the receiving process. Moreover, there is also *resource type* $\mathcal{R}$ (note, that the same symbol is used for the type of resources and the set of resources; the distinction should be clear from the context) – a name, which has resource type represents a resource.

Type environment Γ is a mapping from the set of variables to types. $\Gamma, y : \sigma$ denotes $\Gamma \cup \{y \mapsto \sigma\}$ (i.e. adding statement, that variable y has type σ to the type environment).

Althought quite straightforward, the *transition relation on behavioural types* is of some theoretical importance, it is denoted as $\xrightarrow{l}$, where the label l may be

$$l ::= x \quad | \quad \overline{x} \quad | \quad x^{\xi} \quad | \quad \tau$$

The labels specify the forms of access to a name x: x and $\overline{x}$ denote reception and sending a value (respectively) on a channel x; x^{ξ} denotes access to a resource x. Complete definition of this relation may be found in [3].

Let also $\Rightarrow^{x^{\xi}}$ denote $\xrightarrow{\tau}{}^{*} \xrightarrow{x^{\xi}} \xrightarrow{\tau}{}^{*}$ ($\xrightarrow{\tau}{}^{*}$ denote reflexive and transitive closure of $\xrightarrow{\tau}$).

The set $\mathbf{traces}_x(A)$ is the set of all possible access sequences on a resource $x \in \mathcal{R}$ described by behavioural type A.

$$\mathbf{traces}_x(A) = \{\xi_1, \ldots, \xi_n | A \downarrow_{\{x\}} \Rightarrow^{x^{\xi_1}} \cdots \Rightarrow x^{\xi_n}\}$$

Critically important is the *type judgment relation* $\Gamma \triangleright X : Y$. There are two kinds of type judgments:

- $\Gamma \triangleright v : \sigma$ for variables, which states that according to Γ, variable v has type σ, i.e. $\Gamma(v) = \sigma$
- $\Gamma \triangleright P : A$ for processes, which states, that a process P behaves according to behavioural type A.

The type judgment relation is inferred using rules from table 1. [4]

[4] All these rules were taken from [3], except T-sum, which was missing in the paper, possibly by a mistake.

$\Gamma \triangleright 0 : 0$	T-zero
$\frac{\Gamma \triangleright P{:}A_2 \; \Gamma \triangleright x{:}\mathbf{chan}\langle y{:}\sigma; A_1\rangle \; \Gamma \triangleright v{:}\sigma}{\Gamma \triangleright \overline{x}\langle v\rangle.P{:}\overline{x}.(\langle v/y\rangle A_1 \mid A_2)}$	T-out
$\frac{\Gamma, y{:}\sigma \triangleright P{:}A_2 \; \Gamma \triangleright x{:}\mathbf{chan}\langle y{:}\sigma; A_1\rangle \; A_2{\downarrow_{\{y\}}} \leq A_1}{\Gamma \triangleright x(y).P{:}x.(A_2{\uparrow_{\{y\}}})}$	T-in
$\frac{\Gamma \triangleright P_1{:}A_1 \; \Gamma \triangleright P_2{:}A_2}{\Gamma \triangleright P_1 \mid P_2{:}A_1 \mid A_2}$	T-par
$\frac{\Gamma \triangleright P_1{:}A_1 \; \Gamma \triangleright P_2{:}A_2}{\Gamma \triangleright P_1 + P_2{:}A_1 + A_2}$	T-sum
$\frac{\Gamma \triangleright P{:}A}{\Gamma \triangleright *P{:}*A}$	T-rep
$\frac{\Gamma, x{:}\mathbf{chan}\langle y{:}\sigma; A_1\rangle \triangleright P{:}A_2}{\Gamma \triangleright (\nu x)P{:}(\nu x)A_2}$	T-new
$\frac{\Gamma \triangleright P{:}A \; \Gamma \triangleright x{:}\mathcal{R}}{\Gamma \triangleright \mathbf{acc}_\xi(x).P{:}x^\xi.A}$	T-acc
$\frac{\Gamma, x{:}\mathcal{R} \triangleright P{:}A \; \mathbf{traces}_x(A) \subseteq \Phi}{\Gamma \triangleright (\mathbf{N}^\Phi x)P{:}A{\uparrow_{\{x\}}}}$	T-newR
$\frac{\Gamma \triangleright P{:}A' \; A' \leq A}{\Gamma \triangleright P{:}A}$	T-sub

Table 1: Table of rules for type judgment relation.

Theorem: When it is possible for a process P to derive $\triangleright P : 0$, the process accesses all its resources correctly [3]. [5]

A limitation of this method is, that it cannot deal with channel mobility, which implies, that it can be used to analyse static nets like CPNs, but it cannot be used to analyse more dynamic classes of Petri nets like Reference nets ([4]).

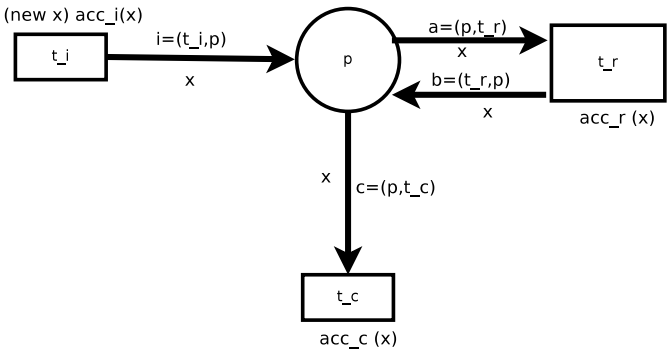

Figure 1: Simple Petri Net from the example

[5] More details about this method may be found in article [3], including description of the type inference algorithm. Since the article was issued under Creative Commons licence, it should be easily and freely available.

4.1 Example

Let's have a simple Petri net $N = (\{p\}, T, A)$, where $T = \{t_i, t_r, t_c\}$ and $A = \{i = (t_i, p),\ a = (p, t_r),\ b = (t_r, p),\ c = (p, t_c)\}$. This net is performing actions with a resource $x \in \mathcal{R}$, which has operations

$$\Sigma_x = \{i, r, c\}$$

denoting "initialise", "read" and "close", respectively and the access protocol is

$$\Phi = ir^*c$$

The net is shown on figure 1.

Each transition $t_a \in T$ is performing corresponding action $a \in \Sigma_x$. The transition t_i creates a new resource x and performs $\mathbf{acc}_i(x)$, similarly t_r and t_c perform $\mathbf{acc}_r(x)$ and $\mathbf{acc}_c(x)$ respectively.

Each arc performs straightforward transfer of resource. t_r neither t_c does not make any change with the transfered resource, except the access to it.

Note, that access only to one resource is considered – the fact, that t_i can anytime create another resource and put it into the place, is ignored.

The behavour of t_r and t_c implies, that we can simplify the pi-calculus code for the two transitions (9) by unifiing the input and output name, omiting the *Copy* function and adding the resource access, thus yielding

$$\begin{aligned} \hat{t}_r &= *\hat{a}(v).\mathbf{acc}_r(v).\overline{\hat{b}}\langle v\rangle \\ \hat{t}_c &= *\hat{c}(v).\mathbf{acc}_c(v) \end{aligned}$$

Of course, the initialising transition must have different code

$$\hat{t}_i = (\mathbf{N}^{ir^*c}x)\mathbf{acc}_i(x).\overline{\hat{i}}\langle x\rangle$$

which is equivalent to (using 10)

$$\hat{t}_i = (\mathbf{N}^{r^*c}x)\overline{\hat{i}}\langle x\rangle$$

The pi-code for the sole place p (straightforwardly from 8):

$$\hat{p} = *(\hat{i}(v) + \hat{b}(v)).Cell[v].(\overline{\hat{a}}\langle v\rangle + \overline{\hat{c}}\langle v\rangle)$$

The typing environment is:

$$\begin{aligned} \Gamma \;=\; & x : \mathcal{R}, \\ & \hat{i} : \mathbf{chan}\langle x : \mathcal{R}; Cell[x].(\overline{a} + \overline{c})\rangle, \\ & \hat{a} : \mathbf{chan}\langle x : \mathcal{R}; x^r.\overline{b}\rangle, \\ & \hat{b} : \mathbf{chan}\langle x : \mathcal{R}; Cell[x].(\overline{a} + \overline{c})\rangle, \\ & \hat{c} : \mathbf{chan}\langle x : \mathcal{R}; x^c\rangle \end{aligned}$$

Thus we need to analyse (pi-calculus code equivalent to the original petri net)

$$\Gamma \triangleright \hat{p}|\hat{t}_i|\hat{t}_c|\hat{t}_r$$

Because the variable in $Cell$ in $\hat{p}$ is free, $\hat{t}_r$ and $\hat{t}_c$ are not enabled, so $\hat{t}_i$ has to be run first (let's also denote the process $\hat{t}_i|\hat{p}$ as Q):

$$Q = \hat{t}_i|\hat{p} = (\mathbf{N}^{r^*c}x)\overline{i}\langle x\rangle| * (i(v) + b(v)).Cell[v].(\overline{a}\langle v\rangle + \overline{c}\langle v\rangle)|\hat{p}$$

which is equivalent to (using 6, 7 and T-out)

$$Q = (\mathbf{N}^{r^*c}x)\left((\overline{a}\langle x\rangle + \overline{c}\langle x\rangle)|\hat{p}\right)$$

Note, that we do not need to use $Cell[x]$ anymore, since in the whole following code x was substituted for all occurences of v.

The process Q has type:

$$\Gamma \triangleright Q : Cell[x](\overline{a} + \overline{c})| * (i + b).Cell.(\overline{a} + \overline{c})$$

Let's follow with processes $\hat{t}_r$ and $\hat{t}_c$

$$Q|\hat{t}_r|\hat{t}_c = (\mathbf{N}^{r^*c}x)\left(((\overline{a}\langle x\rangle + \overline{c}\langle x\rangle)|\hat{p}| * a(x).\mathbf{acc}_r(x).\overline{b}\langle x\rangle| * c(x).\mathbf{acc}_c(x)\right)$$

Let's denote the process following the resource restriction as

$$R = (\overline{a}\langle x\rangle + \overline{c}\langle x\rangle)|\hat{p}| * a(x).\mathbf{acc}_r(x).\overline{b}\langle x\rangle| * c(x).\mathbf{acc}_c(x)$$

So, we are analysing

$$\Gamma \triangleright (\mathbf{N}^{r^*c}x)R$$

Since there is a cycle in the original CPN graph, we have to use a substitution: let A denote behaviour of the process after leaving p on x. Thus we get

$$A = (\overline{a} + \overline{c})$$

and using the types of channels $\hat{a}$, $\hat{b}$, $\hat{c}$ (note that behaviour of continuation of $\hat{b}$ is equivalent to A) together with rule T-out:

$$A = x^r A + x^c$$

This implies

$$\mathbf{traces}_x(A) = r \cdot \mathbf{traces}_x(A) + c$$

This equation has a single solution:

$$\mathbf{traces}_x(A) = r^*c$$

Note, that $\mathbf{traces}_x(A) \subseteq r^*c$ holds, so by using rule T-newR, we get $\triangleright R : 0$. Thus we conclude, that the net accesses the resource correctly.

5 Conclusion

In this work, a method of resource-access analysis of coloured Petri nets based on the pi-calculus was described. Although it does not present exact mathematical proofs of its claims, authors of this article believe, that this article will find its practical realisations and will facilitate a new direction of research.

For the purpose of resource-access analysis, also a method for creating a complex pi-calculus process simulating a given coloured Petri net was described. This method can potentially be used together with various methods of pi-calculus analysis as a means of analysis of Petri nets. The resource-access-analysis method described in this article, is in fact an example of such pi-calculus-based Petri nets analysis. Other methods of such pi-calculus-based Petri nets analysis can be objects of futher research.

References

[1] D. Hirschkoff. A brief survey of the theory of the pi-calculus, 2003. Research report No 2003-13.

[2] K. Jensen and L. M. Kristensen. *Coloured Petri Nets. Modelling and Validation of Concurrent systems*. Springer-Verlag, Berlin, 2009. ISBN 987-3-642-00283-0.

[3] N. Kobayashi, K. Suenaga, and L. Wischik. Resource usage analysis for the pi-calculus. *Logical Methods in Computer Science*, 2006.

[4] O. Kummer, F. Wienberg, M. Duvigneau, and L. Cabac. *Renew – User Guide*, 2012.

[5] R. Milner. The polyadic pi-calculus: a tutorial. Technical report, Logic and Algebra of Specification, 1991.

[6] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part 1 and 2. *Information and Computation*, 100, 1989.

[7] M. Nielsen and V. Sassone. Petri nets and other models of concurrency. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets 1. Basic Models*. Springer-Verlag, Berlin, 1998.

[8] E. Smith. Principles of high-level net theory. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets 1. Basic Models*. Springer-Verlag, Berlin, 1998.

[9] F. Xu and L. Zhang. Unified modeling and analysis based on petri nets and pi calculus. *Sixth International Symposium on Theoretical Aspects of Software Engineering*, 2007.