Methods of translation of Petri nets to NuSMV language

Marcin Szpyrka
AGH University of Science and Technology in Kraków
93 PUBLICATIONS **637** CITATIONS
SEE PROFILE

Agnieszka Biernacka
AGH University of Science and Technology in Kraków
6 PUBLICATIONS **47** CITATIONS
SEE PROFILE

Jerzy Biernacki
AGH University of Science and Technology in Kraków
9 PUBLICATIONS **54** CITATIONS
SEE PROFILE

Methods of translation of Petri nets to NuSMV language

Marcin Szpyrka, Agnieszka Biernacka, and Jerzy Biernacki

AGH University of Science and Technology
Department of Applied Computer Science
Al. Mickiewicza 30, 30-059 Kraków, Poland
mszpyrka@agh.edu.pl, agnieszkazelezik@gmail.com,
jerzyjanbiernacki@gmail.com

Abstract. The paper deals with the problem of translation of reachability graphs for place-transition and coloured Petri nets into the NuSMV language. The translation algorithms presented in the paper have been implemented as a part of the PetriNet2NuSMV tool so the translation is made automatically. The PetriNet2NuSMV tool works with reachability graphs generated by the TINA and CPN Tools software. Thus, it provides the possibility of formal verification of Petri nets designed with these environments using model checking techniques and a mainstream model checker for LTL and CTL temporal logics.

Keywords: place-transition nets, coloured Petri nets, NuSMV, translation, model checking, PetriNet2NuSMV translator

1 Introduction

The most widely used software verification techniques are peer reviewing and testing. In case of concurrent systems, these techniques can cover only a limited portion of possible behaviours. Formal methods can be used to establish a concurrent system correctness with mathematical rigour. Model checking [2], [4] is one of the most promising techniques for automatic software analysis and Petri nets [8], [11], [14], [15] are one of the most widespread formalisms used in software engineering. Unfortunately, software tools for Petri nets are rarely equipped with model checking algorithms. The presented approach combines two popular Petri nets modelling environments (TINA [3] and CPN Tools [9]) with a mainstream model checker for temporal logic (NuSMV [5], [4]).

There are some similar tools described in the literature such as PEP [13] and Model-Checking Kit [12]. Among other features they allow to verify Petri net models using CMU SMV tool. Nevertheless, development of these tools was stopped in 2004. Petri net model checking is also possible in LoLA [17] and PROD [1] software tools that provide LTL and CTL formulae verification. Alas, none of these tools supports NuSMV model checker or popular coloured Petri nets. The only existing solution for model checking of CP-nets is CPN Tools library. It allows to verify formulae expressed in the ASK-CTL temporal logic which is an extension of the CTL logic. One of inconveniences of this solution is that knowledge of ML functional language is required. Another disadvantage is the lack of support for formulae written in the LTL logic.

The aim of this work was to create a tool for translation of reachability graphs of both place-transition and coloured Petri nets into a description of the system in the

NuSMV language. As a result, the tool provides for the automatic model checking of nets created in the TINA and CPN Tools environments. The presented solution is the only one that allows verification of formulae in both LTL and CTL temporal logics for coloured Petri nets as well as for PT-nets.

The paper is organised as follows. Section 2 provides a short introduction to the NuSNV language and tool. Section 3 deals with reachability/coverability graphs for place-transition nets and the algorithm of transformation of such graphs into the NuSMV language. The algorithm of transformation of reachability graphs for coloured Petri nets into the NuSMV language is presented in Section 4. Section 5 provides some test results for the presented algorithms. A short summary is given in the final section.

2 NuSMV

NuSMV [4] is one of the most popular model checkers for temporal logic. Given a finite state model and a formula, NuSMV can be used to check automatically whether or not the model satisfies the formula. Formulae can be treated as a specification of requirements for a given model and can be expressed using LTL [6] or CTL [6], [7] temporal logics.

In the NuSMV approach, the verified system is modelled as a *finite state transition system* [5]. Such a system is described as a tuple $TS = (S, I, \rightarrow, L)$, where S is a finite set of *states*, $I \subseteq S$ is the set of *initial states*, $\rightarrow \subseteq S \times S$ is the *transition relation*, specifying the possible transitions from state to state, and L is the *labelling function* that labels states with *atomic propositions* that hold for the given state. Such a tuple is also called *Kripke structure* [10].

```
MODULE main
VAR
  s : {s0, s1, s2};       } set of states
  a : boolean;            } atomic propositions given implicitly
  b : 0 .. 2;
ASSIGN
  init(s) := {s0, s2};    } initial states

  next(s) := case
    s = s0 : s1;
    s = s1 : {s1, s2};    } transition relation
    s = s2 : s2;
  esac;

  a := case
    s = s0 : TRUE;
    s = s2 : TRUE;
    TRUE   : FALSE;
  esac;
                          } labelling function
  b := case
    s = s0 : 1;
    s = s1 : 2;
    TRUE   : 0;
  esac;
```

Fig. 1. Finite state transition system written with the NuSMV language

NuSMV is equipped with a dedicated modelling language (the SMV language), which is used to define finite state transition systems [5]. An example of such a model is given in Fig. 1. A NuSMV model consists of two sections: `VAR` and `ASSIGN`. The `VAR` section contains definitions of variables, including set of states and atomic propositions variables. The `ASSIGN` section is composed of three main parts. The first one is the initialisation of the state variable. The second part is responsible for defining transitions between the states. The last part assigns values to the atomic propositions for specific states. The set of atomic propositions is given implicitly using variables and their domains. For example, the following expressions can be considered as atomic propositions: a (i.e. $a = TRUE$), $!a$, $b = 0$, $b > 1$ etc.

A finite state transition systems is stored as a text file. NuSMV statements `LTLSPEC` and `CTLSPEC` can be used to include LTL and CTL formulae respectively into the file. In case of the LTL logic, the temporal operators G (globally), F (finally), X (next), U (until) can be used. Moreover, the propositional logic operators are represented by: `!` (not), `&` (and), `|` (or), `xor` (exclusive or), `->` (implies) and `<->` (equivalence). In case of CTL following temporal logic operators can be used: EG (exists globally), EX (exists next state), EF (exists finally), AG (forall globally), AX (forall next state), AF (forall finally), $\mathsf{E[\,U\,]}$ (exists until), $\mathsf{A[\,U\,]}$ (forall until). Satisfaction of each specified formula is verified with the NuSMV tool automatically. If a modelled system does not satisfy a given formula, a proper counterexample is presented. It is finally worth mentioning that NuSMV can verify systems of high complexity, i.e. containing more than 10^{20} states. These features make NuSMV useful and convenient tool for finite automata verification.

3 Place-transition nets

Place-transition nets (PT-nets) [11] are the most popular class of Petri nets. A PT-net is defined as a tuple $\mathcal{N} = (P, T, A, W, M_0)$, where P and T are non-empty finite sets of *places* and *transitions* ($P \cap T = \emptyset$), $A \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs* (*flow relation*), $W\colon A \to \mathbb{N} \setminus \{0\}$ is a *weight function* ($\mathbb{N} = \{0, 1, 2, \dots\}$), and $M_0\colon P \to \mathbb{N}$ is the initial marking.

A state of a PT-net is called *marking* and is a distribution of tokens among places of the net. Tokens are indistinguishable one from another so a marking of a place is represented by the number of tokens stored in the place. If $P = \{p_1, \dots, p_n\}$, then a marking M is represented by the vector $M = (M(p_1), \dots, M(p_n))$. A *firing* of a transition may change the current marking. A transition t is *enabled* if each input place p of t contains at least $W(p, t)$ tokens, where $W(p, t)$ is the weight of the arc from p to t. A firing of an enabled transition t removes $W(p, t)$ tokens from each input place p of t, and adds $W(t, p)$ tokens to each output place p of t. We write $M \xrightarrow{t} M'$ to indicate that a firing of a transition t transforms M to M'.

A sequence of firings results in a sequence of markings. A marking M' is said to be *reachable* from a marking M if there exists a sequence of firings that transforms M to M', i.e. there exist markings $M_1, \dots, M_n$ and transitions $t_1, \dots, t_n$ such that $M \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots \xrightarrow{t_n} M_n = M'$. The firing sequence $t_1, \dots, t_n$ is said to be *feasible* from the marking M. The set of all markings reachable from a marking M is denoted by $\mathcal{R}(M)$ and the set of all firing sequences feasible from M is denoted by $\mathcal{L}(M)$.

A *reachability graph* of a PT-net $\mathcal{N} = (P, T, A, W, M_0)$ is the directed graph $\mathcal{G} = (V, A)$, such that the set of nodes $V = \mathcal{R}(M_0)$, and the set of arcs $A = \{(M, t, M') \in V \times T \times V \colon M \xrightarrow{t} M'\}$. The arcs are labelled with transition names and there may exist more than one arc between the same pair of nodes. An example of a PT-net and its reachability graph are shown in Fig. 2.

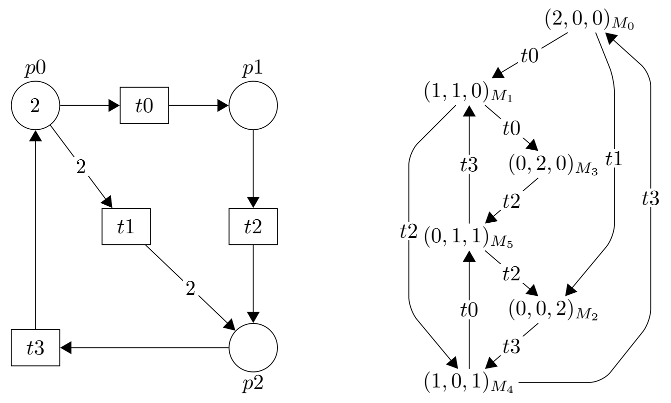


Fig. 2. A PT-net and its reachability graph

Let us consider the PT-net from Fig. 2 but with $W(t1, p2) = 3$, $W(p2, t3) = W(t3, p0) = 2$. In such a case the number of tokens in places of the net can grow infinitely, so the reachability graph is infinite. Then, the coverability graph [11] can be used to represent the PT-net reachable markings. To keep the graph finite, the infinity symbol is used to represent the unbounded number of tokens. The coverability graph for the considered net is given in Fig. 3.

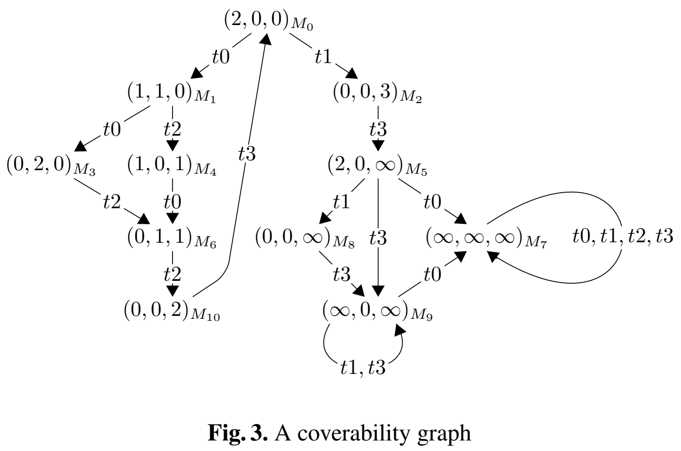


Fig. 3. A coverability graph

The approach presented in the paper uses TINA [3] modelling environment for the design and generation of reachability/coverability graphs for PT-nets. The graphs can be stored using a few different formats. The PetriNet2NuSMV tool uses the *kts* (Kripke transition system) format. To generate a coverability graph for a PT-net choose *Tools→ reachability analysis* and choose options *coverability graph* and *kts* (*ktz*). Then, the generated graph can be stored in a file and finally read by the PetriNet2NuSMV translator. A part of the *kts* file for the reachability graph from Fig. 2 is given below.

```
state 0
props p0*2
trans t0/1 t1/2
```

Kts files contain sections starting with the word `state` and the order number of the state it represents. There is one section for each node of the reachability graph. The second line of each section contains information about the state marking. In the given example, line `props p0*2` indicates that there are two tokens in place `p2` ($M(p_2) = 2$) and other places are empty. The third line of a section describes the active transitions in the given state. For example, line `trans t0/1 t1/2` means that there are two active transitions in the state – `t0` and `t1`. The former leads to state `s1`, and the latter to state `s2`. The letter *w* denotes the infinity. If a coverability graph does not contain the infinity symbol, then the coverability and reachability graphs are identical.

The PetriNet2NuSMV tool parses an input *kts* file and generates the system model in the SMV language. To this end the tool uses the designed translation algorithm. The algorithm is presented in Figure 4. A fragment of the SMV file generated for reachability graph from Fig. 2 is given below.

Listing 1.1. A fragment of SMV file generated for reachability graph from Fig. 2.

```
MODULE main
VAR
  s: {s0, s1, s2, s3, s4, s5};
  p0 : 0..1000;
  p1 : 0..1000;
  p2 : 0..1000;
ASSIGN
  init(s) := s0;
  next(s) := case
    s = s0 : {s1, s2};
    s = s1 : {s3, s4};
    s = s2 : s4;
    s = s3 : s5;
    s = s4 : {s0, s5};
    s = s5 : {s1, s2};
  esac;
  p0 := case
    s = s0 : 2;
    s = s1 : 1;
    s = s4 : 1;
    TRUE : 0;
  esac;
  ...
```

```
1: add MODULE main statement
2: add VAR keyword
3: for all $M_i \in \mathcal{R}(M_0)$ do
4:     add si to the set of states s                         ▷ s : {s0, s1, ...};
5: end for
6: for all $p_i \in P$ do
7:     if $\forall_{M_j \in \mathcal{R}(M_0)} M_j(p_i) \leqslant 1$ then
8:         add Boolean variable pi                            ▷ pi : boolean;
9:     else
10:        add bounded Integer variable pi
                ▷ pi : 0..k; where $\forall_{p_i \in P, M_j \in \mathcal{R}(M_0)} M_j(p_i) \leqslant k$
11:    end if
12: end for
13: add ASSIGN keyword
14: init s variable                                           ▷ init(s) = s0;
15: open transition relation switch statement                 ▷ next(s) := case
16: for all si ∈ s do
17:     add case s = si
18:     for all sj ∈ s do
19:         if $\exists_{t \in T} M_i \xrightarrow{t} M_j$ then
20:             add sj to si successors list
21:         end if
22:     end for                                  ▷ s = si : {sj1, sj2, ...};
23: end for
24: close transition relation switch statement                ▷ esac;
25: for all $p_i \in P$ do
26:     open labelling function switch statement              ▷ pi := case
27:     for all sj ∈ s do
28:         if $M_j(p_i) > 0$ then
29:             assign m to case s = sj, where m $= M_j(p_i)$   ▷ s = sj : m;
30:         end if
31:     end for
32:     if pi is of Boolean type then
33:         set default value to FALSE                        ▷ TRUE: FALSE;
34:     else
35:         set default value to 0                            ▷ TRUE: 0;
36:     end if
37:     close labelling function switch statement             ▷ esac;
38: end for
```

Fig. 4. PT-net to NuSMV translation algorithm

Listing 1.2. Examples of LTL and CTL formulae for the model from Listing 1.1.

```
CTLSPEC AG EF (p0 = 2 & p1 = 0 & p2 = 0)
LTLSPEC G (p0 <= 2 & p1 <= 2 & p2 <= 2)
LTLSPEC G (p0 + p1 + p2 = 2)
LTLSPEC G (p0 = 2 -> X (p0 != 2))
CTLSPEC AG (p1 = 2 -> AX (EF (p1 = 2)))
```

Examples of LTL and CTL formulae for the model from Listing 1.1 are shown in Listing 1.2. For example, the first one denotes that the corresponding PT-net is reversible, the second one denotes that the net is 2-bounded, the third denotes that the net is conservative etc. It is easy to check that all these formulae hold for the model. It is not so obvious if a reachability graph contains thousands of nodes and arcs.

4 Coloured Petri nets

Coloured Petri nets (CP-nets) combines the capabilities of PT nets with the capabilities of a high-level programming language [8]. CP-nets provide graphical notation typical for Petri nets, but net elements are described using CPN ML programming language, which is based on the functional programming language Standard ML [16].

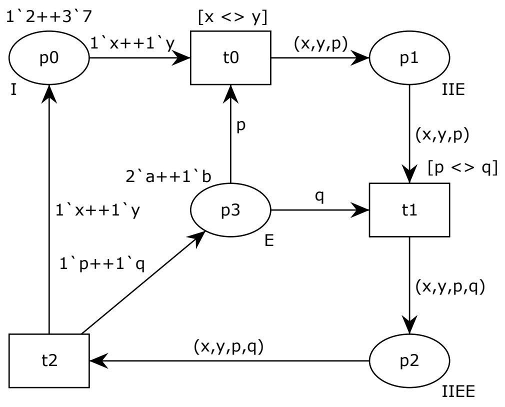
A *non-hierarchical CP-net* is a nine-tuple $\mathcal{N} = (P, T, A, \Sigma, V, C, G, E, I)$, where P, T and A have meaning as for PT-nets, Σ is a finite set of non-empty *colour sets* (types of tokens), V is a finite set of *variables* of types from Σ, $C\colon P \to \Sigma$ is a *colour set function* that assigns to each place the type of its tokens; G is a *guard function* that assigns a guard (Boolean condition) to each transition (the default value is *true*); E is an *arc expression function* that assigns to each arc an expression that evaluates to a multi-set of tokens of the type assigned to the arc place node; and I is an *initialization function* that assigns to each place an expression that evaluates to a multi-set of tokens of the type assigned to the place.

A marking of a place $p \in P$ of a CP-net is multi-set of tokens of type $C(p) \in \Sigma$ [8]. The initial marking is obtained by evaluating the initialization expressions. Due to the fact that arc expressions and guards may contain variables, it is necessary to assign (bind) some values to the variables to check if a transition is *enabled*. Let $Var(t)$ denote the set of variables occurring in arc expressions on the arcs connected to the transition and in the transition guard. A *binding* b of a transition t is a function that maps each variable $v \in Var(t)$ into a value of its type. A transition t is enabled if it is possible to construct such binding b that the guard $G(t)$ evaluates to true and each of the arc expressions evaluates to tokens, which are present on the corresponding input places. A pair (t, b) is called *binding element*. A firing of an enabled transition t removes $E(p,t)\langle b\rangle$ tokens from each input place p of t, and adds $E(t,p)\langle b\rangle$ tokens to each output place p of t, where $E(a)\langle b\rangle$ denotes the result of evaluating the arc expression $E(a)$ of an arc a in the binding b.

Sets $\mathcal{R}(M)$ and $\mathcal{L}(M)$ are defined as for PT-nets, but binding elements are taken into consideration instead of transitions. A *reachability graph* of a CP-net has a node for each reachable marking and an arc for each occurring binding element. An example of a CP-net is presented in Fig. 5. Its reachability graph is shown in Fig. 6.

For an effective modelling CP-nets enable to distribute parts of the net across multiple subnets called modules. The result of such an approach is a hierarchical CP-net [8]. A description of hierarchical CP-nets is out of the scope of the paper. From the presented algorithm point of view, it does not matter whether a reachability graph was generated for a hierarchical or non-hierarchical CP-net.

The most popular software for CP-nets modelling and reachability graphs generation is CPN Tools [9] and therefore, it was used in the approach presented in the paper.



```
colset E = with a | b;
colset I = int with 1..9;
colset IIE = product I*I*E;
colset IIEE = product I*I*E*E;
var x, y : I;
var p, q : E;
```

Fig. 5. An example of CP-net model.

$$\begin{tikzcd}
(\{7,7\},\emptyset,\{(2,7,a,b)\},\{a\})_{M_5} \arrow[dr, "{t2,[2/x,7/y,a/p,b/q]}"] & (\{7,7\},\{(7,2,b)\},\emptyset,\{a,a\})_{M_4} \arrow[r, "{t1,[7/x,2/y,b/p,a/q]}"'] & (\{7,7\},\emptyset,\{(7,2,b,a)\},\{a\})_{M_8} \arrow[dl, "{t2,[7/x,2/y,b/p,a/q]}"] \\
(\{7,7\},\{(2,7,a)\},\emptyset,\{a,b\})_{M_1} \arrow[u, "{t1,[2/x,7/y,a/p,b/q]}"] & (\{2,7,7,7\},\emptyset,\emptyset,\{a,a,b\})_{M_0} \arrow[l, "{t0,[2/x,7/y,a/p]}"] \arrow[u, "{t0,[7/x,2/y,b/p]}"] \arrow[r, "{t0,[2/x,7/y,b/p]}"'] \arrow[d, "{t0,[7/x,2/y,a/p]}"] & (\{7,7\},\{(2,7,b)\},\emptyset,\{a,a\})_{M_3} \arrow[d, "{t1,[7/x,2/y,b/p,a/q]}"] \\
(\{7,7\},\emptyset,\{(7,2,a,b)\},\{a\})_{M_6} \arrow[ur, "{t2,[7/x,2/y,a/p,b/q]}"] & (\{7,7\},\{(7,2,a)\},\emptyset,\{a,b\})_{M_2} \arrow[l, "{t1,[7/x,2/y,a/p,b/q]}"'] & (\{7,7\},\emptyset,\{(2,7,b,a)\},\{a\})_{M_7} \arrow[ul, "{t2,[7/x,2/y,b/p,a/q]}"]
\end{tikzcd}$$

Fig. 6. Reachability graph for the CP-net from Fig. 5.

It is a powerful and efficient tool for editing, simulation and analysis of coloured Petri nets. CPN Tools provides for the automatic reachability graph generation by taking advantage of its state space toolbox. In order to do this, few steps are required. The first step is to make use of *Calculate state space* feature followed by *Calculate SCC graph* operation. To actually generate nodes and arcs of a reachability graph, *Display partial state space* feature must be called on auxiliary text element containing short ML code: `EvalAllArcs(fn arc => arc)`. *Nodes in the list* option must be unchecked for this feature to work properly. Generated graph is saved with the model in a *cpn* file. Parsing process of a *cpn* file is relatively simple due to its XML structure. After parsing of a *cpn* file, PetriNet2NuSMV tool generates SMV output. The translation algorithm for CP-nets is presented in Figure 7. Illustrative parts of the generated SMV file for the reachability graph from Fig. 6 are presented in Listing 1.3.

```
 1: add MODULE main statement
 2: add VAR keyword
 3: for all $M_i \in \mathcal{R}(M_0)$ do
 4:     add si to the set of states s                      ▷ s : {s0, s1, ...};
 5: end for
 6: $L = \emptyset$                                         ▷ the set of defined variables' labels
 7: for all $p_i \in P$ do
 8:     for all colour $c_j \in M_k(p_i)$, where $M_k \in \mathcal{R}(M_0)$ do
 9:         create marking label $l_{temp}$ by concatenating pi and cj
10:         if $\nexists_{l \in L} l = l_{temp}$ then
11:             add bounded Integer variable l_temp
                        ▷ l_temp : 0..k; where $\forall_{p_i \in P, M_j \in \mathcal{R}(M_0)} |M_j(p_i)| \leqslant k$
12:             add l_temp to $L$
13:         end if
14:     end for
15: end for
16: add ASSIGN keyword
17: init s variable                                        ▷ init(s) = s0;
18: open transition relation switch statement              ▷ next(s) := case
19: for all si ∈ s do
20:     add case s = si
21:     for all sj ∈ s do
22:         if $\exists_{t \in T} : M_i \xrightarrow{t} M_j$ then
23:             add sj to si successors list
24:         end if
25:     end for                                            ▷ s = si : {sj1, sj2, ...};
26: end for
27: close transition relation switch statement             ▷ esac;
28: for all $l_i \in L$ do
29:     open labelling function switch statement           ▷ li := case
30:     for all sj ∈ s do
31:         $c = |\{x \in M_j(place(l_i)) : x = colour(l_i)\}|$
32:         if $c > 0$ then
33:             assign $c$ to case s = sj                  ▷ s = sj : c;
34:         end if
35:     end for
36:     set default value to 0                             ▷ TRUE: 0;
37:     close labelling function switch statement          ▷ esac;
38: end for
```

Fig. 7. CP-net to NuSMV translation algorithm

Listing 1.3. Selected fragments of SMV file generated for reachability graph from Fig. 6.

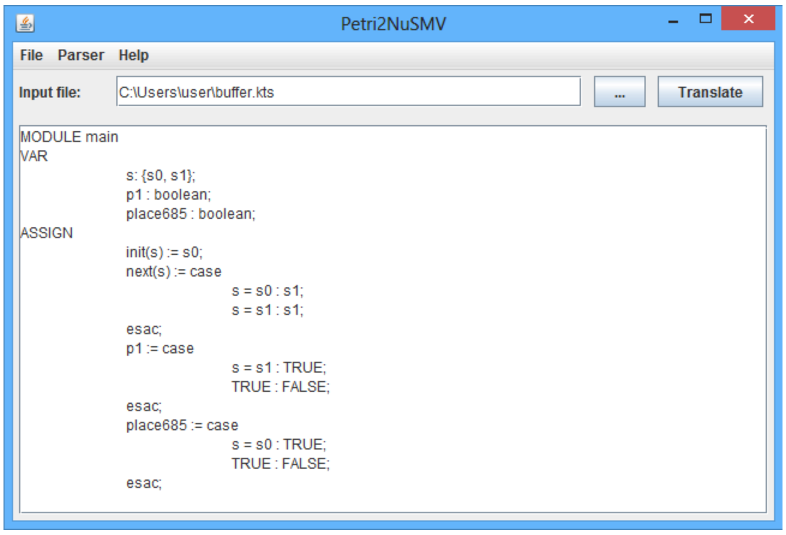
```
MODULE main
VAR
  s: {s1, s2, s3, s4, s5, s6, s7, s8, s9};
  p0_7 : 0..3;
  ...
```

```
ASSIGN
  init(s) := s1;
  next(s) := case
    s = s1 : {s5, s4, s3, s2};
    s = s2 : s6;
    ...
  esac;
  p0_7 := case
    s = s5 : 2;
    s = s1 : 3;
    s = s6 : 2;
    ...
    TRUE : 0;
  esac;
```

In order to make this notation clear, the meaning of labels will be shortly explained. The value of `p0_7` variable denotes the number of tokens of value 7 in place `p0`. Bearing this in mind, `s = s5 : 2` denotes: There are 2 tokens of value 7 in place `p_0` in state `s5`.

5 Usability studies

Along with the presented algorithm, fully functional tool called PetriNet2NuSMV has been developed. In accordance with original ideas this tool allows to translate both *kts* and *cpn* files into *smv* files. As a result, users can easily validate the modelled system using LTL and CTL temporal logic formulae.



C:\Users\user\buffer.kts

```
MODULE main
VAR
        s: {s0, s1};
        p1 : boolean;
        place685 : boolean;
ASSIGN
        init(s) := s0;
        next(s) := case
                s = s0 : s1;
                s = s1 : s1;
        esac;
        p1 := case
                s = s1 : TRUE;
                TRUE : FALSE;
        esac;
        place685 := case
                s = s0 : TRUE;
                TRUE : FALSE;
        esac;
```

Fig. 8. A PetriNet2NuSMV tool screenshot.

In order to facilitate the usage of the tool it was developed with simple graphical user interface (see Fig. 8). It allows to configure the parser, load an input file and save the generated output to the *smv* file. The application is written in Java using Swing library so it can serve both Linux and Windows users.

Table 1. Translation results for illustrative systems modelled with Tina and CPN Tools.

Modelled system	Petri net type	Reachability graph nodes count	Translation time [ms]
Dining philosophers problem	Place/transition Petri net	11	13
Producer-consumer problem	Coloured Petri net	12	134
Combinational logic	Place/transition Petri net	87	51
Simple protocol	Coloured Petri net	2012	20174

A summary of translation results for illustrative systems modelled in Tina and CPN Tools is presented in Table 1. The table contains translation times for four Petri net models with varying complexity and size of their reachability graphs. The measured times proved to be entirely satisfactory. NuSMV code is generated visibly faster for PT-nets. Nonetheless, even complex graph of CP-net model consisting of more than 2000 nodes and 20 000 arcs is translated in approximately 20 seconds. Considering the large size of the CPN Tools file that needs to be parsed, which is more then 20 MB, the result can be considered as adequate. Manual approach to NuSMV code creation for the mentioned Petri net model is practically impossible.

6 Summary

Algorithm for translation of reachability graphs for place-transition and coloured Petri nets into the NuSMV language has been presented in the paper. Both of these algorithms have been implemented as the core of PetriNet2NuSMV software. The tool has been tested against reachability graphs of different sizes and complexity and proved to be quite swift and efficient. PetriNet2NuSMV enables users to translate reachability graphs generated with TINA and CPN Tools environments into a NuSMV model automatically. Thus, a Petri model can be verified using model checking techniques without necessity of learning any additional language for the specification of requirements. Moreover, the presented algorithms for low and high level Petri nets can be adapt to other classes of Petri nets.

References

1. PROD tool home page (2007), `http://www.tcs.hut.fi/Software/prod/`
2. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press, London, UK (2008)
3. Berthomieu, B., Ribet, P.O., Vernadat, F.: The tool TINA – construction of abstract state spaces for Petri nets and time Petri nets. International Journal of Production Research 42(14), 2741–2756 (2004)

4. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV version 2: An opensource tool for symbolic model checking. In: Proceedings of International Conference on Computer-Aided Verification (CAV 2002). LNCS, vol. 2404. Springer-Verlag, Copenhagen, Denmark (2002)
5. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NUSMV: a new symbolic model checker. International Journal on Software Tools for Technology Transfer 2(4), 410–425 (2000)
6. Clarke, E., Grumberg, O., Peled, D.: Model Checking. The MIT Press, Cambridge, Massachusetts (1999)
7. Emerson, E.: Temporal and modal logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 995–1072. Elsevier Science (1990)
8. Jensen, K., Kristensen, L.: Coloured Petri nets. Modelling and Validation of Concurrent Systems. Springer, Heidelberg (2009)
9. Jensen, K., Kristensen, L., Wells, L.: Coloured Petri nets and CPN Tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer 9(3–4), 213–254 (2007)
10. Kripke, S.: A semantical analysis of modal logic I: normal modal propositional calculi. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 9, 67–96 (1963), announced in *Journal of Symbolic Logic*, **24**, 1959, p. 323
11. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
12. Schröter, C., Schwoon, S., Esparza, J.: The model-checking kit. In: Applications and Theory of Petri Nets 2003, Lecture Notes in Computer Science, vol. 2679, pp. 463–472. Springer (2003)
13. Stehno, C.: PEP Version 2.0. In: Tool demonstration ICATPN 2001 (2001)
14. Szpyrka, M.: Analysis of RTCP-nets with reachability graphs. Fundamenta Informaticae 74(2–3), 375–390 (2006)
15. Szpyrka, M.: Analysis of VME-Bus communication protocol – RTCP-net approach. Real-Time Systems 35(1), 91–108 (2007)
16. Ullman, J.: Elements of ML programming (ML97 ed.). Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1998)
17. Wolf, K.: Generating Petri net state spaces. In: Petri Nets and Other Models of Concurrency – ICATPN 2007, Lecture Notes in Computer Science, vol. 4546, pp. 29–42. Springer (2007)