Iwona GROBELNA
UNIWERSYTET ZIELONOGÓRSKI,
Podgórna 50, 65-246 Zielona Góra

Regułowa reprezentacja interpretowanych sieci Petriego sterowania dla potrzeb syntezy i weryfikacji

Mgr inż. Iwona GROBELNA

Absolwentka Wydziału Elektrotechniki, Informatyki i Telekomunikacji Uniwersytetu Zielonogórskiego oraz Fachhochschule Giessen-Friedberg (Niemcy). Od marca 2008 zatrudniona na stanowisku asystenta w Instytucie Informatyki i Elektroniki Uniwersytetu Zielonogórskiego. Studentka studiów doktoranckich. Zainteresowania naukowe obejmują metody weryfikacji specyfikacji systemów osadzonych. Członek Polskiego Towarzystwa Informatycznego.

e-mail: I.Grobelna@iie.uz.zgora.pl

Streszczenie

Artykuł proponuje regułowy sposób reprezentacji interpretowanych sieci Petriego sterowania w logice temporalnej. Sposób ten jest przydatny zarówno do formalnej weryfikacji modelowej, jak i do automatycznej syntezy logicznej z wykorzystaniem języków opisu sprzętu (Verilog, VHDL) jako rekonfigurowalny sterownik logiczny lub PLC. Sieci Petriego weryfikowane są zwykle tylko pod kątem właściwości strukturalnych. Technika weryfikacji modelowej pozwala na weryfikację właściwości behawioralnych opisujących zachowanie projektowanego systemu.

Słowa kluczowe: weryfikacja modelowa, Interpretowane Sieci Petriego Sterowania, synteza logiczna.

Rule-based representation of Control Interpreted Petri Nets for synthesis and verification purposes

Abstract

The paper presents a novel idea of Control Interpreted Petri Nets representation in temporal logic. The proposed logic representation is suitable both for formal model checking and automatic synthesis using hardware description languages (Verilog, VHDL). Petri Nets [1, 2, 3] are currently used in industry, i.e. by logic controller design [4]. Dedicated tools for creating Petri Nets support verification against structural properties. Behavioral properties are also of great importance, however they are rarely considered. Model checking technique [5] allows for verification of properties describing behavior of designed system. So far, there have been some approaches to verify (validate) specification by means of Petri Nets [6, 7, 8, 9], by means of UML diagrams [10] or logic controller programs in ST language [11]. However, none of them have addressed Control Interpreted Petri Nets focused on RTL level. The proposed rule-based representation of Control Interpreted Petri Nets (logical model in Figs. 2-5) is easy to formally verify (model description for NuSMV model checker [13] in Fig. 6-10), as well as to synthezise (VHDL model in Figs. 11-13) as a reconfigurable logic controller or PLC. Verified behavioral specification in temporal logic [14] is an abstract program of matrix reconfigurable logic controller functionality, and logic controller program (implementation) satisfies its primary specification. The logical model built from Control Interpreted Petri Net describes it in a strict and short form.

Keywords: model checking, Control Interpreted Petri Nets, logical synthesis.

1. Wstęp

Interpretowane sieci Petriego sterowania [1, 2, 3] jako formalny zapis specyfikacji działania sterownika logicznego [4] są powszechnie wykorzystywane w przemyśle. Istnieją dedykowane narzędzia do projektowania sieci Petriego, które ułatwiają pracę inżynierom. Część z nich wspiera dodatkowo weryfikację utworzonych sieci pod kątem właściwości strukturalnych. Właściwości behawioralne są nie mniej istotne, choć często pomijane.

Problemem jest zatem weryfikacja właściwości dotyczących działania projektowanego systemu. Po utworzeniu modelu w jednym z języków opisu sprzętu (np. Verilog, VHDL) możliwe jest przeprowadzenie symulacji, pozwoli ona na wykrycie części błędów przed fizyczną implementacją urządzenia. Sprawdzone zostaną jednak tylko przykładowe sytuacje.

Technika weryfikacji modelowej [5] pozwala na weryfikację właściwości opisujących zachowanie projektowanego systemu. Na etapie projektowania można zatem sprawdzić, czy zdefiniowany model spełnia stawiane mu wymagania.

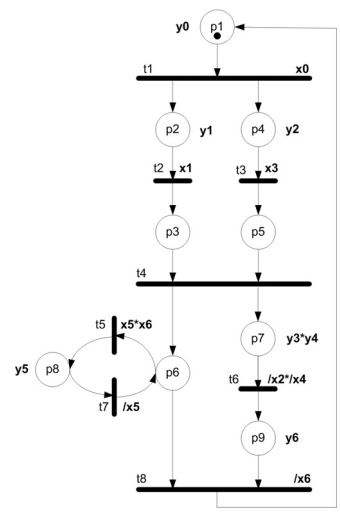
W literaturze podjęte zostały próby weryfikacji sieci Petriego. Jednakże, nie skupiają się one na interpretowanych sieciach Petriego sterowania przedstawionych na poziomie RTL. Rozpatrywane były czasowe sieci Petriego [6, 7, 8], czy też synchroniczne interpretowane sieci Petriego [9]. Podejmowane były próby weryfikacji diagramów UML [10], a także programów dla sterowników PLC zapisanych w postaci tekstowego języka ST [11].

Prezentowane rozwiązanie proponuje zapisanie interpretowanej sieci Petriego sterowania jako modelu logicznego zawierającego zestaw reguł. Taki regułowy sposób reprezentacji nadaje się zarówno do formalnej weryfikacji, jak i do syntezy logicznej jako rekonfigurowalny sterownik logiczny lub PLC. Dużą zaletą rozwiązania jest fakt, że model syntezowalny opisany jest w podobny sposób co model weryfikowalny. Po pomyślnej weryfikacji modelowej osiągamy zatem zweryfikowaną specyfikację behawioralną w języku logiki temporalnej, która będzie abstrakcyjnym programem funkcjonowania matrycowego rekonfigurowalnego sterownika logicznego. Program sterownika logicznego (implementacja) będzie więc poprawny względem jego pierwotnej specyfikacji.

Interpretowana sieć Petriego sterowania jest specyfikacją wyjściową do opisywanego procesu weryfikacji i syntezy. Możliwe jest jednak użycie także innych form specyfikacji, np. diagramów aktywności języka UML 2.x [12].

2. Model logiczny

Proponowany model logiczny wykorzystywany jest zarówno do celów syntezy, jak i do weryfikacji modelowej. Stanowi format pośredni opisujący zachowanie projektowanego sterownika logicznego. Model zawiera definicje zmiennych wraz z wartościami jakie mogą przyjmować. Miejsca, sygnały wejściowe oraz sygnały wyjściowe sieci Petriego (przykładowa sieć na rys. 1) są bezpośrednio odwzorowane w modelu logicznym jako zmienne (rys. 2).



Rys. 1. Przykładowa interpretowana sieć Petriego sterowania
Fig. 1. A sample Control Interpreted Petri Net

Każde miejsce, sygnał wejściowy i sygnał wyjściowy traktowane są jako osobna zmienna typu logicznego. Zdefiniowane są także początkowe wartości zmiennych.

```
VARIABLES
places:  p1, p2, p3, p4, p5, p6, p7, p8, p9
inputs:  x0, x1, x2, x3, x4, x5, x6
outputs: y0, y1, y2, y3, y4, y5, y6

INITIALLY
 p1; !p2; !p3; !p4; !p5; !p6; !p7; !p8; !p9;
!x0; !x1; !x2; !x3; !x4; !x5; !x6;
 y0; !y1; !y2; !y3; !y4; !y5; !y6;
```

Rys. 2. Zmienne i ich początkowe wartości w modelu logicznym
Fig. 2. Variables and their initial values in logical model

Model logiczny poza definicją zmiennych zawiera także zbiór reguł, które opisują zachowanie projektowanego systemu i określają zmiany wartości zmiennych z upływem czasu. Tranzycje interpretowane są jako wspomniane zbiory reguł (rys. 3).

```
TRANSITIONS
t1: p1 & x0 -> X (!p1 & p2 & p4);
t2: p2 & x1 -> X (!p2 & p3);
t3: p4 & x3 -> X (!p4 & p5);
t4: p3 & p5 -> X (!p3 & !p5 & p6 & p7);
t5: p6 & x5 & x6 -> X (!p6 & p8);
t6: p7 & !x2 & !x4 -> X (!p7 & p9);
t7: p8 & !x5 -> X (p6 & !p8);
t8: p6 & p9 & !x6 -> X (!p6 & !p9 & p1);
```

Rys. 3. Zbiór reguł w modelu logicznym
Fig. 3. Set of rules in logical model

Sygnały wyjściowe są aktywne przy aktywnym znakowaniu odpowiednich miejsc. Sygnały są więc przypisane są do miejsc (rys. 4).

```
OUTPUTS
p1 -> y0;
p2 -> y1;
p4 -> y2;
p7 -> y3 & y4;
p8 -> y5;
p9 -> y6;
```

Rys. 4. Sygnały wyjściowe w modelu logicznym
Fig. 4. Output signals in logical model

Model logiczny może zostać poddany syntezie jako rekonfigurowalny sterownik logiczny lub PLC. Najważniejsze stają się wtedy sygnały wyjściowe sterownika, które kontrolują zachowanie systemu i sterują procesami.

Zmiany sygnałów wejściowych (rys. 5) definiowane są w modelu logicznym. Są one wykorzystywane jednak tylko przy weryfikacji modelowej (tworzeniu opisu modelu). Model w języku opisu sprzętu nie zmienia wartości sygnałów wejściowych.

```
INPUTS
p1 -> !x0 | x0;
p2 -> !x1 | x1;
p4 -> !x3 | x3;
p6 -> (!x5 | x5) & (!x6 | x6);
p7 -> (!x2 | x2) & (!x4 | x4);
p8 -> !x5 | x5;
p6 & p9 -> !x6 | x6;
```

Rys. 5. Sygnały wejściowe w modelu logicznym
Fig. 5. Input signals in logical model

3. Weryfikacja modelowa

Model logiczny utworzony na podstawie interpretowanej sieci Petriego sterowania przekształcany jest do formatu wejściowego narzędzia weryfikującego NuSMV (wersja 2.5.2) [13]. Opracowane zostały następujące reguły:

(a) Każde miejsce, sygnał wejściowy i sygnał wyjściowy jest zmienną typu logicznego (rys. 6)
(b) Każda ze zdefiniowanych zmiennych przyjmuje pewną wartość początkową (rys. 7)
(c) Każde miejsce zmienia swoje znakowanie zgodnie z regułami określonymi przez tranzycje (rys. 8)
(d) Każdy sygnał wyjściowy zmienia swoją wartość zgodnie z aktualnie aktywnymi miejscami (rys. 9)
(e) Każdy sygnał wejściowy zmienia losowo swoją wartość (rys. 10)

Podpunkty (a) i (b) (rys. 6, 7) odpowiadają fragmentowi modelu logicznego z rys. 2.

```
VAR
   p1 : boolean;  ...   p9 : boolean;
   x0 : boolean;  ...   x6 : boolean;
   y0 : boolean;  ...   y6 : boolean;
```

Rys. 6. Zmienne w opisie modelu NuSMV
Fig. 6. Variables in NuSMV model description

```
init(p1) := TRUE;   init(p2) := FALSE;  ...
init(x0) := FALSE;  init(x1) := FALSE;  ...
init(y0) := TRUE;   init(y1) := FALSE;  ...
```

Rys. 7. Początkowe wartości zmiennych w opisie modelu NuSMV
Fig. 7. Initial values of variables in NuSMV model description

Podpunkt (c) (rys. 8) odpowiada fragmentowi modelu logicznego z rys. 3. Warunki zmian pomiędzy miejscami (przepływ tokenów) występują parami – zawsze w poprzednim i następnym stanie. Jedno z miejsc przyjmuje wtedy token, a drugie usuwa.

```
next(p1) := case
 p1 & x0        : FALSE;
 p6 & p9 & !x6  : TRUE;
 TRUE           : p1;
esac;
```

```
next(p9) := case
 p7 & !x2 & !x4 : TRUE;
 p6 & p9 & !x6  : FALSE;
 TRUE           : p9;
esac;
```

Rys. 8. Zmiana znakowania miejsc w opisie modelu NuSMV
Fig. 8. Changes of places marking in NuSMV model description

Podpunkt (d) (rys. 9a) odpowiada fragmentowi modelu logicznego z rys. 4. Podpunkt (e*)* (rys. 9b) odpowiada fragmentowi modelu logicznego z rys. 5. Dostępne są tylko wartości oczekiwane związane z konkretnymi miejscami sieci Petriego lub adekwatne do sytuacji.

```
next(y1) := case
 p2   : TRUE;
 TRUE : FALSE;
esac;
```

(a)

```
next(x0) := case
 p1   : {FALSE, TRUE};
 TRUE : FALSE;
esac;
```

(b)

Rys. 9. Zmiana wartości sygnałów wyjściowych (a) i wejściowych (b) w opisie modelu NuSMV
Fig. 9. Changes of output (a) and input (b) signals values in NuSMV model description

Właściwości, jakie projektowany system ma spełniać, definiowane są przy wykorzystaniu logiki temporalnej [14]: logiki liniowej LTL lub logiki rozgałęzionej CTL. Właściwości określają zarówno wymagania dotyczące bezpieczeństwa, jak i wymagania dotyczące żywotności. Przykładowym wymaganiem dotyczącym żywotności jest, że zawsze, gdy sygnał wyjściowy *y0* jest aktywny i wystąpi sygnał wejściowy *x0*, to w końcu aktywne będą sygnały wyjściowe *y1* i *y2* (rys. 10a). Przykładowym wymaganiem dotyczącym bezpieczeństwa jest, że nigdy nie może wystąpić sytuacja, w której jednocześnie aktywne będą miejsca *p1* i *p8* sieci Petriego (rys. 10b).

```
(a)  LTLSPEC G (y0 & x0 -> F (y1 & y2));
(b)  LTLSPEC G !(p1 & p8);
```

Rys. 10. Wymaganie dotyczące żywotności (a) oraz bezpieczeństwa (b)
Fig. 10. Liveness (a) and safety (b) requirement

4. Synteza logiczna

Po pomyślnej weryfikacji modelowej model logiczny może być następnie przekształcony do modelu syntezowalnego, przykładowo w języku VHDL. Opracowane zostały następujące zasady transformacji modelu logicznego do języka VHDL:

(a) Każde miejsce jest sygnałem wewnętrznym układu typu `std_logic` (rys. 13)
(b) Każdy sygnał wejściowy/wyjściowy jest portem wejściowym/wyjściowym do układu typu `std_logic` (rys. 13)
(c) Każdy ze zdefiniowanych sygnałów wewnętrznych (miejsc sieci Petriego) przyjmuje wartość początkową, ustawianą przy narastającym zboczu zegara (`rising_edge(Clk)`) i aktywnym sygnale resetującym (`Reset = '1'`)
(d) Każde miejsce zmienia swoje znakowanie zgodnie z regułami określonymi przez tranzycje (rys. 14):
   - tranzycja, która aktualnie jest odpalana, zmienia znakowanie miejsc wejściowych oraz wyjściowych
   - dla każdej tranzycji podana jest sytuacja, w której nie jest ona odpalona (miejsca wejściowe zawierają token, ale nie jest spełniony warunek odpalenia) – miejsca wejściowe utrzymują wtedy swoje znakowanie
(e) Sygnały wejściowe są pomijane w modelu do syntezy; zmiany sygnałów przychodzą z zewnątrz
(f) Każdy sygnał wyjściowy zmienia się w zależności od aktywnego miejsca*;* sygnały wyjściowe są aktywne przy aktywnym znakowaniu odpowiadających im miejsc (rys. 15)

Podpunkty (a) i (b) (rys. 11) odpowiadają fragmentowi modelu logicznego z rys. 2. Dodatkowym portem wejściowym jest sygnał zegarowy (`Clk`) oraz sygnał resetowania układu (`Reset`).

```
entity Sterownik is
port (Clk, Reset: in STD_LOGIC;
      x0, x1, x2, x3, x4, x5, x6: in STD_LOGIC;
      y0, y1, y2, y3, y4, y5, y6: out STD_LOGIC);
end Sterownik;

architecture arch of Sterownik is
       signal p1, p2, p3, p4, p5, p6, p7, p8, p9 :
                                          std_logic;
```

Rys. 11. Zmienne w modelu VHDL
Fig. 11. Variables in VHDL model

Podpunkt (d) (rys. 12) odpowiada fragmentowi modelu logicznego z rys. 3. Przyjęte rozwiązanie odpowiada metodzie tworzenia modelu w VHDL zorientowanej na miejsca i tranzycje [15]. Alternatywnym rozwiązaniem może być metoda zorientowana na tranzycje, która stanowi obiekt dalszych badań.

```
  if p1='1' and x0='1' then -- t1
      p1 <= '0';
      p2 <= '1';
      p4 <= '1';
  end if;
  if p1='1' and x0='0' then p1 <= '1'; end if;
```

Rys. 12. Zmiana znakowania miejsc w modelu VHDL
Fig. 12. Changes of places marking in VHDL model

Podpunkt (f) (rys. 13) odpowiada fragmentowi modelu logicznego z rys. 4.

```
y0 <= p1;   y1 <= p2;   y2 <= p4;   y3 <= p7;
y4 <= p7;   y5 <= p8;   y6 <= p9;
```

Rys. 13. Sygnały wyjściowe w modelu VHDL
Fig. 13. Output signals in VHDL model

Utworzony model w języku VHDL może zostać wstępnie przesymulowany w środowisku Active HDL. Następnie możliwe jest przeprowadzenie syntezy przygotowanego układu w środowisku XILINX ISE.

5. Wnioski

W artykule zaprezentowane zostało nowatorskie podejście do weryfikacji modelowej interpretowanych sieci Petriego sterowania. Na podstawie istniejącej sieci Petriego tworzony jest model logiczny na poziomie RTL. Model taki nadaje się zarówno do weryfikacji modelowej pod kątem behawioralnych właściwości jakie projektowany system ma spełniać, jak również do syntezy w postaci rekonfigurowalnego sterownika logicznego lub PLC. Otrzymany program sterownika logicznego (jego implementacja) będzie zatem poprawny względem jego pierwotnej (zweryfikowanej) specyfikacji.

Autor jest stypendystą w ramach Poddziałania 8.2.2 „Regionalne Strategie Innowacji", Działania 8.2 „Transfer wiedzy", Priorytetu VIII „Regionalne Kadry Gospodarki" Programu Operacyjnego Kapitał Ludzki współfinansowanego ze środków Europejskiego Funduszu Społecznego Unii Europejskiej i z budżetu państwa.

6. Literatura

[1] Adamski M.: A rigorous design methodology for reprogrammable logic controllers, The International Workshop on Discrete-Event System Design, DESDes'01, June 2001, Przytok.
[2] David R., Alla H.: Petri Nets & Grafcet. Tools for modelling discrete event systems, Prentice Hall 1992.
[3] Adamski M., Karatkevich A., Węgrzyn M. (ed.): Design of embedded control systems, Springer 2005.
[4] Gomes L., Barros J.P., Costa A.: Modeling formalisms for embedded system design, Embedded Systems Handbook, Taylor & Francis Group, LLC, 2006.
[5] Emerson E.A.: The beginning of model checking: a personal perspective, Lecture Notes in Computer Science, 25 Years of Model Checking: History, Achievements, Perspectives, 2008, str. 27 – 45.
[6] Berthomieu B., Peres F., Vernadat F.: Model checking bounded prioritized time Petri nets, ATVA'07 Proceedings of the 5th international conference on Automated technology for verification and analysis, 2007.
[7] Cortés L. A., Eles P., Peng Z.: Formal coverification of embedded systems using model checking, Proceesings of the 26th EUROMICRO Conference (EUROMICRO'00), 2000 IEEE.
[8] Penczek W., Półrola A.: Advances in verification of Time Petri Nets and timed automata, Springer 2006.
[9] Ribeiro Ó. R., Fernades J.M.: Translating synchronous Petri Nets into PROMELA for verifying behavioural properties, IEEE 2007.
[10] Niewiadomski A., Penczek W., Szreter M.: Towards checking parametric reachability for UML state machines, Novosibirsk University 2009, Proc of. 7th International Andrei Ershov Memorial Conference Perspectives of System Informatics, str. 229-240.
[11] Gourcuff V., De Smet O., Faure J.M.: Efficient representation for formal verification of PLC programs, Discrete Event Systems, 2006, 8th International Workshop, pp. 182 – 187.
[12] Grobelna I., Grobelny M., Adamski M.: Petri Nets and activity diagrams in logic controller specification - transformation and verification, Mixed Design of Integrated Circuits and Systems - MIXDES 2010, str. 607 – 612.
[13] Cavada R. et al.: NuSMV 2.5 user manual, dostępne na http://nusmv.fbk.eu/
[14] Ben-Ari M.: Logika matematyczna w informatyce. Klasyka informatyki. Wydawnictwa Naukowo-Techniczne, Warszawa 2005.
[15] Węgrzyn M.: Modelowanie sieci Petriego w języku VHDL, Przegląd Elektrotechniczny, 2010, nr 1, str. 212 – 216.

otrzymano / received: 13.05.2011
przyjęto do druku / accepted: 04.07.2011 *artykuł recenzowany*