Iwona GROBELNA
UNIWERSYTET ZIELONOGÓRSKI, WYDZIAŁ ELEKTROTECHNIKI, INFORMATYKI I TELEKOMUNIKACJI

Formalna weryfikacja maszyny stanów z wykorzystaniem logiki temporalnej

Mgr inż. Iwona GROBELNA

W roku 2007 ukończyła studia na Wydziale Elektrotechniki, Informatyki i Telekomunikacji Uniwersytetu Zielonogórskiego. Absolwentka Zintegrowanych Studiów Zagranicznych Uniwersytetu Zielonogórskiego i 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.

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

Streszczenie

Artykuł przedstawia koncepcję specyfikacji współbieżnego procesu sterowania cyfrowego za pośrednictwem diagramów algorytmicznych maszyn stanów ASM w języku aprobowanym przez profesjonalne narzędzie model checker. Specyfikacja może zostać następnie formalnie zweryfikowana pod kątem wymagań stawianych projektowanemu systemowi. Lista wymagań tworzona jest przy wykorzystaniu liniowej logiki temporalnej LTL. Formalna weryfikacja Model Checking polega na sprawdzeniu, czy model systemu spełnia stawiane mu wymagania. W przypadku wykrycia niespójności generowany jest odpowiedni kontrprzykład.

Słowa kluczowe: algorytmiczne maszyny stanów ASM, logika temporalna, formalna weryfikacja specyfikacji, technika Model Checking.

Formal verification of a state machine with use of temporal logic

Abstract

The paper presents the formal specification method of concurrent control processes in form of algorithmic state machines ASM [5] in a language accepted by a professional model checker tool NuSMV. Basing on linear temporal logic LTL [7, 8, 9, 16] a requirement list (Fig. 6) for the system model is prepared. Formal verification Model Checking [17, 19] consists in comparison of the model description and the requirements list. If some requirements cannot be fulfilled, the appropriate counterexample is generated (Fig. 7), which allows localizing the error source. The ASM diagrams (Fig. 4) are fully determined, but they do not support modularity, that is why they are not well suited for specification of concurrent controlling processes. The paper includes a short introduction to the theory of algorithmic state machines ASM (Section 2), temporal logic (Section 3) and model checking technique (Section 4). The proposed solution is presented on an example (Section 5) of the process of controlling (partially concurrent) movements of two vehicles (Fig. 2). The formal verification method of the ASM diagrams with its advantages and disadvantages as well as the general conclusions are given at the end of the paper (Section 6).

Keywords: algorithmic state machines ASM, temporal logic, formal verification of specification, Model Checking technique.

1. Wprowadzenie

Projekt systemu informatycznego jest jednym z pierwszych etapów jego powstawania i stanowi fundament dalszych prac. Pojawia się tutaj wiele zagrożeń, które wcześnie rozpoznane pozwolą zaoszczędzić czas i pieniądze. Przede wszystkim powinny zostać zdefiniowane wymagania stawiane tworzonemu systemowi. W przypadku, gdy ich lista będzie niekompletna, system może funkcjonować zgodnie z wymaganiami, lecz nie spełniać wszystkich swoich zadań lub pewne z nich wykonywać nie do końca tak, jak by sobie tego życzył przyszły użytkownik. Nieformalne wymagania mogą być również niejednoznaczne, co może być źródłem błędu. Dlatego tak ważna jest formalna specyfikacja wymagań systemu obejmująca zarówno aspekty funkcjonalne jak i te niefunkcjonalne (dotyczące jakości produktu, wydajności, bezpieczeństwa, itd.).

Również model systemu powinien być kompletny i poprawnie zdefiniowany. Implementacja bazująca na niepoprawnej specyfikacji (a więc i końcowy produkt) także będzie niepoprawna. Wtedy cały proces tworzenia produktu zostanie powtórzony lub też, co gorsze, w przypadku nie wykrycia błędu końcowy produkt zostanie wypuszczony na rynek.

Błędy powstałe na etapie projektowania systemu mogą zostać wykryte przy wykorzystaniu formalnych metod weryfikacji [1]. Ich przegląd przedstawiony jest w pracy [2]. Prezentowana w dalszej części pracy technika Model Checking pozwala na automatyczną weryfikację behawioralnej specyfikacji systemu przy wykorzystaniu narzędzi komputerowego wnioskowania model checker.

Rozdział 2 obejmuje krótkie wprowadzenie do algorytmicznych maszyn stanów. Rozdział 3 zawiera podstawy logiki temporalnej. Rozdział 4 przedstawia formalną metodę weryfikacji Model Checking oraz prezentuje dostępne narzędzia wnioskowania komputerowego w logice temporalnej. Rozdział 5 wprowadza przykład formalnej weryfikacji algorytmicznej maszyny stanów opisującej współbieżny proces sterowania. Rozdział 6 zawiera propozycję wykorzystania innego sposobu opisu maszyny stanów oraz podsumowanie omówionych aspektów i wnioski.

2. Algorytmiczne maszyny stanów

Jednym z modeli formalnej specyfikacji systemów sterowania jest algorytmiczna maszyna stanów ASM (ang. *Algorithmic State Machine*), obok innych modeli takich jak np. skończony automat cyfrowy FSM wraz z jego rozszerzeniami czy też sieci Petriego [3]. Inne formalne modele projektowania systemów osadzonych omówione są w [4]. Aktualnie model ten jest wykorzystywany m.in. przy projektowaniu systemów cyfrowych, algorytmach, systemach sterowania opartych na automatach [5] oraz w specyfikacji sterowników logicznych. Utworzony diagram może zostać następnie przetransformowany do języków opisu sprzętu (np. Verilog, VHDL) oraz przesymulowany [6].

Algorytmiczne maszyny stanów pozwalają na przedstawienie sekwencji następujących po sobie stanów w przejrzysty sposób. Taka reprezentacja ułatwia późniejszą weryfikację i walidację modeli przez inspekcję [5]. Notacja graficzna obejmuje wierzchołki operacyjne (prostokąty) oraz wierzchołki decyzyjne (romby). Wierzchołki są ze sobą połączone łukami skierowanymi. Wierzchołki operacyjne określają akcje wyjściowe systemu, zaś wierzchołki decyzyjne wprowadzają warunki zmiany stanu (wychodzą z nich dwa łuki – jeden określający logiczną wartość true, drugi – false). Ponadto możliwe jest określanie wyjść warunkowych typu Mealy'ego, przedstawianych jako owale. Wyjścia typu Moore'a przedstawiane są jako odpowiednie adnotacje wewnątrz prostokątów. Przykład algorytmicznej maszyny stanów ASM przedstawiono na rysunku 4.

3. Logika temporalna

Logika temporalna [7, 8, 9] została wprowadzona do dziedziny informatyki pod koniec lat 70-tych ubiegłego stulecia, kiedy to Amir Pnuelli zaproponował jej wykorzystanie w równoległych i reaktywnych systemach. Obecnie jest ona stosowana zarówno w specyfikacji programów, jak i ich późniejszej weryfikacji, syntezie, czy też przy programowaniu logicznym. Można także formalnie specyfikować funkcjonalność systemów osadzonych.

Klasyczną logiką temporalną jest liniowa logika temporalna LTL (ang. *Linear Temporal Logic*). Opisuje ona zależności w systemie przedstawiając sekwencję stanów. Pewna określona formuła może zmieniać swoją wartość przy zmianie stanów. Podstawowe operatory liniowej logiki temporalnej przedstawiono w tabeli 1.

Tab. 1. Podstawowe operatory logiki temporalnej
Tab. 1. Basic operators of temporal logic

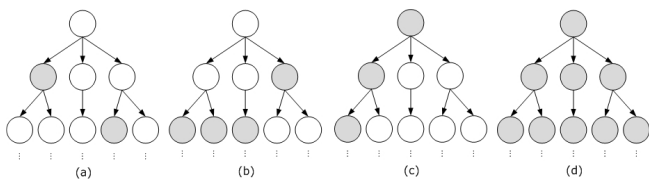
Znak	Znaczenie	Przykład
□	zawsze (always)	□ p formuła p prawdziwa we wszystkich stanach
◊	czasami (sometimes)	◊ p formuła p prawdziwa w pewnych stanach
o	następnie (next)	o p formuła p prawdziwa w następnym stanie

Logiką temporalną z podziałem czasu jest logika rozgałęziona CTL (ang. *Computation Tree Logic*). Czas przedstawiany jest tutaj jako drzewo rozszerzające się w przyszłość, gdzie głównym korzeniem jest aktualna chwila. Charakterystyczne dla logiki CTL są kwantyfikatory ścieżkowe (ang. *path quantifiers*) oraz kwantyfikatory punktowe (ang. *state quantifiers*). Kwantyfikatory ścieżkowe są dla ścieżek zaczynających się w danym stanie, a kwantyfikatory stanowe dla stanów w danej ścieżce. Podstawowe operatory temporalne przedstawiono w tabeli 2.

Tab. 2. Operatory temporalne w logice CTL
Tab. 2. Temporal operators in CTL logic

Kwantyfikator	Znaczenie
kwantyfikatory ścieżkowe	
E	dla pewnej ścieżki
A	dla każdej ścieżki
kwantyfikatory punktowe	
F	dla pewnego stanu
G	dla wszystkich stanów

Wykorzystując kwantyfikatory ścieżkowe i punktowe możliwe jest opisywanie złożonych zależności. Rysunek 1 prezentuje najczęściej używane konstrukcje:
(a) EFp – w pewnej ścieżce istnieje taki stan, w którym formuła p jest prawdziwa.
(b) AFp – w każdej ścieżce istnieje taki stan, w którym formuła p jest prawdziwa.
(c) EGp – w pewnej ścieżce formuła p jest prawdziwa w każdym stanie.
(d) AGp – w każdej ścieżce formuła p jest prawdziwa w każdym stanie.



Rys. 1. Łączenie kwantyfikatorów ścieżkowych i punktowych (formuła p prawdziwa w ciemnych kołach)
Fig. 1. Combining path and state quantifiers (formula p true in dark circles)

4. Weryfikacja modelu

Technika Model Checking [17, 19] jest jedną z metod formalnej weryfikacji specyfikacji (przegląd metod przedstawiony jest w pracy [2]). Weryfikacja przeprowadzana jest automatycznie przez narzędzia wnioskowania komputerowego (narzędzia typu model checker). Dane wejściowe to opis modelu (przedstawiony w języku opisu danego narzędzia) oraz lista wymagań stawianych projektowanemu systemowi (zawierająca zdefiniowane przy pomocy logiki temporalnej właściwości zapisane w charakterystycznym dla danego narzędzia języku specyfikacji). Należy mieć na uwadze fakt, że tylko wyspecyfikowane właściwości zostaną sprawdzone. Narzędzie model checker przeprowadza weryfikację systemu poprzez sprawdzenie czy dany model spełnia stawiane mu wymagania. Danymi wyjściowymi jest odpowiedź, czy model i jego specyfikacja są spójne, a w przypadku gdy tak nie jest – dodatkowo wygenerowany kontrprzykład, który pozwala na ręczne przeanalizowanie nieprawidłowej sytuacji. Formalna metoda weryfikacji Model Checking pozwala na zdiagnozowanie błędów w specyfikacji wymagań albo w opisie modelu.

Narzędzia model checker mogą być wykorzystywane do weryfikacji kompletnych systemów, bądź też tylko ich części. Jest to szczególnie istotne w przypadku złożonych systemów, gdzie proces ich projektowania trwa długo, zaś poszczególne moduły systemu można weryfikować krok po kroku uwzględniając za każdym razem pewien podzbiór wymagań.

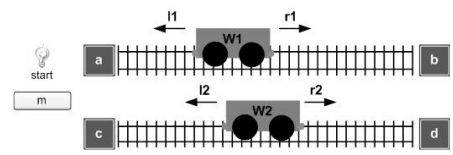
Jednym z narzędzi model checker pozwalających na automatyczną formalną weryfikację specyfikacji jest NuSMV [10, 11].

NuSMV jest reimplementacją i rozszerzeniem narzędzia SMV (Symbolic Model Verifier [1, 12, 13], narzędzie stworzone na Carnegie Mellon University umożliwiające weryfikację synchronicznych oraz asynchronicznych systemów o skończonej liczbie stanów). Rozprowadzany jest z licencją OpenSource, a jego rozwojem zainteresowane są zarówno instytuty naukowe, jak i firmy komercyjne. NuSMV pozwala na opisywanie skończonych maszyn stanów FSM, które zawierają zmienne i predykaty. Narzędzie umożliwia analizę specyfikacji wyrażonych przy pomocy logiki liniowej LTL oraz rozgałęzionej CTL wykorzystując oparte na binarnych diagramach decyzyjnych (ang. *Binary Decision Diagrams*) techniki Model Checking.

5. Formalna weryfikacja algorytmicznej maszyny stanów

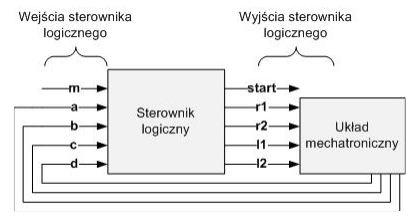
Przykład formalnej weryfikacji algorytmicznej maszyny stanów opisującej współbieżny proces sterowania zaczerpnięty jest z pracy [15]. Układ został zweryfikowany przy wykorzystaniu liniowej logiki temporalnej LTL i narzędzia NuSMV w aktualnej wersji 2.4.3

Układ odpowiedzialny jest za ruch dwóch wózków. W chwili początkowej wózki znajdują się odpowiednio w punkcie *a* i *c*. Po naciśnięciu przez użytkownika przycisku *m*, wózki rozpoczynają przejazd w prawą stronę. Wózki poruszają się tak długo, aż dojadą odpowiednio do punktu *b* i *d*. Następnie pierwszy wózek rozpoczyna ruch w lewą stronę. Gdy dojedzie do punktu startowego *a*, drugi wózek rozpoczyna przejazd do punktu *c*.



Rys. 2. Model rzeczywisty procesu sterowania ruchem wózków
Fig. 2. Real model of the controlling process of vehicle movement

Schemat sterownika logicznego dla przedstawionego procesu sterowania zamieszczony jest na rysunku 3.

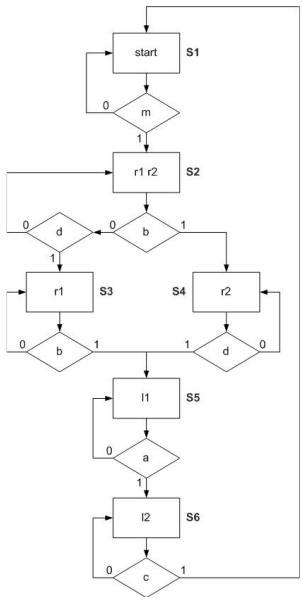


Rys. 3. Sterownik logiczny
Fig. 3. Logic controller

Nieformalny opis sterownika logicznego realizującego dane zadania może zostać przekształcony w diagram algorytmicznej maszyny stanów ASM (rysunek 4). Diagram ASM jest bardzo popularny wśród projektantów układów cyfrowych, zwłaszcza że jego tranformacja na języki opisu sprzętu (VHDL, Verilog) jest łatwa i wsparta licznymi publikacjami książkowymi.

Układ może się znajdować w jednym ze stanów S1 – S6. Do każdego stanu przypisane są sygnały wyjściowe, które są aktywne w czasie, gdy układ znajduje się w danym stanie. W chwili początkowej układ znajduje się w stanie S1 z aktywnym sygnałem wyjściowym *start*. W algorytmicznych maszynach stanów ko-

nieczne jest ciągłe sprawdzanie warunków, co tym samym wprowadza pętle – następny stan układu jest wtedy taki sam jak poprzedni. Ruch wózków w prawą stronę odbywa się współbieżnie, więc do punktu końcowego może dojechać jako pierwszy dowolny z wózków. Na diagramie ASM zaznaczone jest to przez kolejno sprawdzane warunki aktywnych sygnałów wejściowych *b* oraz *d* i przejście do odpowiedniego stanu (co wiąże się z odpowiednim ustawieniem sygnałów wyjściowych) w zależności od kolejności wystąpienia podanych sygnałów wejściowych. Ruch wózków w lewą stronę odbywa się sekwencyjnie.



Rys. 4. Algorytm procesu sterowania dla badanego przykładu przedstawiony za pomocą algorytmicznej maszyny stanów
Fig. 4. Algorithm of controlling process for the case study in form of an algorithmic state machine

Program sterownika przedstawiony diagramem algorytmicznej maszyny stanów stanowi podstawę do zdefiniowania opisu modelu w formacie charakterystycznym dla narzędzia NuSMV. Opis modelu może zostać sporządzony wprost jako sekwencja następujących po sobie stanów. Deklaracja zmiennych stanowych w opisie modelu (rysunek 5) zawiera stan wewnętrzny systemu, zmienne wyjściowe oraz zmienne wejściowe. Przyjęto, że w danej chwili na wejściu tylko jeden sygnał może być aktywny oraz że w danym stanie mogą wystąpić tylko oczekiwane sygnały wejściowe. Aktualny brak aktywnych sygnałów wejściowych określony jest jako wartość `none`. Przyjęte rozwiązanie bliskie jest automatycznej konwersji algorytmicznej maszyny stanów do formatu danego narzędzia model checker.

Opis modelu rozpoczyna się deklaracją zmiennych stanowych (rysunek 5). Stan systemu określony jest jako zestaw wartości każdej ze zmiennych. W omawianym przykładzie istnieje 216 możliwych kombinacji wartości zmiennych, z czego tylko 29 kombinacji jest osiągalnych.

```
VAR
   state  : {s1, s2, s3, s4, s5, s6};
   output : {start, r1, r2, l1, l2, r1r2};
   input  : {none, m, a, b, c, d};
```

Rys. 5. Sekcja VAR w opisie modelu NuSMV
Fig. 5. VAR section in NuSMV model description

W sekcji `ASSIGN` opisu modelu określone są przejścia między stanami. Określone są wartości początkowe zmiennych stanowych oraz ich następne możliwe wartości (wszystkie przypisania następnych wartości zmiennych odbywają się jednocześnie). W opisie modelu nie przewidziano sytuacji awaryjnych, np. możliwości ponownego uruchomienia wózków w przypadku ich wcześniejszego unieruchomienia pomiędzy punktami początkowymi a końcowymi. W takiej sytuacji konieczne jest ręczne dopchanie wózków do punktów startowych, co pozwoli na ponowne uruchomienie całego procesu. Przycisk *m* może zostać wciśnięty wyłącznie wtedy, gdy oba wózki znajdują się w punktach startowych.

Lista wymagań stawianych danemu modelowi systemu została sporządzona przy wykorzystaniu liniowej logiki temporalnej LTL (logika temporalna z podziałem czasu jest lepsza do weryfikacji niedeterministycznych programów [16]). Zawiera ona żądane właściwości systemu, które będą następnie weryfikowane. Pośród przedstawionych wymagań znajdują się zarówno wymagania dotyczące bezpieczeństwa (sytuacje, które nie mogą się zdarzyć), jak i wymagania dotyczące żywotności (sytuacje, które muszą się zdarzyć). Przykładowo dla zdefiniowanego opisu modelu spełnione są właściwości przedstawione na rysunku 6. Pierwsza właściwość podaje, że możliwe jest aby układ znajdował się w stanie zainicjowania (stan *s1*). Właściwość druga dotyczy żywotności i określa, że zawsze jeżeli sygnał wejściowy *m* będzie aktywny, to w końcu układ przejdzie do następnego stanu (lub aktywny będzie sygnał wyjściowy *r1r2* – właściwość trzecia). Właściwość czwarta dotyczy bezpieczeństwa i określa, że nigdy nie wystąpi sytuacja, w której aktywny będzie sygnał wyjściowy *l1* oraz jednocześnie *r1* lub *r1r2* – jednoczesne sygnały wymuszające ruch pierwszego (lub drugiego, właściwość piąta) wózka w lewą i w prawą stronę. Ostatnia właściwość definiuje, że nigdy nie będą aktywne jednocześnie sygnały wejściowe *b* lub *d* oraz sygnał *start* gotowości układu do rozpoczęcia procesu.

```
LTLSPEC -- nr 1
  F (state = s1);
LTLSPEC -- nr 2
  G (input = m -> F(state = s2));
LTLSPEC -- nr 3
  G (input = m -> F(output = r1r2));
LTLSPEC -- nr 4
  G !(output = l1 & (output = r1 | output = r1r2));
LTLSPEC -- nr 5
  G !(output = l2 & (output = r2 | output = r1r2));
LTLSPEC -- nr 6
  G !((input = b | input = d) & output = start);
```

Rys. 6. Lista spełnionych wymagań
Fig. 6. List of satisfied requirements

Narzędzie model checker porównuje opis modelu oraz listę wymagań i generuje odpowiedź na pytanie, czy pożądane właściwości są spełnione w zdefiniowanym modelu. Jeżeli którakolwiek z właściwości nie jest spełniona, generowany jest odpowiedni kontrprzykład, który zawiera ścieżkę pokazującą w jakich okolicznościach wymagana sytuacja nie może zostać spełniona.

Przykładowe wymaganie, które nie może być spełnione dla omawianego procesu sterowania, wraz z wygenerowanym kontrprzykładem przedstawione jest na rysunku 7.

```
-- specification  G (input = b ->  F output = l1)  is
false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  state = s1
  output = start
  input = none
-> Input: 1.2 <-
-> State: 1.2 <-
  input = m
-> Input: 1.3 <-
-> State: 1.3 <-
  state = s2
-> Input: 1.4 <-
-> State: 1.4 <-
  output = r1r2
  input = b
-> Input: 1.5 <-
-> State: 1.5 <-
  state = s4
  input = none
-> Input: 1.6 <-
-- Loop starts here
-> State: 1.6 <-
  output = r2
-> Input: 1.7 <-
-> State: 1.7 <-
```

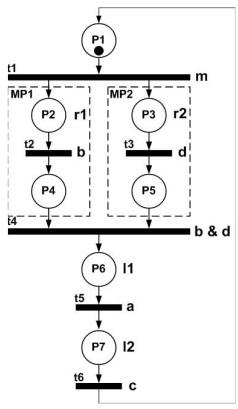
Rys. 7. Kontrprzykład
Fig. 7. Counterexample

Badana jest właściwość określająca, że zawsze jeżeli aktywny będzie sygnał wejściowy *b*, to w końcu aktywny będzie sygnał wyjściowy *l1*. Okazuje się, że ta właściwość nie może być zagwarantowana. Z analizy kontrprzykładu wynika, że możliwa jest sytuacja, w której pierwszy wózek dojedzie do punktu *b* (aktywny będzie sygnał wejściowy *b*), zaś drugi wózek nigdy nie dojedzie do punktu *d*, a więc powrót wózków nigdy się nie rozpocznie. Sytuacja może mieć miejsce w przypadku awarii drugiego wózka. Na podstawie wygenerowanego kontrprzykładu użytkownik musi podjąć decyzję, czy należy zmienić opis systemu, czy może listę właściwości.

6. Podsumowanie i wnioski

W artykule przedstawiono sposób formalnego opisu specyfikacji współbieżnego procesu sterowania w postaci diagramów algorytmicznych maszyn stanów ASM w języku aprobowanym przez profesjonalne narzędzie model checker NuSMV. Proponowane rozwiązanie polega na potraktowaniu diagramu ASM jako zwartej formy opisu systemu tranzycyjnego.

Formalna specyfikacja w postaci algorytmicznych maszyn stanów ASM posiada swoje zalety oraz wady. Niewątpliwą zaletą jest jej akceptacja przez projektantów układów cyfrowych ze względu na kompletny determinizm. Wszystkie możliwe sytuacje są dokładnie określone i zawsze wiadomo, który stan będzie następny. W porównaniu do formalnej specyfikacji w postaci sieci Petriego (rysunek 8), diagram ASM (rysunek 4) posiada dużo większą liczbę wierzchołków, przez co staje się mniej czytelny.



Rys. 8. Sieć Petriego dla omawianego przykładu
Fig. 8. Petri net for the discussed example

Sekwencyjne zapisanie przebiegu współbieżnego procesu sterowania nie jest zatem najlepszym rozwiązaniem w przypadku procesów silnie współbieżnych. Diagramy ASM nie wspierają ponadto modularności, co mogłoby zostać wykorzystane przy współbieżnych akcjach (rysunek 8, makromiejsca *MP1* i *MP2* określają ruch w prawą stronę odpowiednio pierwszego i drugiego wózka).

Przykład wykorzystania narzędzia NuSMV do weryfikacji specyfikacji sterownika logicznego w postaci sieci Petriego przedstawiony jest w [14].

Diagramy algorytmicznych maszyn stanów ASM mogą zostać przekonwertowane do języka aprobowanego przez narzędzie NuSMV. Opis modelu może zostać sporządzony wprost jako sekwencja następujących po sobie stanów (rozdział 5). Tworzenie opisu modelu obejmuje definicję zmiennych stanowych oraz przejść pomiędzy stanami. W sekcji `VAR` zadeklarowane są zmienne określające aktualny stan, sygnały wejściowe (określone na diagramie ASM wewnątrz wierzchołków decyzyjnych) oraz sygnały wyjściowe (określone na diagramie ASM wewnątrz wierzchołków operacyjnych) układu. W sekcji `ASSIGN` zdefiniowane są początkowe wartości zmiennych (`INIT`), które określają stan układu w momencie jego inicjalizacji. Następnie zdefiniowane są możliwe zmiany wartości zmiennych (`NEXT`). Lista wymagań tworzona jest w oparciu o zdefiniowane w opisie modelu zmienne i zawiera określone za pomocą logiki temporalnej właściwości, które system powinien spełniać. Innym sposobem jest sporządzenie opisu modelu bazując na zbiorach sygnałów wejściowych, wyjściowych oraz statusu i nieformalnej specyfikacji procesu sterowania [18].

Należy mieć na uwadze fakt, iż weryfikacja systemu nigdy nie potwierdzi, że system jest wolny od błędów. Możliwe jest jedynie sprawdzenie, czy system spełnia pewne zdefiniowane przez użytkownika właściwości. Po zakończeniu procesu weryfikacji konieczna jest interakcja użytkownika, który w przypadku wykrycia niespójności pomiędzy opisem modelu a listą wymagań będzie w stanie ręcznie przeanalizować wygenerowane kontrprzykłady.

7. Literatura

[1] C. Kern, M.R. Greenstreet: Formal Verification in Hardware Design: A Survey. ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol. 4, Issue 2, kwiecień 1999, str. 123 – 193.
[2] E.M. Clarke, J.M. Wing et al.: Formal methods: State of the Art and Future Directions. ACM Computing Surveys, Vol. 28, No. 4, gru-dzień 1996.
[3] G. Andrzejewski: Programowy model interpretowanej sieci Petriego dla potrzeb projektowania mikrosystemów cyfrowych. Oficyna Wydawnicza Uniwersytetu Zielonogórskiego 2003, str. 16 – 26.
[4] L. Gomes, J.P. Barros, A. Costa: Modeling Formalisms for Embedded System Design. Embedded Systems Handbook, Taylor & Francis Group, LLC, 2006.
[5] J.P. Davis, H.A. Wake, C. Leehaug, J. Branton, N. Namilae: Design versus Programming in Custom Computing Machine Applications: Experiences Using the Algorithmic State Machine Method. Proceedings Military Applications of Programmable Logic Devices – MAPLD-2004, Washington, DC, 8-10 września 2004.
[6] M.G. Arnold, J.R. Cowles, R.D. Joslin, J.J. Cupal: Simulation of cooperating algorithmic state machines using Verilog HDL. International Conference on Simulation and Hardware Description Languages (ICSHDL), 24-26 stycznia 1994, Tempe, Arizona, (edytorzy P.A. Widsey oraz D. Rhodes), The Society for Computer Simulation, str. 63 – 69.
[7] R. Klimek: Wprowadzenie do logiki temporalnej. AGH Uczelniane Wydawnictwa Naukowo-Dydaktyczne, Kraków 1999.
[8] M. Huth, M. Ryan: Logic in Computer Science. Modelling and Reasoning about Systems. Cambridge University Press 2004.
[9] M. Ben-Ari: Logika matematyczna w informatyce. Klasyka informatyki. Wydawnictwa Naukowo-Techniczne, Warszawa 2005.
[10] R. Eshuis: Symbolic Model Checking of UML Activity Diagrams. ACM Transactions on Software Engineering and Methodology, Vol. 15, No. 1, styczeń 2006, str. 1 – 38.
[11] R. Cavada et al.: NuSMV 2.4 User Manual.
[12] K.L. McMillan, The SMV system for SMV version 2.5.4.
[13] An introduction to model checking, By Girish Keshav Palshikar, Courtesy of Embedded Systems Programming, February 2004, www.embedded.com
[14] I. Grobelna: Formal verification of logic controller specification using NuSMV model checker. X Międzynarodowe Warsztaty Doktoranckie OWD'2008, Archiwum konferencji PTETIS, Vol. 25, październik 2008, str. 459 – 464.
[15] M. Adamski, M. Chodań: Modelowanie układów sterowania dyskretnego z wykorzystaniem sieci SFC. Wydawnictwo Politechniki Zielonogórskiej, Zielona Góra 2000.
[16] L. Lamport: "Sometime" is sometimes "not never", On the Temporal Logic of Programs, Proceedings of te Seventh ACM Symposium on Principles of Programming Languages, ACM SIGACT-SIGPLAN 1980, str. 174 – 185.
[17] E.M. Clarke, E.A. Emerson, A.P. Sistla: Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications, ACM Transactions on Programming Languages and Systems, Vol. 8, No. 2, April 1986, str. 244 – 263.
[18] I. Grobelna: Formalna analiza interpretowanych algorytmicznych maszyn stanów ASM z wykorzystaniem narzędzia model checker, Metody Informatyki Stosowanej. Przyjęte do publikacji.
[19] E.A. Emerson: 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.

Artykuł recenzowany