Agnieszka WĘGRZYN
UNIWERSYTET ZIELONOGÓRSKI, INSTYTUT INFORMATYKI I ELEKTRONIKI

Algorytm równoległy wyznaczania blokad i pułapek w sieciach Petriego

Dr inż. Agnieszka WĘGRZYN

Adiunkt w Instytucie Informatyki i Elektroniki Uniwersytetu Zielonogórskiego. Zainteresowania naukowe obejmują: modelowanie i weryfikację układów sterowania opisanych sieciami Petriego, formalne metody specyfikacji algorytmów sterowania, bazy danych oraz aplikacje internetowe.

e-mail: a.wegrzyn@iie.uz.zgora.pl

Streszczenie

W artykule została przedstawiona metoda równoległa wyznaczania blokad i pułapek w sieciach Petriego. Metoda ta bazuje na sekwencyjnym algorytmie obliczania zbiorów blokad i pułapek opisanym w [15]. Sekwencyjna metoda ma złożoność obliczeniową wykładniczą, dlatego istotna jest modyfikacja algorytmu w ten sposób, aby czas obliczania zdecydowanie zredukować. Rozpatrywana metoda wykorzystuje algorytm Thelena [13] – wyznaczania implikantów prostych na podstawie generowania i przeszukiwania drzewa.

Słowa kluczowe: sieci Petriego, analiza, blokady, pułapki, klaster.

Parallel algorithm for computation of deadlocks and traps in Petri nets

Abstract

In the paper the method of computation all deadlocks and traps in the Petri net is presented. This method is based on Thelen method [13] and it was proposed in [15]. Methods of calculation of all deadlocks and traps in Petri nets are very time consuming. Therefore it is very important to optimize of a computation. The parallel computation method for the time reduction is proposed. Experimental results of presented method are discussed, as well.

Keywords: Petri net, analysis, deadlocks, traps, cluster.

1. Wstęp

Proces projektowania układów cyfrowych można podzielić na kilka etapów – między innymi na modelowanie i weryfikację. W artykule rozpatrywane są układy sterowania binarnego. Do opisu takich układów można zastosować różne sposoby modelowania: języki opisu sprzętu (HDL), grafy stanów, sieci działań, diagramy przejść [5, 15], itp. Jednakże, ze względu na możliwość łatwej reprezentacji współbieżności oraz ze względu na dobrze zdefiniowane pojęcia, sieci Petriego najlepiej nadają się do modelowania układów sterowania dyskretnego. Ponadto, ze względu na szeroki aparat matematyczny, modele takie mogą być weryfikowane w sposób formalny. Specyfikacja funkcjonalna układu powstaje na podstawie analizy wymagań użytkownika. Zadaniem specyfikacji jest precyzyjne wyrażenie zewnętrznych skutków działania układów cyfrowych. Wykorzystując formalną specyfikację można już we wczesnym stadium projektowania wykryć i usunąć błędy.

Weryfikację układu cyfrowego opisanego siecią Petriego można sprowadzić do badania pewnych własności sieci Petriego. Do najważniejszych z nich należą żywotność i ograniczoność. Cechy te zapewniają odpowiednio, że w układzie nie dojdzie do zapętlenia procesów lub ich zatrzymania oraz, że dany proces wykonywany będzie skończoną liczbę razy. Wymienione własności można sprawdzać z wykorzystaniem różnych metod [5, 9, 14], jednakże znaczna większość z nich daje odpowiedź, czy dana sieć jest żywa i ograniczona, bez podania dokładnej informacji, w którym miejscu sieci wystąpił błąd, co w przypadku sieci opisujących układy sterowania jest bardzo istotne. Opisany w tym artykule algorytm wyznacza blokady i pułapki. Po sprawdzeniu zależności pomiędzy nimi można wskazać dokładne miejsce występowania defektu w sieci. Jednakże takie podejście jest czasochłonne, ponieważ liczba blokad i pułapek wzrasta wykładniczo wraz ze wzrostem liczby miejsc i tranzycji w sieci. Dlatego istotne jest prowadzenie badań nad przyspieszeniem obliczeń. W tym celu postanowiono wykonywać algorytm w sposób równoległy.

W następnym rozdziale zostaną opisane podstawy teoretyczne niezbędne do wyjaśnienia sposobu działania algorytmu wyznaczania blokad i pułapek. W rozdziale trzecim zostanie przedstawiona równoległa metoda weryfikacji sieci Petriego.

2. Podstawy teoretyczne

W rozdziale tym zostaną wymienione podstawowe definicje dotyczące sieci Petriego i ich własności, niezbędne do przedstawienia algorytmu wyznaczania blokad i pułapek.

Sieci Petriego są matematycznym obiektem, który istnieje niezależnie od fizycznej reprezentacji. Z wykorzystaniem sieci można wygodnie przedstawiać zjawiska zachodzące równolegle. Sieci Petriego są grafami dwudzielnymi o dwóch rodzajach wierzchołków zwanych miejscami i tranzycjami.

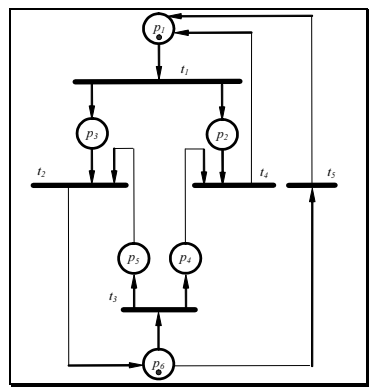
Sieć Petriego jest pojęciem abstrakcyjnym, formalnie zadanym w postaci uporządkowanej trójki $N = (P, T, F)$, dla której zachodzą następujące zależności (rys.1) [6, 9, 10]:

a) $P \cap T = \varnothing$
b) $P \cup T \neq \varnothing$
c) $F \subseteq (P \times T) \cup (T \times P)$
d) $\mathrm{dom}\,(F) \cup \mathrm{cod}\,(F) = P \cup T$

gdzie: P jest skończonym zbiorem wierzchołków - miejsc; T jest skończonym zbiorem wierzchołków - tranzycji; F jest relacją przepływu w sieci N, elementy zbioru F nazwane są łukami.

Relacja F miedzy miejscami i tranzycjami może być przedstawiana dwoma oddzielnymi relacjami kierunkowymi:

$$a \subseteq (P \times T)$$
$$b \subseteq (T \times P)$$



Rys. 1. Przykład sieci Petriego
Fig. 1. Example of Petri net

Zbiory tranzycji wejściowych i wyjściowych miejsca p definiowane są następująco:

$$\bullet p = \{t \in T : (t, p) \in F\}$$
$$p \bullet = \{t \in T : (p, t) \in F\}.$$

W podobny sposób określa się zbiory miejsc wejściowych i wyjściowych konkretnej tranzycji t:

$$\bullet t = \{p \in P : (p,t) \in F\}$$
$$t \bullet = \{p \in P : (t,p) \in F\}.$$

W wadliwie skonstruowanych sieciach może dojść do zakleszczeń i pułapek. Blokada (zakleszczenie) charakteryzuje się tym, że jeżeli nie ma znacznika przy pewnym znakowaniu, to nie będzie miała już znacznika w każdym następnym znakowaniu. Natomiast stan sieci, dla którego taka sytuacja występuje nazywany jest zastojem. W literaturze przedmiotu używana jest alternatywna nazwa dla blokady [11] - zatrzask [12].

Pułapka charakteryzuje się tym, że jeżeli jest oznakowana w pewnym znakowaniu (tzn. ma co najmniej jeden znacznik), to pozostaje oznakowana w każdym następnym znakowaniu, tzn. znacznik nigdy nie zostanie usunięty z pułapki.

Zbiór powstały w wyniku sumowania dwóch blokad (pułapek) jest również blokadą (odpowiednio pułapką). Blokada (pułapka) jest nazywana podstawową, jeżeli nie może zostać przedstawiona jako suma blokad (pułapek). Wszystkie blokady (pułapki) w sieci Petriego mogą zostać otrzymane jako sumy podstawowych blokad (pułapek).

Badając zależności pomiędzy blokadami i pułapkami można sprawdzić, czy sieć jest żywa i ograniczona [9] oraz dokonać dekompozycji sieci na składowe automatowe [5]. Więcej informacji na temat własności sieci można znaleźć w [1, 9, 10, 11, 12].

3. Równoległy algorytm wyznaczania blokad i pułapek

Algorytm wyznaczania zbiorów blokad i pułapek podzielony jest na kilka etapów. Pierwszym etapem jest wyznaczenie formuł Horna na podstawie struktury sieci Petriego opisujących blokady i pułapki. Formuła Horna jest koniunkcją podstawowych formuł Horna. Podstawową formułą Horna (klauzulą) nazywana jest dysjunkcja literałów z co najwyżej jednym pozytywnym literałem [3, 8]. Następnie na podstawie formuł generowane są drzewa Thelena, które są przeszukiwane algorytmem DFS. Drzewa wyznaczają wszystkie rozwiązania dla podanych formuł. W poszczególnym podrozdziałach zostaną opisane kolejne etapy algorytmu wraz z modyfikacją metody generowania drzewa.

3.1. Wyznaczanie formuł Horna

Każdą sieć Petriego można zapisać za pomocą formuł Horna. Rozwiązanie formuł Horna odpowiada zbiorom blokad/pułapek w sieciach. Topologiczna struktura sieci przedstawiona jest w postaci formuł Horna. W literaturze znane są również inne sposoby zapisu struktury sieci w postaci równań, jednakże większość z nich jest znacznie rozbudowana w stosunku do struktury sieci, np. w [4]. W celu otrzymania rozwiązań, należy równanie sprowadzić do postaci dysjunkcji. Rozwiązanie równania daje odpowiedź, czy sieć opisana danym równaniem spełnia określone własności, czyli w rozpatrywanym przypadku, czy zawiera blokady i pułapki. Rozwiązywanie formuł Horna polega na znalezieniu takich podstawień 0 lub 1 dla każdego literału z formuły, w taki sposób, że każda klauzula danej formuły będzie miała wartość 1, oraz formuła będzie miała wartość 1. Na podstawie wyrażeń przedstawionych w postaci koniunkcyjnej otrzymywane jest wyrażenie w postaci dysjunkcyjnej.

Postać wyrażenia (formuły Horna) dla danej sieci Petriego opisująca blokady:

$$\Pi t \in T \ \Pi pi \in t\bullet\ (yi + \Sigma pj \in \bullet t\ /yj) \qquad (1)$$

gdzie:

t, T - (odpowiednio) tranzycja, zbiór tranzycji;
$\bullet t, t\bullet$ - miejsca wejściowe i wyjściowe tranzycji t;
pi, pj - rozpatrywane miejsca $pi \in t\bullet$, $pj \in \bullet t$;
yi - zmienna reprezentująca miejsce pi, tzn. $pi \rightarrow yi=0$.

Twierdzenie *Blokady w sieci Petriego* [8]

Wszystkie wektory, będące rozwiązaniem równania (1), odpowiadają blokadom w sieci Petriego.

Postać wyrażenia dla danej sieci Petriego opisująca pułapki:

$$\Pi t \in T \ \Pi pi \in \bullet t\ (yi + \Sigma pj \in t\bullet\ /yj) \qquad (2)$$

gdzie:

t, T - (odpowiednio) tranzycja, zbiór tranzycji;
$\bullet t, t\bullet$ - miejsca wejściowe i wyjściowe tranzycji t;
pi, pj - kolejne miejsca;
yi - zmienna reprezentująca miejsce pi, tzn. $pi \rightarrow yi=0$.

Twierdzenie *Pułapki w sieci Petriego* [8]

Wszystkie wektory, będące rozwiązaniem równania (2), odpowiadają pułapkom w sieci Petriego.

Dla sieci z rys.1 wyznaczono dwie formuły Horna opisujące odpowiednio blokady *FH_ B* i pułapki *FH_P*:

FH_B = (/p1+p2)*(/p1+p3)*(/p3+/p5+p6)*(/p6+p4)*
(/p6+p5)*(/p2+/p4+p1)*(/p6+p1)
FH_P = (p1+/p2+/p3)*(p3+/p6)*(p5+/p6)*(p6+/p4+/p5)*
(p2+/p1)*(p4+/p1)*(p6+/p1)

3.2. Metoda równoległa generowania zbiorów blokad i pułapek

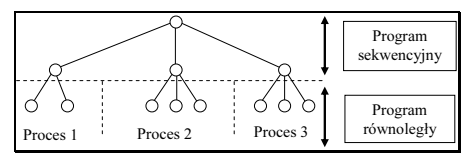
W celu wyznaczenia rozwiązań dla formuł Horna, wykorzystywany jest algorytm Thelena-Mathonego. Algorytm ten jest symboliczną metodą otrzymywania implikantów prostych. Nie są wykonywane tutaj algebraiczne mnożenia, a tworzone i przeszukiwane jest jedynie drzewo, zgodnie z dobrze znanym algorytmem przeszukiwania grafu wgłąb – DFS.

Konstrukcję drzewa, dla funkcji zadanej koniunkcją, można przedstawić następująco:

- każdy łuk reprezentuje literał;
- każdy wierzchołek reprezentuje iloczyn otrzymany przez wymnożenie literałów łuków tworzących ścieżkę od korzenia drzewa do danego wierzchołka;
- wierzchołki wiszące są implikantami prostymi lub ewentualnie implikantami pochłanianymi przez wygenerowane już implikanty proste.

Szersze opracowanie na temat sekwencyjnego algorytmu generowania drzewa można znaleźć w [7, 13, 15].

W celu zrównoleglenia obliczeń dokonano pionowego podziału drzewa (rys. 2) tak jak i w przypadku sekwencyjnie generowanego drzewa, algorytm przekształca funkcję z postaci koniunkcyjnej do postaci dysjunkcyjnej. Dane przesyłane są jedynie pomiędzy procesem głównym a procesami „dziećmi". Komunikacja pomiędzy „dziećmi" została wyeliminowania dzięki odpowiedniej strukturze funkcji optymalizującej. W funkcji skorzystano z dwóch metod komunikacji: plików i komunikatów. Pliki zostały wykorzystane do przekazywania danych wejściowych do procesów oraz wyników działania procesów. Przekazywanie listy dynamicznej przy użyciu plików było znacznie prostsze w implementacji niż przy komunikatach, oraz pliki dawały możliwość łatwego testowania aplikacji na etapie projektu. Komunikaty zostały jedynie zastosowane do przesyłania statusów, czyli informacji o stanie procesu „dziecka" do procesu głównego. Podczas podziału drzewa, procesy generowane są na podstawie częściowo wygenerowanego drzewa, to znaczy, że na początku algorytm wykonywany jest sekwencyjnie, tak długo, aż liczba węzłów w drzewie będzie równa, bądź większa od liczby zadeklarowanych procesów.

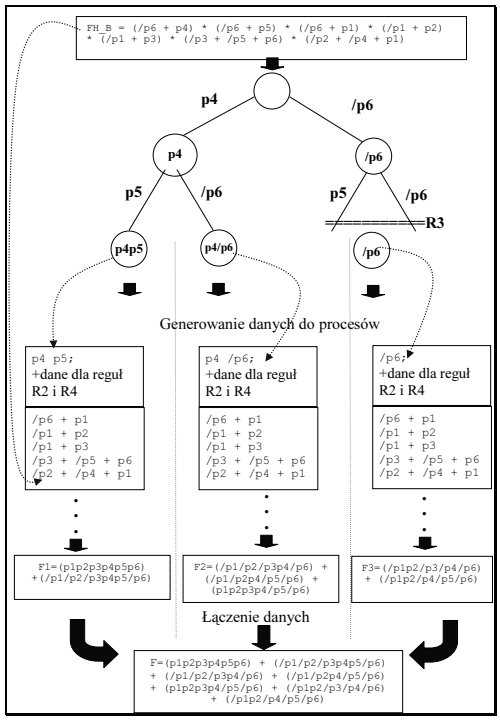


Rys. 2. Podział pionowy drzewa Helena
Fig. 2. Vertical decomposition of Petri net

W rozważanym przypadku drzewo zostało podzielone na trzy procesy (rys. 3). Drzewo generowano sekwencyjnie tak długo, aż wyznaczone zostały trzy węzły: *p4p5*, *p4*/*p6* oraz */p6*. Następnie każdy węzeł jest przetwarzany przez osobny proces. W każdym procesie przetwarzane są pozostałe klauzule wcześniej nieprzetworzone. Kolejnym krokiem jest połączenie uzyskanych wyników (*F1*, *F2* oraz *F3*). Następnie na podstawie otrzymanych wyników:

F = (p1p2p3p4p5p6)+(/p1/p2/p3p4p5/p6)+
(/p1/p2/p3p4/p6)+(/p1/p2p4/p5/p6)+
(p1p2p3p4/p5/p6)+(/p1p2/p3/p4/p6)+
(/p1p2/p4/p5/p6)

oraz zgodnie z dowodem zawartym w [8] wyznaczone zostały zbiory blokad (tab. 1). Podobnie postępuje się w przypadku generowania zbioru pułapek.



Rys. 3. Przykład podziału pionowego drzewa Thelena
Fig. 3. An example of vertical decomposition of Thelen tree

Badana sieć należy do klasy *FC* [9], jest spójna, bez tranzycji typu źródło (tzn. $t \in T$, takich że $\bullet t = 0$), bez tranzycji typu ujście (tzn. $t \in T$, takich że $t \bullet = 0$);czyli do sprawdzenia żywotności sieci należy rozpatrywać zawieranie się minimalnych pułapek oznakowanych w znakowaniu początkowym w minimalnych blokadach [9].

Tab. 1. Zbiór blokad
Tab. 1. A set of deadlocks

Blokada
{p5,p6}
{p1,p2,p3,p6}
{p1,p2,p5,p6}
{p1,p3,p4,p6}
{p1,p4,p5,p6}

Na podstawie tej własności można określić, że blokada {p5,p6} nie zawiera minimalnych pułapek, czyli sieć z rys. 1 nie jest żywa.

3.3. Wyniki eksperymentalne

Zmodyfikowany algorytm wyznaczania blokad i pułapek został przetestowany na komputerach równoległych, które zostały oparte na architekturze klastra MOSIX [2]. Klaster został zbudowanych na 4 komputerach klasy PC opartych na procesorach począwszy od Pentium II 400 MHz poprzez procesory AMD kończąc na Pentium IV 2,4 GHz.

Pomimo, iż różnice w szybkości pracy węzłów klastra są duże efektywność aplikacji pracującej na klastrze sięgała prawie 1500% (przyspieszenie 15-krotne). Podział danych na mniejsze porcje bardzo przyspiesza pracę całej aplikacji, gdyż poszczególne części drzewa są znacznie mniejsze i operacje na nim mogą się wykonywać dużo szybciej. Efektywność pracy aplikacji w systemie rozproszonym zwiększa się wraz ze wzrostem wielkości danych wejściowych. W przypadku małych rozmiarów danych wejściowych rozwiązanie sekwencyjne wykonuje się szybciej niż rozproszone, ponieważ podział danych dla kilku procesów zajmuje więcej czasu niż wykonanie całego algorytmu na jednym komputerze (tab. 2).

Tab. 2. Wyniki eksperymentalne
Tab. 2. Experimental results

Literały/ klauzule	Program sekw. [s]	Program równ. [s]	Literały/ klauzule	Program sekw. [s]	Program równ. [s]
20 / 20	0	1	30 / 30	47	11
20 / 30	1	1	30 / 40	166	17
20 / 40	2	1	35 / 20	57	13
25 / 10	1	2	35 / 30	155	17
25 / 20	5	4	35 / 40	795	47
25 / 30	21	6	40 / 20	360	45
25 / 40	46	10	40 / 30	611	79
30 / 20	11	8	40 / 40	1983	176

4. Podsumowanie

W artykule przedstawiono algorytm weryfikacji układów sterowania opisanych sieciami Petriego. Metoda bazuje na wyznaczaniu blokad i pułapek w sieciach na podstawie drzewa Thelena. Należy ona do metoda dokładnych, jednakże złożoność obliczeniowa jest wykładnicza. W związku z tym istotne jest prowadzenie prac nad przyspieszeniem obliczeń. Dlatego zdecydowano się na zrównoleglenie obliczeń. Przedstawione wyniki eksperymentalne udowodniły, że podejście takie jest bardzo efektywne.

Zaprezentowany algorytm można wykorzystać do badania żywotności i ograniczoności sieci Petriego. Aby określić, czy dana sieć jest żywa, należy sprawdzić zależności między blokadami a pułapkami. Metodą tą można sprawdzać sieci z klasy FC, EFC oraz AC i NSC z pewnymi ograniczeniami, które zostały opisane szerzej w [15].

Praca naukowa współfinansowana ze środków Komitetu Badań Naukowych w latach 2004-2006 jako projekt badawczy (grant nr 3 T11C 046 26).

5. Literatura

[1] E. R. Boer, T. Murata, Generating Basis Siphons and Traps of Petri Nets using the Sign Incidence Matrix, IEEE Transaction on Circuits and Systems, Fundamental Theory and Applications, Vol.41, 1994
[2] A. Barak, O. La'adan. The MOSIX Multicomputer Operating System for High Performance Cluster Computing. Journal of Future Generation Computer Systems, 13(4-5):361–372, March 1998.
[3] K.Barkaoui, M.Minoux, "A polynomial-time graph algorithm to decide liveness of some basic classes of bounded Petri nets", Lecture Notes in Computer Science, Springer Verlag, Berlin, Vol.616, 1992, pp.62 75
[4] C.Girault, R.Valk, Petri nets for systems engineering, Springer-Verlag Berlin Heilderberg, 2003
[5] A.Karatkevich, M.Adamski, M.Węgrzyn, "Rapid correctness analysis for sequential function chart", 45th International Scientific Coloquium, 04-06.10.2000, Ilmenau, ss.679-684, 2000
[6] Е.Котов, Сети Петри, М.: Наука, Главная редакция физико-математической литературы, 1984
[7] H.J.Mathony, Universal logic design algorithm and its application the synthesis of two-level switching circuits, IEE Proceedings, Vol.136, Pt.E, No.3, 1989
[8] M.Minoux, K.Barkaoui, Deadlocks and traps in Petri nets as a Hornsatisfability solutions and some related polynomially solvable problems Discrete Applied Mathematics, Elsevier Science Publishers (North Hol-land), Vol.29, 1990, pp.195-210
[9] T.Murata, Petri Nets: Properties, Analysis and Applications, Proceedings of the IEEE, Vol.77, No.4, ss.541-580, April 1989
[10] J.L.Peterson, Petri Net Theory and The Modeling of Systems, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1981
[11] W.Reisig, Sieci Petriego. Wprowadzenie, Wydawnictwa Naukowo-Techniczne, Warszawa, 1988
[12] P.H.Starke, Sieci Petri - podstawy, zastosowania, teoria, PWN, Warszawa, 1987
[13] B.Thelen, Investigations of algorithms for computer-aided logic design of digital circuits, PhD dissertation, ITIV, Univ. of Karlsruhe, 1981
[14] R.Valette, "Analysis of Petri nets by stepwise refinements", Journal of Computer and System Science, Vol.18, ss.35-36, 1979
[15] A. Wegrzyn, Symboliczna analiza układów sterowania z wykorzystaniem wybranych metod analizy sieci Petriego, Rozprawa doktorska, Politechnika Warszawska, 2003

Artykuł recenzowany