Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych

Agnieszka Węgrzyn

Symboliczna analiza układów sterowania binarnego z wykorzystaniem wybranych metod analizy sieci Petriego

Rozprawa doktorska

Promotor:
Prof. dr hab. inż. Marian Adamski

Warszawa, styczeń 2003

Streszczenie

W pracy przedstawiono nową metodę analizy sterowników cyfrowych modelowanych z wykorzystaniem hierarchicznej sieci Petriego. Stosując proponowaną metodę, za pośrednictwem badania strukturalnych własności sieci odzwierciedlonych w postaci formuły Horna, uzyskuje się informację o niezamierzonych defektach algorytmu sterowania procesem binarnym. Defekty przejawiające się w topologicznej strukturze sieci polegają na braku jej żywotności oraz wielokrotnym oznakowaniu tego samego miejsca (braku 1-ograniczoności, czyli bezpieczeństwa sieci).

Komputerowa realizacja opracowanych metod stanowi ważną część rozwijanego w Instytucie Informatyki i Elektroniki Uniwersytetu Zielonogórskiego akademickiego systemu CAD wspomagającego projektowanie współbieżnych sterowników cyfrowych, implementowanych w programowalnych strukturach logicznych FPGA i CPLD. W pracy zamieszczono przykład translacji sieci Petriego na równoważny model w języku opisu sprzętu Verilog-HDL. Wprowadzono nowy, użyteczny format PNSF3 regułowego opisu sterownika cyfrowego modelowanego hierarchiczną, interpretowaną, kolorowaną siecią Petriego. Obok szczegółowej weryfikacji modelu w dedykowanym środowisku wykorzystującym relacyjną bazę danych Oracle oraz technologię XML w połączeniu z grafiką SVG, proponuje się zastosowanie do tego celu również profesjonalnego pakietu Design/CPN. Końcowa symulacja sprawdzonego pod względem formalnym i poprawionego modelu może się również odbywać w nowoczesnym środowisku języków HDL, na przykład Cadence Verilog-XL lub Aldec Active-HDL.

Zasadnicza część metody analizy bazuje na algebrze Boole'a i symbolicznym przetwarzaniu danych. Ponieważ analizowana jest zredukowana sieć Petriego, abstrahowane zostają te podsieci, które nie mają wpływu na proces analizy lub nie są istotne na rozpatrywanym etapie projektu. Wartościowania spełniające złożoną formułę klauzulową Horna określają pułapki i blokady występujące w badanej sieci. W przypadku badania żywotności bada się zależności pomiędzy blokadami i pułapkami, co zgodnie z dotychczasowym stanem wiedzy zawęża klasę analizowanych sieci, do AC i NSC. Dodatkowo wyznaczone P-niezmienniki wykorzystywane są między innymi do dekompozycji sieci na składowe automatowe i kodowania miejsc sieci.

Proponowana metoda może również znaleźć zastosowanie w przemysłowych systemach bazujących na specyfikacji w języku graficznym SFC (*Sequential Function Chart*, norma IEC-1131-3).

Słowa kluczowe: układy cyfrowe, systemy CAD, hierarchiczne sieci Petriego, analiza współbieżnych algorytmów sterowania binarnego, żywotność, ograniczoność, analiza, modelowanie, dekompozycja, formuły Horna, XML, PNSF3, sieci SFC, języki opisu sprzętu (HDL), reprogramowalne struktury logiczne FPGA

Adres autora:
mgr inż. Agnieszka Węgrzyn
Uniwersytet Zielonogórska, Instytut Informatyki i Elektroniki
ul. Podgórna 50, 65-246 Zielona Góra
tel. (68) 3282484; fax (68) 3244733
e-mail: A.Wegrzyn@iie.uz.zgora.pl

Autorka pragnie złożyć podziękowania Panu Profesorowi Marianowi Adamskiemu za opiekę naukową nad prowadzonymi badaniami.

Ponadto autorka składa szczególne podziękowania Panu Doktorowi Andreiowi Karatkievichowi za dyskusje na temat teorii sieci Petriego.

Wyniki prezentowanych prac powstały częściowo w ramach realizacji grantów Komitetu Badań Naukowych - 8 T11C 018 18 oraz 7 T11C 009 20.

Spis treści

Spis rysunków

Spis tabel

1. Wstęp

Układy współbieżne stanowią ważną grupę układów cyfrowych. Znaczenie tego typu systemów wzrasta na skutek postępu technologicznego w dziedzinie elementów scalonych typu ASIC. Dużego znaczenia nabiera również konstruowanie równolegle działających części układu operacyjnego i związanego z nim sterownika współbieżnego. Do opisu takich układów można zastosować różne sposoby modelowania, np. języki opisu sprzętu (HDL), grafy stanów, sieci działań, diagramy przejść, itp. [ADAMSKI90], [BELHADIJ,ET.AL.93], [VENKATESH,ET.AL.94], [WOLAŃSKI98], [ŁABIAK01A], [ŁABIAK01B]. 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. Opisanie za ich pomocą działania układu sterującego sekwencją czynności, wykonywanych współbieżnie jest znacznie prostsze, niż opis funkcjonowania tego samego układu za pomocą innych metod. Ponadto, układy opisane sieciami Petriego, dzięki rozbudowanemu aparatowi matematycznemu i dużemu zestawowi metod analizy, mogą być weryfikowane w sposób formalny.

W 1962 roku Carl Adam Petri swoją rozprawą doktorską zapoczątkował rozwój nowej teorii umożliwiającej badanie zjawisk współbieżnych [PETRI62]. Nosząca imię swojego twórcy teoria sieci Petriego stanowi obecnie obszerną i szybko rozwijającą się dziedzinę nauki [MURATA89]. Modele sieci Petriego odgrywają ważną rolę w projektowaniu i eksploatacji systemów informatycznych oraz w planowaniu i sterowaniu przepływem produkcji [ADAMSKI90], [KALINOWSKI84], [KOZŁOWSKI95], [MISIUREWICZ78]. Badania prowadzone w dziedzinie projektowania układów cyfrowych potwierdziły przydatność sieci Petriego do analizy, weryfikacji i syntezy tego typu układów [DAVID,ALLA92], [JENSEN92], [JENSEN94], [JENSEN97], [REISIG88], [STARKE87].

Teoria sieci obejmuje bogaty zestaw algorytmów badających wybrane ich właściwości takie, jak: żywotność i ograniczoność, które znalazły zastosowania w różnych dziedzinach techniki [BARKAOUI,MINOUX92], [BILIŃSKI96], [COMMONER72], [DATTA,GHOSH86], [ESPERZA,SILVA91A], [JENSEN97], [KARATKEVICH,ET.AL.00], [KOTOW84], [WAGNER,RAKOWSKI81], [WĘGRZYN,WĘGRZYN99], [WĘGRZYN,WĘGRZYN00], [WĘGRZYN01]. W odniesieniu do projektowania systemów cyfrowych, analiza formalna sieci Petriego pozwala wykryć szereg niezamierzonych defektów realizowanego algorytmu

sterowania. Przykładowo, żywotność sieci zapewnia, że w układzie zamodelowanym siecią, nie występują zagnieżdżenia i zapętlenia. Jeżeli sieć, opisująca układ sterowania, nie jest żywa, to może nastąpić częściowe lub całkowite unieruchomienie modelowanego systemu. Natomiast ograniczoność sieci określa, że każda operacja układu zakończy się przed jej powtórną inicjalizacją.

Istnieje wiele algorytmów badania własności sieci Petriego, jednakże większość z nich daje odpowiedź, czy zamodelowana sieć spełnia, czy też nie, pewne własności. W przypadku analizy modeli opisujących sterowniki logiczne, ważnym elementem jest podanie, w którym miejscu sieci wystąpił defekt, aby w łatwy sposób go zlokalizować i usunąć.

1.1. Motywacja

Pomysł wykorzystania sieci Petriego w celu modelowania układów cyfrowych, a zwłaszcza kontrolerów logicznych, pojawił się pod koniec lat 70-tych. Aparat formalny sieci Petriego w połączeniu z metodami logiki formalnej umożliwia weryfikację i syntezę reprogramowanego sterownika logicznego metodą systematyczną. Układy współbieżne w intuicyjny sposób można zapisać w postaci interpretowanej sieci Petriego.

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. Specyfikacja formalna jednoznacznie określa postulowane działania układu cyfrowego. Formalna specyfikacja sterownika po przekształceniu jest odzwierciedlona w sposób bezpośredni w instrukcjach języków HDL (np. VHDL) i implementowana z wykorzystaniem programowalnych struktur CPLD lub FPGA [ADAMSKI98].

Stosowanie sieci Petriego jako pośredniej formy specyfikacji pomiędzy opisem w języku naturalnym i specyfikacją logiczną ma wiele zalet. Do najważniejszych z nich należy możliwość weryfikacji opisanego algorytmu z wykorzystaniem bogatego aparatu matematycznego teorii sieci Petriego [ADAMSKI90].

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.

W literaturze opisano wiele algorytmów badania wyżej wymienionych własności ([MURATA89], [BARKAOUI,MINOUX92], [KARATKEVICH,ET.AL.00], [VALETTE79], [ZAKREVSKIJ99]), 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 istotnym elementem. Dlatego celowym było podjęcie się opracowania nowej metody wyznaczania wszystkich blokad i pułapek, oraz na ich podstawie stwierdzenia, czy sieć spełnia badane własności. Metoda będzie bazowała na symbolicznym przetwarzaniu danych. W literaturze można znaleźć metody wyznaczania wszystkich blokad i pułapek, jednakże metody te bazują na algebrze liniowej (złożonym rachunku macierzowym) [ZAKREVSKIJ85], [EZPELETA,ET.AL.93].

W związku z opracowaniem formalnego odwzorowania sieci Petriego w symboliczną postać regułową, zdecydowano, że opracowywana metoda powinna również operować na symbolach, gdyż niecelowym byłoby ich odwzorowywanie, np. w algebrze liniowej.

Ponadto, nowe poglądy pojawiające się w literaturze dotyczącej weryfikacji układów cyfrowych sugerują, że metody symboliczne, w tym metody SAT mogą być bardzo obiecujące [BIERE,ET.AL.99], [NOVIKOV,GOLDBERG01], [GOLDBERG,NOVIKOV02]. Powstające, w dużych zespołach międzynarodowych, uniwersalne profesjonalne oprogramowanie może stanowić alternatywę dla oprogramowania dedykowanego, zrealizowanego w Uniwersytecie Zielonogórskim.

Na podstawie przeglądu literatury, wyciągnięto wniosek, że warto rozwinąć dotychczasowe badania dotyczące symbolicznych metod analizy sieci Petriego, wprowadzając szereg przydatnych uzupełnień o charakterze teoretycznym, w konsekwencji których metody symboliczne przydatne będą w praktyce. Przesłanki zostały wymienione w rozdziale 7.

Oprócz elementu analizy sieci, ważnym zagadnieniem poruszanym w pracy jest dekompozycja sieci na podsieci sekwencyjne, które reprezentują automaty cyfrowe. Na podstawie opracowanej metody analizy, równocześnie wyznaczane są P-niezmienniki sieci określające podsieci o charakterze sekwencyjnym. P-niezmienniki stosowane są między innymi do P-kolorowania sieci [WĘGRZYN98], co następnie pozwala na dekompozycję sieci oraz odpowiednie kodowanie stanów lokalnych oraz otwiera możliwość jej dalszej analizy (np. powtarzalności).

W trakcie badań nad możliwościami modelowania i analizy układów współbieżnych z wykorzystaniem sieci Petriego powstało wiele tekstowych formatów opisu sieci, na przykład Petri Net Specification Format (PNSF) [KOZŁOWSKI,ET.AL.95].

Podobną postać regułową wykorzystano w języku CONPAR [FERNANDES,ET.AL.97]. Oba formaty posłużyły następnie do opracowania formatu rozszerzonego o możliwość opisu również kolorowanych sieci Petriego (PNSF2) [WĘGRZYN98]. Powstanie nowej technologii zapisu dokumentów (XML) dało możliwość automatycznej transformacji pomiędzy różnymi formami opisu układów cyfrowych. W ramach pracy został opracowany, łącząc zalety wcześniejszych, nowy format zapisu sieci Petriego z wykorzystaniem języka XML. Ponadto opracowano metodę automatycznej transformacji sieci SFC na sieci Petriego opartą na metodach opisanych w [ADAMSKI,CHODAŃ00].

1.2. Teza i cele pracy

Analizując aktualny stan badań postawiono następującą tezę pracy:

> ***Analiza algorytmów sterowania binarnego opisanych w sposób regułowy i modelowanych sieciami Petriego może być wystarczająco efektywnie przeprowadzona metodami symbolicznymi powiązanymi z redukcją sieci.***

W celu udowodnienia powyższej tezy, można sformułować siedem głównych celów pracy:

1. Opracowanie oryginalnej metody analizy sieci Petriego bazującej na symbolicznym przetwarzaniu danych.
2. Opracowanie nowej metody wyznaczania P-niezmienników, w celu dekompozycji i kodowania miejsc sieci Petriego.
3. Opracowanie formatu XML opisu interpretowanej hierarchicznej i kolorowanej sieci Petriego.
4. Opracowanie modelu relacyjnego bazy danych, reprezentującego sieci Petriego, w celu przechowywania informacji o strukturze sieci oraz jej efektywnej konwersji do formatu XML.
5. Opracowanie metody symulacji interpretowanej, hierarchicznej i kolorowanej sieci Petriego z wykorzystaniem formatu XML.
6. Zademonstrowanie przydatności opracowanej metody do analizy sieci SFC.
7. Opracowanie metody transformacji sieci SFC na sieci Petriego z wykorzystaniem formatu XML.

Praca ma charakter teoretyczny, praktyczny i eksperymentalny.

1.3. Struktura pracy

Praca została podzielona na dziesięć części. Rozdział pierwszy wprowadza do tematyki poruszanej w pracy. W rozdziale tym przedstawiona została teza i cele pracy oraz motywacja podjęcia się badań.

Rozdział drugi zawiera wstęp teoretyczny, przydatny dla lepszego zrozumienia pracy. Zamieszczono w nim najważniejsze twierdzenia i definicje dotyczące teorii sieci Petriego oraz przegląd ważniejszych własności sieci.

Rozdział trzeci przedstawia zagadnienia dotyczące spełnialności formuł logicznych, ze szczególnym uwzględnieniem formuł Horna. Zawarto w nim również odpowiednie definicje oraz wybrane algorytmy sprawdzania spełnialności.

W rozdziale czwartym zaprezentowano dwie metody modelowania sterowników logicznych: interpretowane sieci Petriego i sieci SFC oraz sposoby transformacji sieci SFC do sieci Petriego.

Kolejny rozdział, piąty, stanowi przegląd wybranych metod analizy sieci Petriego. Każda metoda została zilustrowana przykładem. Rozdział kończy się podsumowaniem, w którym porównano opisane metody, w postaci tabelarycznego zestawienia.

W rozdziale szóstym przedstawiono nowy sposób opisu sieci Petriego w języku XML nazwanego formatem PNSF3. Zaprezentowano również zalety takiego formatu wykorzystywanego do automatycznej symulacji sieci.

Rozdział siódmy prezentuje nową symboliczną metodę analizy hierarchicznej sieci Petriego z wykorzystaniem formuł Horna i drzewa Thelena.

Rozdział ósmy podaje sposób weryfikacji, symbolicznej metody analizy sieci, na podstawie programu Design/CPN. Zademonstrowano również możliwości modelowania sterowników logicznych, kolorowanymi sieciami Petriego.

Rozdział dziewiąty omawia sposób wykorzystania systemu PN-XML w modelowaniu i analizie sterowników logicznych. Przedstawione w nim zostały przykłady analizy sterowników pod kątem autorskiej syntezy w ramach systemu PeNCAD opracowywanego na Uniwersytecie Zielonogórskim.

Rozdział dziesiąty stanowi podsumowanie pracy oraz wskazuje kierunki dalszych badań.

2. Wybrane zagadnienia dotyczące sieci Petriego

W rozdziale tym zostaną przedstawione podstawowe pojęcia dotyczące teorii sieci Petriego. Zwrócona zostanie uwaga na te elementy teorii, które związane są z analizą sieci Petriego, pod kątem badania zastojów.

2.1. Sieci Petriego typu P/T

Poniżej zostaną przedstawione podstawowe definicje dotyczące wybranych zagadnień sieci Petriego [BERNADINELLO,CINDIO92], [BEST,FERNANDEZ86], [BANASZAK,ET.AL.93], [DAVID,ALLA92], [GIRAULT,VALK03], [PETERSON81], [REISIG88], [STARKE87], [BARKAOUI95], [ŻURAWSKI,ZHOU94].

Definicja 1. Sieć Petriego

Sieć Petriego jest pojęciem abstrakcyjnym, formalnie zadanym w postaci uporządkowanej trójki $N = (P, T, F)$, dla której zachodzą następujące zależności:

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: F jest relacja przepływu w sieci N. Elementy F noszą nazwę *łuków*.

Elementy należące do zbioru P nazywane są *P-elementami (miejscami)*.

Elementy ze zbioru T nazywane są *T-elementami (tranzycjami)*.

Dziedzina relacji F, $\mathrm{dom}(F)$, określona jest jako $\mathrm{dom}(F) = \{x \mid \exists y: (x,y) \in F\}$, a przeciwdziedzina jako $\mathrm{cod}(F) = \{x \mid \exists y: (y,x) \in F\}$.

Zbiór $X = \{P \cup T\}$ jest zbiorem *wierzchołków* sieci.

Relacja F między *miejscami* i *tranzycjami* niekiedy przedstawiana jest dwoma oddzielnymi relacjami kierunkowymi:

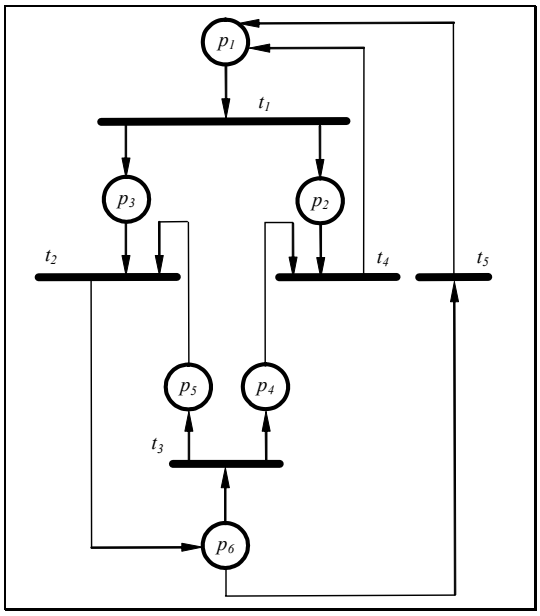
$a \subseteq (P \times T)$

$b \subseteq (T \times P)$

Rys. 1 przedstawia sieć Petriego [KOTOW84], w której:

- do zbioru P należą miejsca *(p1, p2, p3, p4, p5, p6)*;

- do zbioru *T* należą tranzycje *(t1, t2, t3, t4, t5)*;
- do zbioru *X* należą wierzchołki sieci *(p1, p2, p3, p4, p5, p6, t1, t2, t3, t4, t5)*.



Rys. 1. Przykładowa sieć Petriego

Definicja 2. Zbiory wierzchołków wejściowych i wyjściowych miejsca lub tranzycji

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*:

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

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

Zamiennie mogą być stosowane następujące oznaczenia:

$\bullet p = \mathrm{pre}(p)$

$p\bullet = \mathrm{post}(p)$

$\bullet t = \mathrm{pre}(t)$

$t\bullet = \mathrm{post}(t)$

Powyższe definicje łatwo jest rozszerzyć dla podzbiorów miejsc lub tranzycji:

$X_1 \subseteq X \quad \Rightarrow \quad \bullet X_1 = \cup_{x \in X_1} \bullet x \quad$ oraz $\quad X_1\bullet = \cup_{x \in X_1} x\bullet$

Dla sieci Petriego z Rys. 1:

- zbiorem tranzycji wejściowych miejsca *p1* jest $\bullet p1=(t4, t5)$;
- zbiorem tranzycji wyjściowych miejsca *p1* jest $p1\bullet=(t1)$;
- zbiorem miejsc wejściowych tranzycji *t1* jest $\bullet t1=(p1)$;

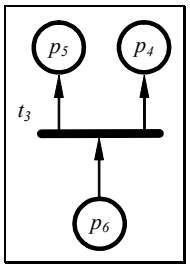
- zbiorem miejsc wyjściowych tranzycji *t1* jest *t1•*=(*p2, p3*).

Definicja 3. Podsieć

Niech $N_1 = (P_1, T_1, F_1)$ i $N_2 = (P_2, T_2, F_2)$ są parą sieci Petriego. Sieć N_1 jest *podsiecią sieci* N_2 wtedy i tylko wtedy, gdy:

$$P_1 \subseteq P_2, T_1 \subseteq T_2 \quad \text{oraz} \quad F_1 = F_2 \cap ((P_1 \times T_1) \cup (T_1 \times P_1)).$$

Podsiecią N_1 (Rys. 2) sieci z Rys. 1 jest sieć składająca się z miejsc (*p4, p5, p6*) oraz z tranzycji (*t3*) wraz z łukami łączącymi te wierzchołki.



Rys. 2. Przykładowa podsieć

Definicja 4. Sieć prosta

Niech $N = (P, T, F)$ jest siecią Petriego.

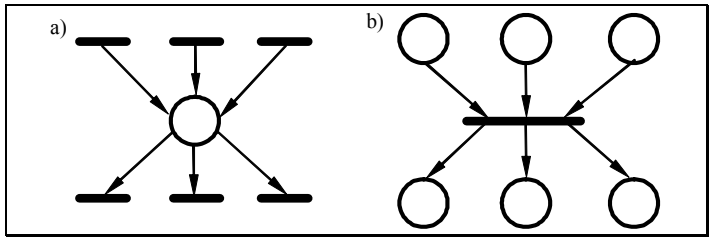
Sieć jest *P-prostą*, wtedy i tylko wtedy, gdy:

$$\forall\, p_1,p_2 \in P\colon\ (\bullet p_1 = \bullet p_2 \ \wedge\ p_1\bullet = p_2\bullet) \ \Rightarrow\ p_1 = p_2.$$

Sieć jest *T-prostą*, wtedy i tylko wtedy, gdy:

$$\forall\, t_1,t_2 \in T\colon\ (\bullet t_1 = \bullet t_2 \ \wedge\ t_1\bullet = t_2\bullet) \ \Rightarrow\ t_1 = t_2.$$

Sieć jest nazywana *prostą*, wtedy i tylko wtedy, gdy jest równocześnie P-prostą i T-prostą. Rys. 3a prezentuje P-prostą sieć Petriego, natomiast Rys. 3b T-prostą sieć Petriego.



Rys. 3. Przykładowa a) P-prosta sieć b) T-prosta sieć

Definicja 5. P-sieć

Sieć $N = (P, T, F)$ jest nazywana *P-siecią* wtedy i tylko wtedy, gdy:

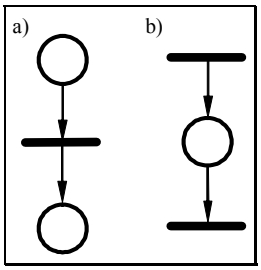
$$\forall t \in T\colon\ |\bullet t| \le 1 \quad \text{i} \quad |t\bullet| \le 1.$$

Definicja 6. T-sieć

Sieć $N = (P, T, F)$ jest nazywana *T-siecią* wtedy i tylko wtedy, gdy:

$$\forall p \in P\colon\ |\bullet p| \le 1 \quad \text{i} \quad |p\bullet| \le 1.$$

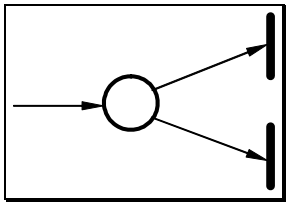
Rys. 4 a) i b) przedstawia odpowiednio przykładową P-sieć i T-sieć.



Rys. 4. Przykładowa a) P-sieć, b) T-sieć

Definicja 7. Sieć swobodnego (wolnego) wyboru (FC)

Sieć $N = (P, T, F)$ jest nazywana *siecią swobodnego (wolnego) wyboru (FC)*, wtedy i tylko wtedy, gdy: $\forall p \in P$: $|p\bullet| > 1 \Rightarrow \bullet(p\bullet) = \{p\}$.

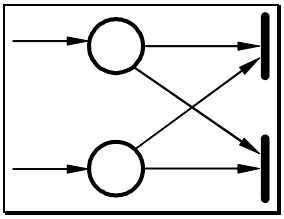


Rys. 5. Fragment sieci swobodnego wyboru (FC)

UWAGA: Sieć Petriego jest siecią swobodnego wyboru wtedy i tylko wtedy, gdy wszystkie tranzycje należące do konfliktu mają tylko jedno miejsce wejściowe.

Definicja 8. Rozszerzona sieć swobodnego wyboru (EFC)

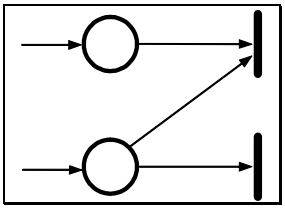
Sieć $N = (P, T, F)$ jest nazywana *rozszerzoną siecią swobodnego wyboru (EFC)*, wtedy i tylko wtedy, gdy: $\forall p_1,p_2 \in P$: $p_1\bullet \cap p_2\bullet \neq \varnothing \Rightarrow p_1\bullet = p_2\bullet$



Rys. 6. Fragment rozszerzonej sieci swobodnego wyboru (EFC)

Definicja 9. Sieć asymetrycznego wyboru (AC)

Sieć $PN = (P, T; F)$ jest nazywana *siecią asymetrycznego wyboru (AC)* wtedy i tylko wtedy, gdy: $\forall p_1,p_2 \in P$: $p_1\bullet \cap p_2\bullet \neq \varnothing \Rightarrow p_1\bullet \subseteq p_2\bullet \vee p_2\bullet \subseteq p_1\bullet$



Rys. 7. Fragment sieci asymetrycznego wyboru (AC)

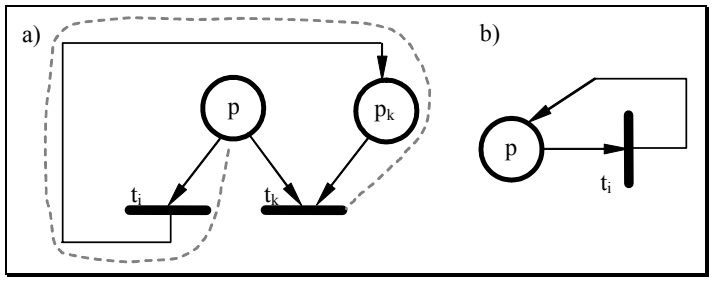
Definicja 10. Nie samo-kontrolująca sieć

Sieć $N = (P, T, F)$ jest nazywana nie samo-kontrolującą siecią *(ang. non-self controlling nets, NSC)*, wtedy i tylko wtedy, gdy:

$\forall p \in P$ takiego, że $p\bullet \geq 2$ nie istnieje $t_i \in p\bullet$, dla których spełnione są następujące dwa warunki:

1. istnieje elementarna ścieżka w postaci $(p,t_i,...,t_k)$, gdzie $t_k \in p\bullet$ oraz $t_k \neq t_i$;
2. istnieje cykl zawierający p i t_i

Elementarna ścieżka charakteryzuje się tym, że wszystkie węzły w takiej ścieżce muszą być różne. Na Rys. 8 przedstawione zostały dwie konstrukcje, które nie są dozwolone w sieciach Petriego z klasy NSC, rysunek a) ilustruje warunek 1 z definicji (przerywana linia pokazuje elementarną ścieżkę), natomiast rysunek 2) warunek 2.



Rys. 8. Niedozwolone konstrukcje w sieciach z klasy NSC

Definicja 11. α-sieć [ZAKREVSKIJ87]

Sieć $N = (P, T, F)$ jest nazywana *α-siecią* jeżeli sieć Petriego należy do klasy *EFC* i znakowanie początkowe zawiera tylko jeden znacznik.

Definicja 12. Znakowana sieć Petriego (P/T sieć)

Znakowana sieć Petriego (*P/T system* lub *P/T sieć*) reprezentowana jest uporządkowaną szóstką:

$PN = (P, T, F, K, W, M_0)$

w której:

a) $(P, T; F)$ jest siecią Petriego z miejscami P i tranzycjami T,

b) K: $P \rightarrow \aleph^+ \cup \{\infty\}$ jest *funkcją pojemności miejsc*,

c) W: $F \rightarrow \aleph^+$ jest *funkcją wagi łuków*,

d) M_0: $P \rightarrow \aleph$ jest *funkcją znakowania początkowego* spełniającą warunek $M_0(p) \leq K(p)$ dla każdego $p \in P$

UWAGA: Najczęściej wykorzystuje się podklasę PT sieci, dla której zachodzi $K(p)=\infty$ dla każdego $p \in P$ oraz $W(f)=1$ dla wszystkich f w F. Nazywane one są *zwykłymi sieciami Petriego.*

W graficznej reprezentacji P/T systemu łuki $f \in F$ są etykietowane przez $W(f)$ tylko wtedy, kiedy $W(f) > 1$. Jeżeli pojemność miejsc jest skończona, liczbę $K(p)$ podaje się obok miejsca. Z tego względu należy wyraźnie odróżnić oznaczenie miejsca (często w postaci numeru porządkowego) od pojemności miejsca. W przypadku $K(p) = \infty$ oznaczenie pojemności miejsca można pominąć. Znakowanie początkowe reprezentowane jest znacznikami w każdym kółku, przedstawiającym miejsce znakowane początkowo.

Definicja 13. Znakowanie, przygotowanie, realizacja tranzycji

Niech $PN = (P, T, F, K, W, M_0)$ jest znakowaną siecią Petriego.

a) Funkcja $M: P \rightarrow \aleph$ jest nazywana *znakowaniem sieci PN* wtedy i tylko wtedy, gdy:

$M(p) \leq K(p)$ dla wszystkich $p \in P$.

a) Tranzycja $t \in T$ jest *przygotowana do realizacji* (*dozwolona*) w oznakowaniu M wtedy i tylko wtedy, gdy

$\forall p \in P$: $W(p,t) \leq M(p) \leq K(p) - W(t,p)$.

b) Jeżeli $t \in T$ jest tranzycją, która jest dozwolona w oznakowaniu M to t może *zostać zrealizowana*.

Nowe znakowanie określone jest wyrażeniem

$M'(p) = M(p) - W(p,t) + W(t,p)$ dla wszystkich $p \in P$.

Wystąpienie tranzycji t zmienia znakowanie M na znakowanie M', co oznacza się przez:

$M[t\rangle M'$ lub $M \rightarrow^t M'$.

UWAGA: Zamiast tradycyjnie stosowanego pojęcia *zapalenie (odpalenie) tranzycji* używa się pojęcia *realizacja* (*wystąpienie*) *tranzycji*.

Definicja 14. Osiągalność znakowania M' ze znakowania M

Znakowanie M' jest *osiągalne* ze znakowania M jeżeli istnieje ciąg znakowań M_0, M_1, M_2,…, M' i słowo $t = t_1\ t_2\ \ldots\ t_m$ w alfabecie T takie, że

$M[t_1\rangle M_1[t_2\rangle \ldots [t_m\rangle M'$.

Definicja 15. Współbieżne wzbudzenie tranzycji, konflikt

Niech $PN = (P, T, F, K, W, M_0)$ jest znakowana siecią Petriego, $M \in [M_0\rangle$ i $t_1, t_2 \in T$. Wówczas:

a) t_1 i t_2 są *współbieżnie dozwolone* w znakowaniu M, wtedy i tylko wtedy, gdy:

$\forall p \in P$: $W(p,t_1) + W(p,t_2) \leq M(p) \leq K(p) - W(t_1,p) - W(t_2,p)$

b) t_1 i t_2 są w *konflikcie* w znakowaniu M wtedy i tylko wtedy, gdy $t_1 \neq t_2$ oraz M zezwala zarówno na realizację t_1 jak i t_2 oraz jeśli $M\ [t_1\rangle\ M_1$ to M_1 nie zezwala na realizację tranzycji t_2 albo jeśli $M\ [t_2\rangle\ M_2$ to M_2 nie zezwala na realizację tranzycji t_1.

Definicja 16. Graf znakowań osiągalnych

Graf znakowań osiągalnych skończonej sieci $PN = (P, T; F, K, W, M_0)$ jest grafem, którego wierzchołki są etykietowane funkcjami z $P \rightarrow \aleph \cup \{\omega\}$ (zasady dotyczące ω podane są poniżej) i którego łuki etykietowane są elementami z T. *Graf znakowań osiągalnych* konstruowany jest w następujący sposób:

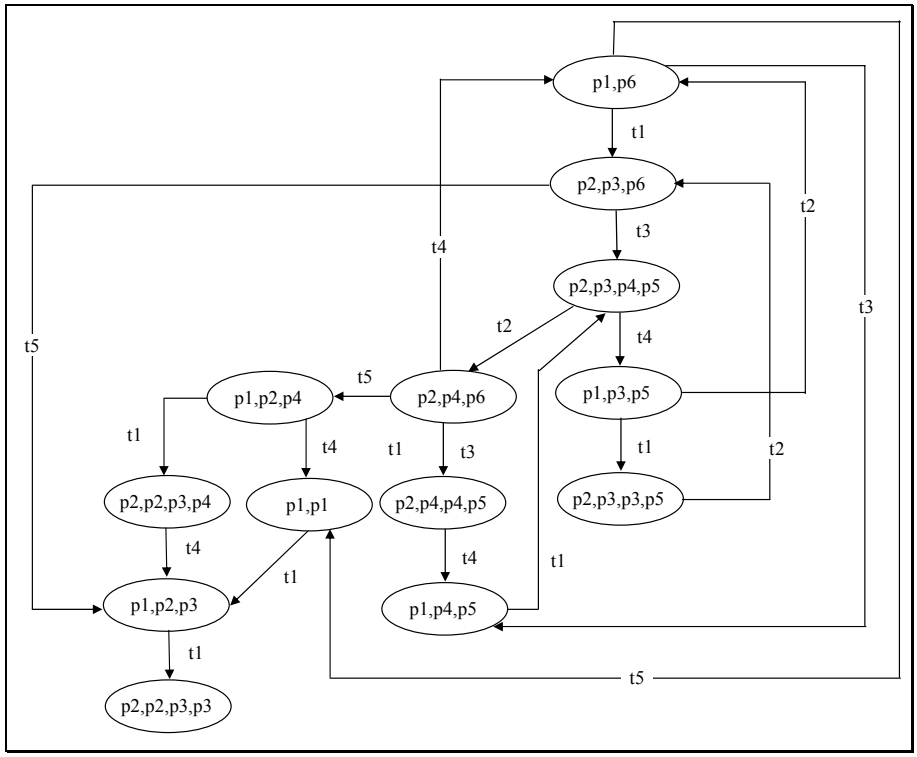
1. Dwa wierzchołki drzewa znakowań osiągalnych są równoważnymi, wtedy i tylko wtedy, gdy mają tą samą etykietę,
2. Wierzchołki grafu znakowań są klasami równoważności drzewa znakowań. Łuk z etykietą t prowadzi z wierzchołka grafu znakowań Y do wierzchołka grafu Z wtedy i tylko wtedy, gdy $\exists\ y \in Y$ i $\exists\ z \in Z$, takie, że w drzewie znakowań występuje łuk z y do z.

Zasady dotyczące stosowania symbolu ω są następujące:

$n < \omega$ dla wszystkich $n \in \aleph$,

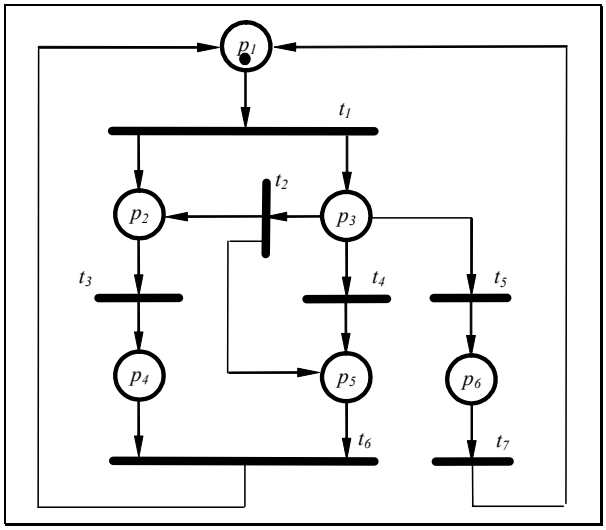
$n + \omega = \omega + \omega = \omega + \mathrm{n} = \omega - n = \omega$ dla wszystkich $n \in \aleph$.

Rys. 9 przedstawia graf znakowań dla sieci 1-ograniczonej (bezpiecznej) z Rys. 1.

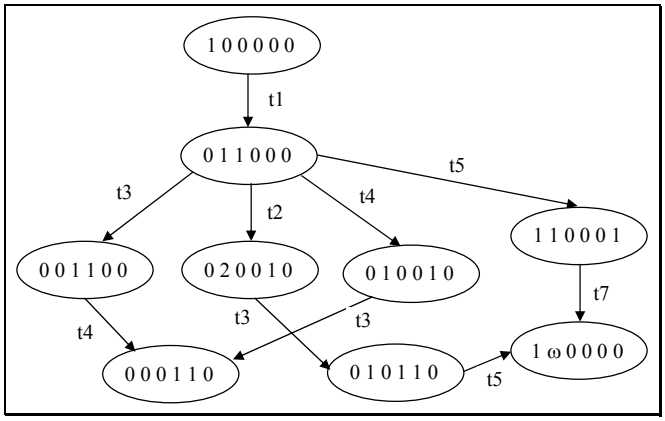


Rys. 9. Graf znakowań dla sieci z Rys. 1

W celu dokładniejszego zilustrowania definicji 15 przedstawiono nieograniczoną sieć Petriego (Rys. 10) oraz jej graf znakowań (Rys. 11).



Rys. 10. Nieograniczona sieć Petriego



Rys. 11. Graf znakowań dla nieograniczonej sieci Petriego

Definicja 17. Macierz incydencji

Niech $PN = (P, T, F, K, W, M_0)$ jest siecią Petriego o zbiorze miejsc $P = \{p_1, p_2, \ldots, p_n\}$ i zbiorze tranzycji $T = \{t_1, t_2, \ldots, t_m\}$. Macierz $C = \|c_{ij}\|$ $(1 \le i \le n,\ 1 \le j \le m)$ jest *macierzą incydencji* sieci PN, gdy $c_{ij} = W(t_j, p_i) - W(p_i, t_j)$.

UWAGA: Oprócz macierzy incydencji sieci w podanej wyżej postaci stosuje się dwie oddzielne macierze: *macierz incydencji wejściowej A* i *macierz incydencji wyjściowej B* o elementach odpowiednio różnych a i b:

$\|a_{ij}\|$: $a_{ij} = a\ (p_i, t_j)$

$\|b_{ij}\|$: $b_{ij} = b\ (t_i, p_j)$.

Rys. 12 przedstawia macierz incydencji C dla sieci z Rys. 1, natomiast Rys. 13 pokazuje odpowiednio, macierz incydencji wejściowej A i macierz incydencji wyjściowej B.

$$C = \begin{array}{c} \\ t1 \\ t2 \\ t3 \\ t4 \\ t5 \end{array} \begin{array}{c} \begin{array}{cccccc} p1 & p2 & p3 & p4 & p5 & p6 \end{array} \\ \begin{bmatrix} 1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 & -1 & 1 \\ -1 & 1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Rys. 12. Macierz incydencji dla sieci z Rys. 1

$$A = \begin{array}{c} \\ t1 \\ t2 \\ t3 \\ t4 \\ t5 \end{array} \begin{array}{c} \begin{array}{cccccc} p1 & p2 & p3 & p4 & p5 & p6 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \qquad B = \begin{array}{c} \\ t1 \\ t2 \\ t3 \\ t4 \\ t5 \end{array} \begin{array}{c} \begin{array}{cccccc} p1 & p2 & p3 & p4 & p5 & p6 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

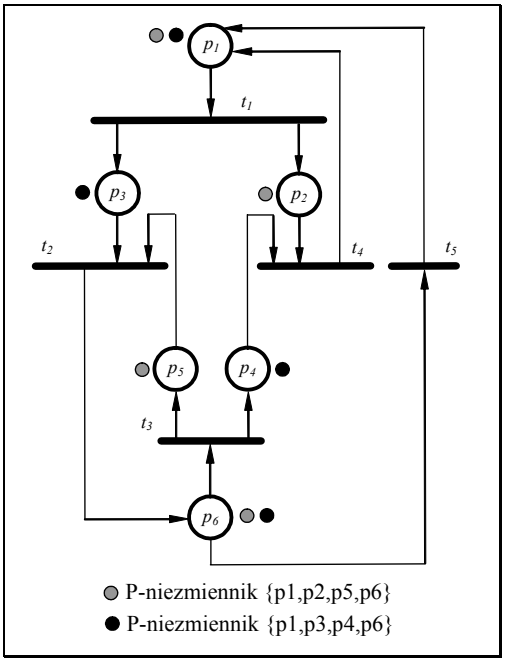
Rys. 13. Macierze incydencji A - wejściowej i B - wyjściowej

Definicja 18. P-niezmienniki

Niech N będzie P/T siecią. Wektor miejsc I: $P_N \rightarrow Z$ jest *P-niezmiennikiem* wtedy i tylko wtedy gdy C' * I = 0, gdzie C' jest macierzą transponowaną macierzy incydencji C.

P-niezmiennikami dla sieci z Rys. 1 są dwa zbiory miejsc {p1,p2,p5,p6} oraz {p1,p3,p4,p6}, które odpowiadają wektorom [1,0,1,1,0,1] oraz [1,1,0,0,1,1].

Rys. 14 przedstawia pokrycie P-niezmiennikami przykładowej sieci Petriego.



Rys. 14. P-niezmienniki sieci Petriego

Definicja 19. T-niezmienniki

Niech N będzie P/T siecią. Wektor I: $T_N \rightarrow Z$ jest *T-niezmiennikiem* wtedy i tylko wtedy gdy C * I = 0.

T-niezmiennikiem dla sieci z Rys. 1 jest zbiór tranzycji {t1,t2,t3,t4}, któremu odpowiada wektor [1,1,1,1,0].

Definicja 20. Pokrycie sieci P-niezmiennikami

P/T sieć N jest siecią *pokrytą P-niezmiennikami* wtedy i tylko wtedy, gdy dla każdego miejsca $p \in P_N$ istnieje dodatni P-niezmiennik I sieci N, przy czym I(p)>0.

Definicja 21. Sieć ograniczona

P/T sieć N jest siecią *ograniczoną* wtedy i tylko wtedy, gdy M_N jest skończone oraz istnieje takie $n \in N$, że $M(p) \leq n$ dla wszystkich $M \in [M_N>$ i dla wszystkich $p \in P_N$.

Twierdzenie 1. Ograniczoność sieci Petriego [REISIG88]

Niech N będzie P/T-siecią i niech M_N będzie skończone. Jeżeli sieć N jest pokryta P-niezmiennikami, to jest ona *ograniczona*.

Definicja 22. Pokrycie sieci T-niezmiennikami

P/T sieć N nazywamy siecią pokrytą T-niezmiennikami wtedy i tylko wtedy, gdy dla każdej tranzycji $t \in T_N$ istnieje dodatni T-niezmiennik I sieci *N*, przy czym I(t)>0.

Twierdzenie 2. Żywotność i ograniczoność sieci Petriego [REISIG88]

Jeżeli sieć Petriego jest żywa i ograniczona to jest ona pokryta T-niezmiennikami.

UWAGA: Cechą określaną przez P-niezmienniki jest stała liczba znaczników w danym podzbiorze miejsc sieci.

P-niezmiennikiem dla danej sieci jest każdy wektor I, którego współrzędne niezerowe wyznaczają te miejsca, w których łączna liczba znaczników nie zmienia się, niezależnie od tego, które tranzycje zostały zrealizowane.

Cechą określaną przez T-niezmienniki jest liczba realizacji tranzycji w celu otrzymania zadanego znakowania początkowego.

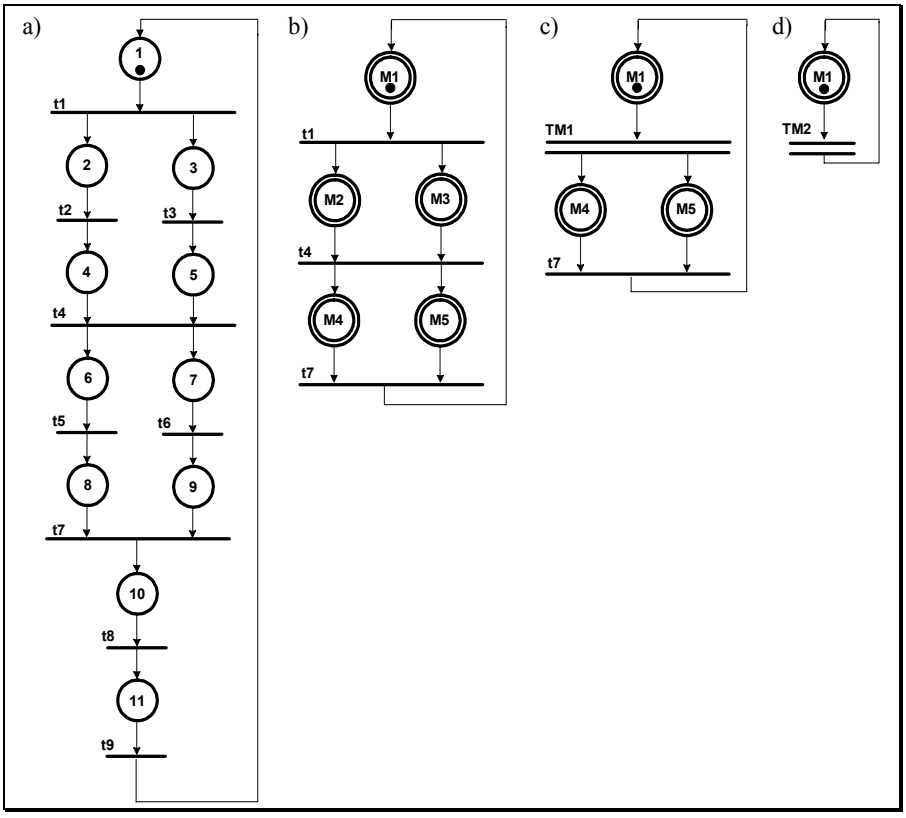
2.2. Hierarchiczne sieci Petriego typu P/T

Konstruowanie sieci hierarchicznej można realizować dwoma sposobami. Pierwszy z nich - *od szczegółu do ogółu* (ang. *bottom-up*), polega na wyodrębnieniu w modelowanym systemie małych podsystemów i zbudowaniu dla nich sieci. Następnie, sieci te są składane w coraz większe części, aż do otrzymania całej sieci będącej modelem

dla zadanego systemu. Drugi sposób - *od ogółu do szczegółu* (ang. *top-down*), polega na zbudowaniu sieci dla całego systemu, bez szczegółowego analizowania jego fragmentów. Następnie, w podobny sposób traktuje się poszczególne fragmenty, itd. Postępowanie takie nosi nazwę *modelowania strukturalnego (hierarchicznego)*. Sieć zbudowana z wykorzystaniem jednej z tych metod nazywa się ***siecią hierarchiczną***, a elementy sieci o złożonej strukturze są nazywane *elementami sieci hierarchicznej*. Szersze opracowanie dotyczące omawianego zagadnienia zawarte jest między innymi w pracach [ADAMSKI,WĘGRZYN94], [BUCHHOLZ94], [DATTA,GHOSH84], [DATTA,GHOSH86], [FEHLING94], [JENSEN,ROZENBERG91], [SURAJ,KOMAREK94], [WĘGRZYN,ADAMSKI99], [LEE,FAVREL85].

Wybór sposobu modelowania zależy głównie od rodzaju systemu przeznaczonego do zamodelowania. Jeżeli jest on zupełnie nieznany projektantowi systemu, to efektywniejszy jest sposób drugi. Natomiast, gdy w modelowaniu systemu można wykorzystać dotychczasowe doświadczenie, to wygodniejsza staje się metoda pierwsza.

Rys. 15a przedstawia płaską sieć Petriego zaczerpniętą z [ADAMSKI,CHODAŃ00]. Na podstawie tego przykładu, zostało zaprezentowane konstruowanie hierarchicznej sieci Petriego, sposobem od szczegółu do ogółu (Rys. 15b, Rys. 15c, Rys. 15d). Elementy oznaczone symbolami *M1*, *M2*, *M3*, *M4*, *M5* reprezentują fragmenty sieci zwane makromiejscami. Natomiast elementy *TM1* i *TM2* odpowiadają makrotranzycjom.



Rys. 15. Kolejne poziomy hierarchicznej sieci Petriego

2.3. Wybrane własności sieci Petriego

Analiza sieci Petriego może zostać sprowadzona do badania pewnych własności sieci na przykład żywotności [MURATA89]. Żywotność sieci jest ściśle związana z występowaniem blokad w sieci. Własność tą można określić jako możliwość utrzymania żywego znakowania sieci, czyli dla każdej tranzycji można z każdego znakowania osiągalnego dojść do takiego znakowania, w którym ta tranzycja będzie mogła zostać zrealizowana. Żywotność sieci można analizować pod kątem występowania blokad i pułapek oraz badania zależności między nimi (własności Commoner'a i Hack'a) [COMMONER72].

Żywotności sieci Petriego można podzielić na trzy poziomy:

1. żywotność strukturalna;
2. żywotność dynamiczna;
3. żywotność sieci interpretowanej.

Na poziomie pierwszym analizowana jest sieć bez interpretacji i znakowania. Metody analizy takiej sieci są znacznie szybsze od metod analizy sieci ze znakowaniem. Jednakże w niektórych przypadkach defekt układu zamodelowanego siecią zależy od tego, który stan jest stanem początkowym, czyli od znakowania początkowego. W przypadku stwierdzenia, że układ opisany siecią został zamodelowany poprawnie, nie można zapewnić, że po dodaniu znakowania układ zamodelowany siecią również będzie działać poprawnie.

Na drugim poziomie weryfikowana jest sieć ze znakowaniem. Metody analizy takiej sieci dają odpowiedź na pytanie, czy przy zadanym znakowaniu sieć jest poprawnie skonstruowana. Złożoność obliczeniowa tego typu metod jest większa, niż metod badania żywotności strukturalnej sieci. Za pomocą tych metod można analizować również sieci z łukami zabraniającymi i zezwalającymi, które można traktować jako dodatkowe warunki przejść dla tranzycji.

Na trzecim poziomie analizowana jest sieć interpretowana, jednakże metody te są bardzo skomplikowane, a złożoność obliczeniowa może być bardzo duża.

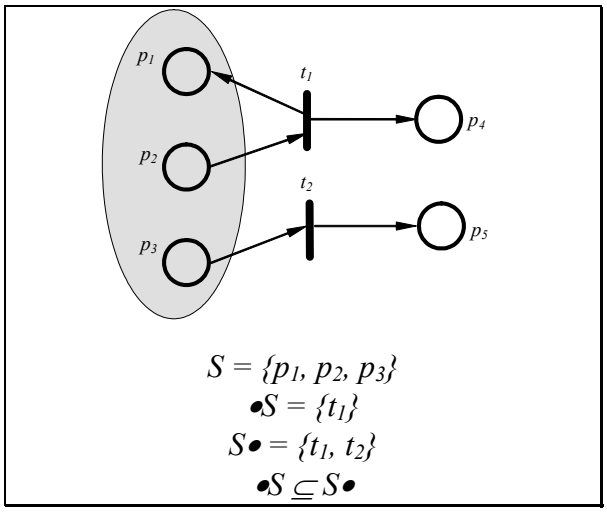
W celu wyjaśnienia zagadnień związanych z wybranymi własnościami sieci Petriego zostaną przedstawione dodatkowe definicje i twierdzenia [MURATA89], [BEST,FERNANDEZ86], [BEST87], [GIRAULT,VALK03].

Definicja 23. Blokada

Niech $N = (P, T; F)$ jest siecią Petriego oraz $S \subseteq P$. Podzbiór miejsc S jest nazywany *blokadą* wtedy i tylko wtedy, gdy $\bullet S \subseteq S\bullet$.

Ilustrację blokady można przedstawić na przykładzie fragmentu sieci składającej się z pięciu miejsc (Rys. 16). Każda tranzycja mająca wyjściowe miejsca w podzbiorze *S* ma również miejsca wejściowe w tym samym podzbiorze *S*. Liczba znaczników w blokadzie *S* pozostaje taka sama podczas realizacji tranzycji t_1, ale zmniejsza się podczas realizacji tranzycji t_2. Blokada 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 [REISIG88] - zatrzask [STARKE87], [SURAJ, SZPYRKA99].



Rys. 16. Przykład blokady

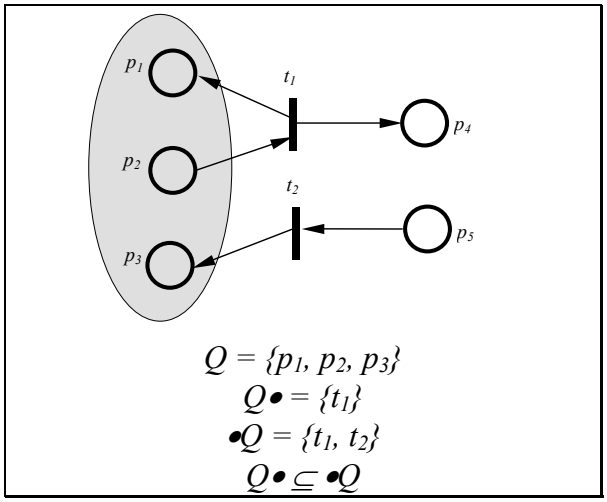
Definicja 24. Pułapka

Niech $N = (P, T; F)$ jest siecią Petriego oraz $Q \subseteq P$. Podzbiór Q jest nazywany *pułapką* wtedy i tylko wtedy, gdy $Q\bullet \subseteq \bullet Q$.

Podobnie jak w przypadku blokady, pułapka zostanie wyjaśniona na przykładzie (Rys. 17). W rozpatrywanej pułapce Q liczba znaczników pozostaje bez zmian przy realizacji tranzycji t_1, natomiast zwiększa się przy realizacji tranzycji t_2.

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).



Rys. 17. Przykład pułapki

Definicja 25. Blokada (pułapka) podstawowa

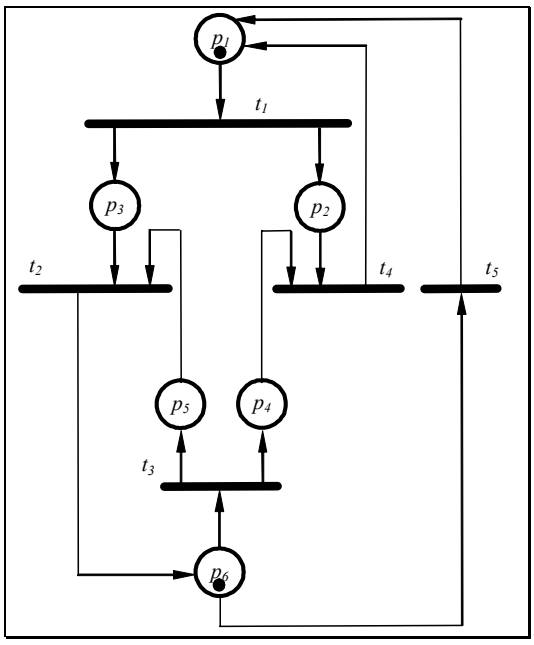
Blokada (pułapka) jest nazywana *podstawową*, jeżeli nie może zostać przedstawiona jako suma blokad (pułapek).

UWAGA: Wszystkie blokady (pułapki) w sieci Petriego mogą zostać otrzymane jako sumy podstawowych blokad (pułapek).

Definicja 26. Blokada minimalna

Blokada jest nazywana *minimalną*, jeżeli nie zawiera żadnych innych blokad.

UWAGA: Podstawowa blokada (pułapka) nie może być połączeniem dwóch blokad (pułapek). Minimalne blokady (pułapki) są podstawowymi, ale nie wszystkie podstawowe są minimalnymi.



Rys. 18. Sieć Petriego z blokadami i pułapkami

Przykład sieci zawierającej blokady i pułapki zamieszczono na rysunku (Rys. 18) [KOTOW84].

Sieć zawiera dziesięć blokad i dziesięć pułapek. Tabela 1 zawiera wszystkie zbiory blokad i pułapek, oraz blokady i pułapki minimalne.

Tabela 1. Zbiór blokad i pułapek dla sieci z Rys. 18

Blokada	*Blokada minimalna*	*Pułapka*	*Pułapka minimalna*
{p5,p6}	tak	{p1,p2}	tak
{p4,p5,p6}	nie	{p1,p2,p5,p6}	nie
{p1,p2,p3,p6}	tak	{p1,p2,p4}	nie
{p1,p2,p5,p6}	nie	{p1,p2,p4,p6}	nie
{p1,p2,p3,p5,p6}	nie	{p1,p2,p3,p5,p6}	nie
{p1,p3,p4,p6}	tak	{p1,p2,p4,p5,p6}	nie
{p1,p2,p3,p4,p6}	nie	{p1,p3,p4,p6}	tak
{p1,p4,p5,p6}	nie	{p1,p2,p3,p4,p6}	nie
{p1,p2,p4,p5,p6}	nie	{p1,p3,p4,p5,p6}	nie
{p1,p3,p4,p5,p6}	nie	{p1,p3,p5,p6}	tak

Definicja 27. Ograniczoność sieci

Dana jest sieci Petriego $PN = (P, T; F, K, W, M_0)$:

a) Miejsce $p \in P$ jest *n-ograniczone* ($n \in \aleph$) wtedy i tylko wtedy, gdy:
$\forall M \in [M_0\rangle: M(p) \leq n$

b) Sieć PN jest *n-ograniczona* ($n \in \aleph$) wtedy i tylko wtedy, gdy $\forall\ p \in P$: p jest n-ograniczone.

c) Sieć PN jest *ograniczona* wtedy i tylko wtedy, gdy $\exists\ n \in \aleph$: PN jest n-ograniczona.

d) Sieć PN nazywamy *bezpieczną*, gdy jest 1-ograniczoną.

e) Sieć PN nazywamy *strukturalnie ograniczoną* wtedy i tylko wtedy, gdy sieć PN jest ograniczona dla dowolnego znakowania początkowego.

Definicja 28. Żywotność sieci

a) Tranzycja $t \in T$ w $PN = (P, T; F, K, W, M_0)$, jest *żywą* wtedy i tylko wtedy, gdy:
$\forall M \in [M_0\rangle\ \exists\ M' \in [M\rangle$: t jest dozwolone w M'.

b) Sieć PN jest *żywa* wtedy i tylko wtedy, gdy $\forall t \in T$: t jest żywa.

c) Sieć PN jest *strukturalnie żywa* wtedy i tylko wtedy, gdy $\exists\ M$: sieć jest żywa w tym znakowaniu.

d) Sieć PN jest *wolna od zastoju* wtedy i tylko wtedy, gdy $\forall M \in [M_0\rangle\ \exists\ t \in T$: t jest dozwolone w M.

e) Znakowanie $M \in [M_0\rangle$ jest *znakowaniem reprodukowalnym* wtedy i tylko wtedy, gdy:
$\exists\ M' \in [M\rangle$: $M' \neq M\ \wedge\ M \in [M'\rangle$.

f) Znakowanie $M' \in [M_0\rangle$ jest *znakowaniem domowym* wtedy i tylko wtedy, gdy: $\forall M \in [M_0\rangle: M' \in [M\rangle$.

g) Tranzycja $t \in T$ w $PN = (P, T; F, K, W, M_0)$ jest *martwą* wtedy i tylko wtedy, gdy: $\neg\exists\, M \in [M_0\rangle$: t jest dozwolone w M.

h) $M \in [M_0\rangle$ jest *znakowaniem martwym* $\Leftrightarrow \neg\exists\, t \in T \;\; \exists\, M' \in [M_0\rangle: [M\, t\rangle\, M'$.

Definicja 29. Sieć strukturalnie ograniczona

Sieć Petriego N jest strukturalnie ograniczona wtedy i tylko wtedy, gdy dla każdego znakowania początkowego M_0, sieć $<N, M_0>$ jest ograniczona.

Definicja 30. Sieć strukturalnie żywa

Sieć Petriego N jest strukturalnie żywa wtedy i tylko wtedy, gdy istnieje takie znakowanie początkowe M_0, dla którego sieć $<N, M_0>$ jest żywa.

Własność Commoner'a [COMMONER72]

Oznakowana sieć Petriego PN ma własność Commoner'a wtedy i tylko wtedy, gdy każda blokada w sieci PN zawiera przynajmniej jedną pułapkę z co najmniej jednym miejscem oznakowanym w znakowaniu początkowym M_0.

Własność Commoner'a jest warunkiem koniecznym i wystarczającym do sprawdzenia żywotności sieci EFC.

UWAGA: Strukturalna żywotność jest warunkiem żywotności sieci Petriego.

Twierdzenie 3. Sieć swobodnego wyboru żywa i ograniczona [MURATA89]

Niech sieć Petriego PN będzie siecią swobodnego wyboru. Istnieje takie znakowanie M_0, dla którego sieć jest żywa i ograniczona, wtedy i tylko wtedy, gdy sieć jest strukturalnie żywa i strukturalnie ograniczona.

Twierdzenie 4. Strukturalnie żywa i strukturalnie bezpieczna sieć swobodnego wyboru [MURATA89]

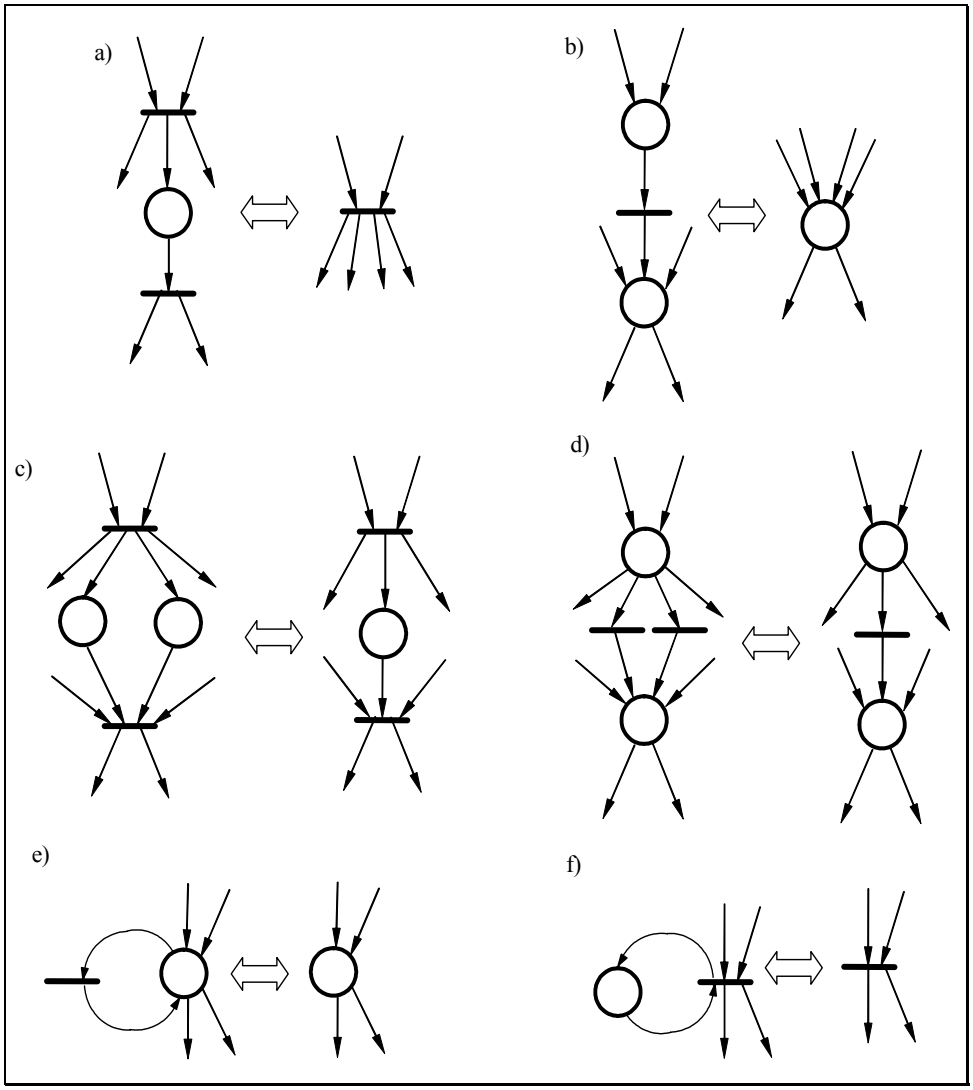
Niech sieć Petriego $PN = (P, T; F)$ będzie siecią swobodnego wyboru strukturalnie żywą i strukturalnie bezpieczną. Sieć $<N, M_0>$ jest żywa wtedy i tylko wtedy, gdy wszystkie P-niezmienniki sieci są oznakowane w znakowaniu początkowym (tzn. $\forall Y \gneq 0, Y^T \cdot C = 0 : Y^T \cdot M_0 > 0$).

Twierdzenie 5. Żywotność sieci AC i NSC [BARKAOUI, MINOUX92]

Niech sieć Petriego $PN = (P, T; F)$ będzie ograniczoną siecią asymetrycznego wyboru lub ograniczoną siecią z klasy NSC. PN spełnia własność Commoner'a, wtedy i tylko wtedy gdy każda minimalna blokada jest pułapką.

2.4. Redukcja sieci Petriego

W przypadku analizy dużych systemów, można redukować modele sieci, reprezentujące te systemy. W takiej sytuacji można zastosować twierdzenia, które pozwolą na redukcję sieci z zachowaniem jej własności, takich jak żywotność, ograniczoność i bezpieczeństwo. Istnieje wiele sposobów redukcji sieci [BOUSSIN79], [ESPERZA,SILVA91A], [ESPERZA,SILVA91B], [GIRAULT,VALK03], [JANICKI84], [MURATA89], [SURAJ,SZPYRKA99]. W rozdziale tym zostaną przedstawione tylko najprostsze.



Rys. 19. Metody redukcji sieci Petriego

Rys. 19 przedstawia sześć metod redukcji sieci:

a) scalanie kolejnych tranzycji;

b) scalanie kolejnych miejsc;

c) scalanie równoległych miejsc;

d) scalanie równoległych tranzycji;

e) usuwanie pętli tranzycji;

f) usuwanie pętli miejsc.

Twierdzenie 6. Żywa, bezpieczna, ograniczona zredukowana sieć Petriego [SURAJ,SZPYRKA99]

Niech PN=(P, T, F, M_0) i PN'=(P', T', F', M_0') będą sieciami Petriego, i przed, i po zastosowaniu metod redukcji pokazanych na Rys. 19. Sieć PN' jest żywa, (ograniczona, bezpieczna), wtedy i tyko wtedy, gdy sieć PN jest żywa (ograniczona, bezpieczna).

2.5. Kolorowane sieci Petriego

Kolorowane sieci Petriego są pewną klasą sieci Petriego. Kolory w sieciach dają możliwość nadawania pewnych atrybutów. Kolory mogą reprezentować na przykład obiekty systemu sterowania lub procesy sekwencyjne kontrolerów cyfrowych. Takie podejście pomocne jest między innymi przy dekompozycji na składowe automatowe.

Do weryfikacji algorytmu symbolicznej analizy sieci Petriego (rozdział 8) wykorzystane zostało oprogramowanie Design/CPN, w którym modelowane są systemy z wykorzystaniem kolorowanych sieci Petriego według koncepcji Jensena. W związku z tym w rozdziale tym przedstawiony został właśnie taki rodzaj kolorowanych sieci.

Poniżej zamieszczono definicję kolorowanej sieci Petriego [JENSEN92].

Definicja 31. Kolorowana sieć Petriego

Kolorowana sieć Petriego jest pojęciem abstrakcyjnym, formalnie zadanym w postaci uporządkowanej dziewiątki:

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

dla której zachodzą następujące zależności:

a) Σ jest skończonym, niepustym zbiorem typów zwanych *zbiorami kolorów*;

b) P jest skończonym zbiorem *miejsc*;

c) T jest skończonym zbiorem *tranzycji*;

d) A jest skończonym zbiorem *łuków* takim, że $P \cap T = P \cap A = T \cap A = \varnothing$;

e) N: $A \rightarrow PxT \cup TxP$ jest funkcją określająca miejsce łuków w sieci;

f) C: $P \rightarrow \Sigma$ jest funkcją przypisującą miejscom palety kolorów;

g) G jest *funkcją warunku* przypisującą do każdej tranzycji wyrażenie, zdefiniowane:

$$\forall t \in T : [Typ(G(t)) = \boldsymbol{B} \ \wedge \ Typ(Var(G(t))) \subseteq \Sigma]$$

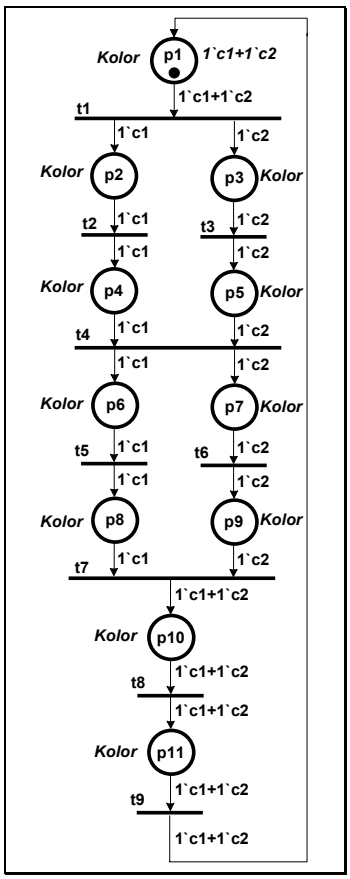
gdzie ***B*** określa typ logiczny oraz funkcja *Typ* zwraca informację o typie argumentu.

h) E jest *funkcją łuku* przypisującą do każdego łuku wyrażenie;

i) I jest *funkcją inicjującą* przypisującą do każdego miejsca *znakowanie początkowe* będące multizbiorem (ang. *multiset*) zbiorów kolorów Σ (w tym również zbiorem pustym).

Zgodnie z tradycyjną terminologią kolorowanych sieci Petriego typ danych nazywa się paletą kolorów, a aktualna wartość przypisana do znacznika — kolorem. Znaczniki różnych kolorów, zgromadzone w pewnym miejscu sieci, tworzą kolekcję znaczników. Liczbę i kolor znaczników usuwanych w chwili realizacji tranzycji definiują wyrażenia wejściowe, przypisane do jej łuków wejściowych. Liczbę i kolor znaczników tworzonych w wyniku realizacji tranzycji definiują wyrażenia wyjściowe, przypisane do jej łuków wyjściowych. Dodatkowy warunek wzbudzenia określa przypisane do tranzycji wyrażenie, nazywane dozorem [SACHA].

Rys. 20 przedstawia kolorowaną sieć Petriego według Jensena [JENSEN92]. Miejsca *p1*, *p10* i *p11*, pokolorowane zostały dwoma kolorami *c1* i *c2*, natomiast miejsca *p2*, *p4*, *p6*, *p8*, pokolorowano kolorem *c1*, miejsca *p3*, *p5*, *p7*, *p9*, kolorem *c2*.



Rys. 20. Kolorowana sieć Petriego według Jensena

3. Metody badania spełnialności formuł logicznych

W rozdziale tym zostaną poruszone zagadnienia dotyczące formuł logicznych oraz spełnialności, czyli te elementy teorii, które związane są z proponowaną symboliczną metodą analizy sieci Petriego, rozpatrywaną w niniejszej pracy. Podane zostaną również dwa algorytmy sprawdzania spełnialności formuł logicznych (formuł Horna).

3.1. Zagadnienie spełnialności

Formuła ϕ rachunku zdań jest spełnialna, gdy istnieje wartościowanie logiczne T odpowiednie dla ϕ, *takie że* $T|=\phi$ (*T* spełnia ϕ). Formuła rachunku zdań ϕ jest prawdziwa czyli jest tautologią, gdy $T|=\phi$, dla wszystkich *T* odpowiednich dla ϕ. Formuła prawdziwa musi być spełnialna, a formuła niespełnialna nie może być prawdziwa. Wartościowanie spełniające to takie, dla którego formuła przybiera wartość 1. Formułę, dla której istnieje wartościowanie spełniające, nazywana jest formułą spełnialną. Formuła rachunku zdań jest niespełnialna wtedy i tylko wtedy, gdy jej negacja jest tautologią.

Spełnialność (ang. *satisfiability, SAT*) jest następującym problemem:
czy zadana formuła ϕ w koniunktywnej postaci normalnej jest spełnialna [PAPADIMITROU00].

Twierdzenie 7. Spełnialność formuł logicznych [PAPADIMITROU00]

Problem spełnialności formuł logicznych jest problemem NP-zupełnym.

UWAGA: Jednakże istnieją klasy wyrażeń logicznych, które można rozwiązać w czasie liniowym, tzn. [MINOUX,BARKAOUI90]:

- problem 2-spełnialności (*2-SAT*), gdzie każda klauzula zawiera co najwyżej dwa literały;
- problem spełnialności formuł Horna (*HORN-SAT*).

3.2. Podstawowe pojęcia dotyczące formuł Horna

W rozdziale tym zostały zamieszczone podstawowe pojęcia dotyczące formuł Horna i spełnialności formuł Horna oraz podane przykładowe algorytmy sprawdzające

spełnialność formuł [BARKAOUI,MINOUX92], [MINOUX88], [MINOUX,BARKAOUI90], [PAPADIMITROU02], [DOWLING,GALLIER76].

Definicja 32. Formuła rachunku zdań

Formuła rachunku zdań może być:

- zmienną zdaniową taką jak x_p, gdzie x_p należy do zbioru zmiennych zdaniowych X;
- wyrażeniem postaci $\neg\phi_i$, gdzie ϕ_i jest formułą;
- wyrażeniem postaci $(\phi_i \vee \phi_j)$, gdzie ϕ_i i ϕ_j są formułami;
- wyrażeniem postaci $(\phi_i \wedge \phi_j)$, gdzie ϕ_i i ϕ_j są formułami.

Definicja 33. Literał

Literałem jest nazywany symbol zdaniowy (inaczej *pozytywny literał*) lub jego negację (*negatywny literał*).

Definicja 34. Podstawowa formuła Horna

Podstawową formułą (*klauzulą*) *Horna KH* nazywamy dysjunkcję literałów, z co najwyżej jednym pozytywnym literałem.

UWAGA: W szczególności klauzula Horna może składać się z następujących elementów:

a) jednego pozytywnego literału, np. $KH_i = x_p$;

b) jednego negatywnego literału, np. $KH_j = /x_q$;

c) dysjunkcji negatywnych literałów, np. $KH_k = /x_r \vee /x_s$;

d) dysjunkcji negatywnych literałów i jednego pozytywnego literału, np. $KH_l = x_t \vee /x_u \vee /x_v$.

Klauzule (a) oraz (d) zwane są również implikacjami, gdyż mogą zostać przekształcone do postaci:

$$(x_1 \wedge x_2 \wedge \ldots \wedge x_r) => y$$

gdzie y jest literałem pozytywnym.

W przypadku koniunkcji bez literałów negatywnych, klauzula ma postać:

$$\textbf{\textit{true}} => x_p.$$

Klauzule można również zapisać w postaci [KOWALSKI89]:

a) $=> x_p$

b) $x_q =>$

c) $x_r \vee x_s =>$

d) $x_u \vee x_v => x_t$

Definicja 35. Formuła Horna

Formuła Horna FH jest koniunkcją podstawowych formuł Horna.

UWAGA: Przykładowo formuła Horna składająca się z trzech literałów (x_1, x_2, x_3) może mieć postać:

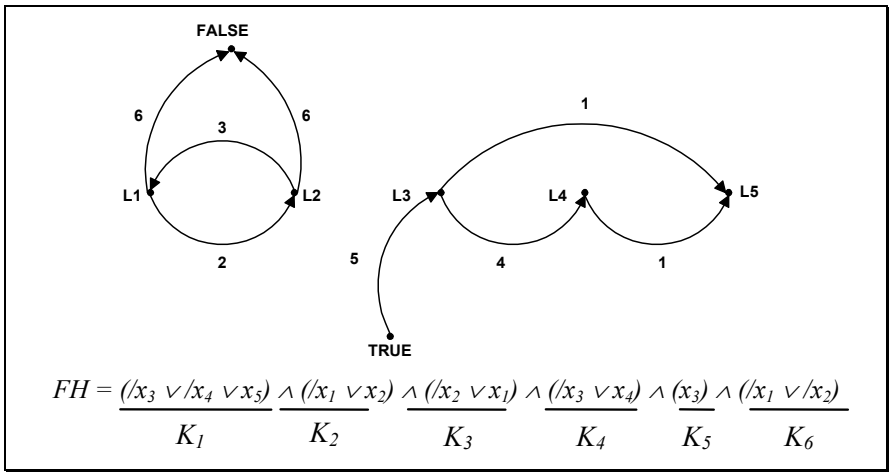
$$FH = (x_1) \wedge (x_1 \vee /x_2) \wedge (/x_2) \wedge (/x_2 \vee /x_3) \wedge (x_1 \vee /x_2 \vee /x_3)$$

Niektóre algorytmy sprawdzania spełnialności formuł Horna bazują na analizie grafu etykietowanego skojarzonego z daną formułą. W tym celu zamieszczona została definicja grafu etykietowanego oraz sposób jego konstruowania.

Definicja 36. Graf etykietowany formuły Horna

G_A jest *etykietowanym grafem skierowanym* o $n+2$ węzłach i zbiorze etykiet $[E]$. Występuje w nim odrębny węzeł dla każdego literału zawartego w formule F oraz dwa dodatkowe węzły dla wartości *prawda* (*true*) i dla wartości *fałsz* (*false*). Liczba n węzłów odpowiada liczbie literałów w formule. Zbiór etykiet E odpowiada numerowi klauzuli w formule. Konstrukcja grafu przedstawiona jest w następujący sposób:

- Jeżeli i-ta klauzula Horna w formule *FH* jest pojedynczym, pozytywnym literałem L_p (np. x_3 w klauzuli K_5), to w grafie G_A występuje krawędź z węzła *true* do węzła x_p o etykiecie i.
- Jeżeli j-ta klauzula Horna w formule *FH* ma postać $/x_1 \vee \ldots \vee /x_r$ (np. klauzula K_6), to w grafie występuje r krawędzi o etykiecie j z węzłów $x_1, \ldots, x_r$, prowadzących do węzła *false*.
- Jeżeli k-ta klauzula Horna w formule *FH* ma postać $/x_1 \vee \ldots \vee /x_s \vee x_t$ (np. klauzula K_1), to w grafie występuje s krawędzi z węzłów $x_1, \ldots, x_s$ prowadzących do węzła x_t o etykiecie k.



Rys. 21. Graf skojarzony z formułą Horna FH

Istnieje efektywny algorytm, który na podstawie postaci implikacyjnej klauzuli Horna sprawdza spełnialność tej klauzuli. Algorytm, sprawdzający spełnialność formuł Horna, działa w następujący sposób. Na początku przypisywana jest wszystkim zmiennym wartość *false*. Następnie wybierana jest dowolna niespełniona klauzula, mającą literał bez negacji i odpowiedniej zmiennej przypisywana jest wartość *true*. Operacja kontynuowana jest dopóki istnieją takie klauzule. Formuła jest spełnialna wtedy i tylko wtedy, gdy wektor wartości zmiennych, który powstanie w wyniku tego algorytmu spełnia wszystkie klauzule. Algorytm ten został dokładnie opisany w [PAPADIMITROU02].

UWAGA: Problem spełnialności formuł Horna nazywany jest HORN-SAT.

Twierdzenie 8. [PAPADIMITROU02]

Problem HORN-SAT należy do klasy P.

3.3. Algorytm LTUR

Do sprawdzenia spełnialności formuł Horna można zastosować również algorytm LTUR (ang. *Linear-Time Unit Resolution*). Metoda ta bazuje na analizie etykietowanego grafu skierowanego skojarzonego z daną formułą Horna (definicja 36) [DOWLING,GALIER84], [MINOUX88]. Zostanie ona opisana w sposób krokowy.

Dana jest formuła Horna w postaci:

$$F = K_1 * K_2 * ... * K_j,$$

gdzie K_j - klauzula Horna.

1. W pierwszym kroku należy ustawić wartości początkowe. Dla każdej klauzuli K_j *(j=1,2,...,m)* obliczane jest *vj* będące liczbą negatywnych literałów w klauzuli (w czasie działania algorytmu, oznacza aktualną liczbę negatywnych literałów w klauzuli K_j, które jeszcze nie mają wartości true). Następnie ustawiane są wartości dla zbioru S zawierającego numery klauzul zawierających pojedyncze pozytywne literały (przy pierwszym przejściu algorytmu), oraz pozostałe rodzaje klauzul (w pozostałych przejściach algorytmu). Należy również ustawić wartość początkową dla poszczególnych literałów, która równa się 0 (false) ($val(k) \leftarrow 0$, $\forall k=1,2,...,n$).
2. Następnym krokiem jest sprawdzenie, czy zbiór S jest pusty, jeżeli tak, to należy zakończyć działanie algorytmu. Formuła jest spełnialna, a pierwiastkami wyrażenia są wartości wszystkich literałów tworzących formułę zawarte w *val*. Jeżeli zbiór S nie jest zbiorem pusty, wówczas należy pobrać pierwszy numer klauzuli (oraz usunąć go ze zbioru S).

3. Jeżeli pobrana klauzula nie posiada pozytywnego literału, wówczas formuła nie jest spełnialna. W przeciwnym przypadku analizowany jest pozytywny literał danej klauzuli.
4. Jeżeli nie istnieje łuk łączący dany literał z węzłem nie należącym do aktualnie badanej klauzuli, należy wrócić do kroku 3. W przeciwnym przypadku dla klauzuli, dla której istnieje łuk łączący węzeł reprezentujący jej literał z danym literałem, wartość *vj* dla tej klauzuli jest zmniejszana o 1.
5. Jeżeli *vj* jest różne od zera to należy powrócić do kroku 4. W przeciwnym przypadku, jeżeli drugi węzeł łuku jest węzłem false, wówczas formuła nie jest spełnialna. Jeżeli natomiast drugi węzeł nie jest węzłem false i wartość drugiego węzła jest równa zero, to klauzulę tę należy dodać do zbioru *S*, a wartość drugiego węzła ustawić na 1. W innych przypadkach należy powrócić do kroku 4.

Algorytm LTUR jest ma złożoność obliczeniową liniową względem liczby literałów w klauzuli [DOWLING,GALIER84].

3.4. Podsumowanie

Podsumowując rozdział rozpatrujący metody badania spełnialności formuł logicznych, można stwierdzić, że ogólny problem spełnialności formuł logicznych jest problemem NP-zupełnym [PAPADIMITROU02]. Jednakże wykorzystując pewne ograniczenia nałożone na formuły logiczne, problem ten można sprowadzić do problemu w klasie P. Przykładem takich równań są formuły Horna. W rozdziale zostały podane dwa algorytmy sprawdzania spełnialności formuł w czasie liniowym. Oprócz odpowiedzi, czy dana formuła jest prawdziwa, podawane jest również jedno rozwiązanie. Niemniej jednak podejście to jest niewystarczające do rozwiązania problemu postawionego w niniejszej rozprawie.

4. Modelowanie i synteza współbieżnych sterowników cyfrowych

W rozdziale tym zostaną przedstawione sposoby modelowania współbieżnych sterowników cyfrowych z wykorzystaniem interpretowanych sieci Petriego i sieci SFC [ADAMSKI90], [HALANG89], [KOZŁOWSKI,ET.AL.95], [KOZŁOWSKI95], [LEWIS95], [WOLAŃSKI98], [JIANG,HOLDING96], [KALINOWSKI84]. Kryterium wyboru sposobu modelowania jest łatwość analizy i syntezy z wykorzystaniem nowoczesnych narzędzi CAD.

4.1. Interpretowane sieci Petriego

W odróżnieniu od klasycznego automatu sekwencyjnego, automat współbieżny znajduje się równocześnie w jednym lub kilku stanach wewnętrznych. Maksymalne zbiory równocześnie występujących stanów lokalnych definiują stany globalne automatu. Dowolny podzbiór równocześnie występujących stanów lokalnych nazywany jest stanem częściowym. W automatach współbieżnych rozpatruje się zamiast globalnych funkcji przejść lokalne relacje, wiążące ze sobą wewnętrzne stany częściowe, aktualne i następne, oraz odpowiednie stany wejść i wyjść automatu. Interpretowana sieć Petriego jest obrazową formą przedstawienia automatu współbieżnego [ADAMSKI90].
Poniżej zostały zamieszczone definicje dotyczące interpretowanych sieci Petriego.

Definicja 37. Sieć Petriego z wejściami
Siecią Petriego z wejściami nazywana jest uporządkowana trójka:

$PN_I = (PN, X, \rho)$,

gdzie:

PN żywa i bezpieczna sieć Petriego;
X jest zbiorem stanów wejść;
$\rho: T \to 2^X$ jest funkcją, która każdej tranzycji przyporządkowuje jednoznacznie podzbiór stanów wejść $X(t)$.

Symbol 2^X oznacza zbiór wszystkich możliwych podzbiorów X.

Definicja 38. Sieć Petriego z wyjściami (typu Moore'a)
Siecią Petriego z wyjściami typu Moore'a nazywana jest uporządkowana trójka:

$PN_{O1} = (PN, Y, \lambda)$,

gdzie:

PN żywa i bezpieczna sieć Petriego;

Y jest zbiorem stanów wyjść;

λ: $[M_0\rangle \rightarrow 2^Y$ jest funkcją, która każdemu znakowaniu sieci M przyporządkowuje jednoznacznie pewien stan wyjść $Y(M)$.

Definicja 39. Sieć Petriego z wyjściami (typu Mealy'ego)

Siecią Petriego z wyjściami typu Mealy'ego nazywana jest uporządkowana czwórka:

$PN_{O2} = (PN, X, Y, \gamma)$,

gdzie:

PN żywa i bezpieczna sieć Petriego;

X jest zbiorem stanów wejść;

Y jest zbiorem stanów wyjść;

γ: $(M \times X) \rightarrow 2^Y$ jest funkcją, która przy znakowaniu sieci M określonemu podzbiorowi stanów wejść przyporządkowuje jednoznacznie podzbiór stanów wyjść $Y(t)$.

Podzbiór stanów wejść określa się najczęściej w postaci kostki lub odpowiedniej formuły boolowskiej [ŁUBA,ZBIERZCHOWSKI02].

Definicja 40. Interpretowana sieć Petriego

Interpretowaną siecią Petriego nazywana jest uporządkowana szóstka:

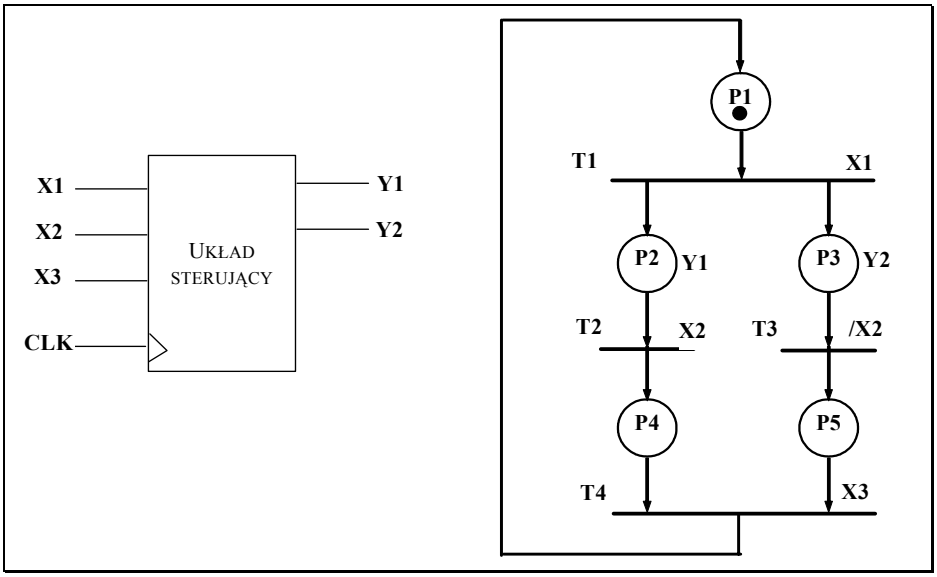
$PN_{IO} = (PN, X, Y, \rho, \lambda, \gamma)$,

gdzie poszczególne symbole zgodne są z odpowiednimi symbolami występującymi w sieciach z wejściami oraz wyjściami.

Miejsca sieci reprezentują stany lokalne sterownika, tranzycje ich zmiany. Wszystkie miejsca oznakowane w tym samym czasie, definiują stan globalny sterownika. Znaczniki wyznaczą, które stany sterownika są aktywne w danym czasie. Oznakowanie początkowe reprezentuje stan początkowy układu. Każda tranzycja typu rozwidlenie (ang. *fork*), tzn. tranzycja posiadająca więcej niż jedno miejsce wyjściowe, jest początkowym punktem dla równoległych procesów, natomiast każda tranzycja typu złączenie (ang. *join*), tzn. tranzycja posiadająca więcej niż jedno miejsce wejściowe, synchronizuje procesy, które łączą się w tym punkcie. Do specyfikacji automatów współbieżnych wykorzystywane są najczęściej synchroniczne, interpretowane sieci Petriego [KOZŁOWSKI,ET.AL.95].

W rozprawie doktorskiej przyjmuje się mniej restrykcyjną formę implementacji synchronicznej realizacji algorytmów, gdyż algorytmy specyfikowane w sposób ogólny jako asynchroniczne [CORTADELLA,ET.AL.02], będą jednak realizowane w postaci synchronicznych układów sekwencyjnych, taktowanych tym samym zegarem.

Rys. 22 przedstawia sterownik o trzech wejściach *X1*, *X2*, *X3* i dwóch wyjściach *Y1*, *Y2*, oraz model interpretowanej sieci Petriego reprezentującej jego funkcjonowanie.



Rys. 22. Model interpretowanej sieci Petriego

Układ cyfrowy może być rozpatrywany jako system składający się z części operacyjnej i części sterującej (czyli sterownika logicznego) [MAJEWSKI98]. Przetwarzaniem danych w części operacyjnej steruje sterownik, który rozpoznaje stan systemu i wytwarza odpowiednie sygnały sterujące, odpowiednio synchronizując pracę części operacyjnej. Często pewne dane są przetwarzane w niezależnych procesach, które są wykonywane przez niezależnie pracujące części systemu. Dlatego sterowniki muszą być przystosowane do sterowania procesami równoległymi, co powoduje, że opis sterownika z wykorzystaniem sieci Petriego jest bardzo efektywny [WOLAŃSKI98].

4.2. Sieci SFC

Sieć SFC jest językiem graficznym, objętym normą IEC [IEC92]. Z wykorzystaniem sieci SFC można przedstawić w sposób współbieżny operacje sekwencyjne. Proces binarny jest reprezentowany za pomocą poprawnie zdefiniowanych kolejnych kroków połączonych z tranzycjami. Elementy sieci SFC są przydatnym narzędziem graficznym opisu sterowników logicznych [LEWIS95].

Definicja 41. Sieć SFC [HALANG89]

Sieć SFC definiowana jest w postaci uporządkowanej czwórki *SFC = (S, T, L, I)*
gdzie:

- *S* – niepusty, skończony zbiór kroków;
- *T* – niepusty, skończony zbiór tranzycji;
- *L* – niepusty, skończony zbiór połączeń pomiędzy krokiem a tranzycją lub tranzycją a krokiem;
- $I \subset S$ - zbiór kroków inicjujących.

Zgodnie z normą IEC 1131, zbiór kroków inicjujących został ograniczony do jednego kroku.

Krok określający stan procesu, oraz tranzycja, określająca zmianę stanu wraz z warunkiem tej zmiany, są dwoma podstawowymi i niezbędnymi elementami sieci. Wykorzystanie tylko tych dwóch komponentów daje realny obraz działania sterownika. Dodatkowy składnik sieci, taki jak blok działania, ukazuje bardzo szczegółowy, techniczny opis układu [ADAMSKI98], [LEWIS95]. Pozwala on określić rodzaj akcji (kwalifikator), symbol zwrotnej zmiennej logicznej lub opisać zachowanie układu sterującego z wykorzystaniem języka ST (*Structured Text*), IL (*Instruction List*), a także reprezentację akcji boolowskiej [IEC92], [ADAMSKI,CHORAŃ00].

Elementy S i T są reprezentowane jako węzły grafu. Zbiór kroków inicjujących rozpoczyna proces i determinuje stan inicjujący procesu. Z każdym krokiem powiązana jest akcja, której wykonanie następuje wtedy, gdy krok jest aktywny. Z każdą tranzycją powiązany jest warunek boolowski: jeżeli krok poprzedzający tranzycję jest aktywny i warunek spełniony, to następuje dezaktywacja kroku poprzedzającego tranzycję i aktywacja kroku następującego po przejściu [IEC92], [LEWIS95].

4.2.1. Sieci SFC a sieci Petriego

Sposób przedstawiania procesu sterowania za pomocą sieci SFC jest podobny do sposobu reprezentacji tego samego procesu za pomocą interpretowanych sieci Petriego [ADAMSKI,CHODAŃ00], [WĘGRZYN,CHODAŃ00]. Podobne są również reguły modelowania, natomiast inna jest reprezentacja graficzna podstawowych elementów obu sieci: kroków, kroku początkowego, realizacji procesów współbieżnych. Podstawowe porównanie elementów grafu sieci SFC i sieci Petriego zostały przedstawione na Rys. 23.

W sieciach SFC, z każdym krokiem może być skojarzonych jedna lub wiele bloków akcji, zawierających opis zachowania występujący w czasie aktywacji danego kroku. Opis akcji może być podany w językach: ST, FBD, LD oraz IL, zdefiniowanych w

normie IEC 1131-3. Jednakże takiej specyfikacji sieci SFC nie można odwzorować bezpośrednio w sieciach Petriego.

Rys. 23. Porównanie elementów graficznych sieci SFC i sieci Petriego

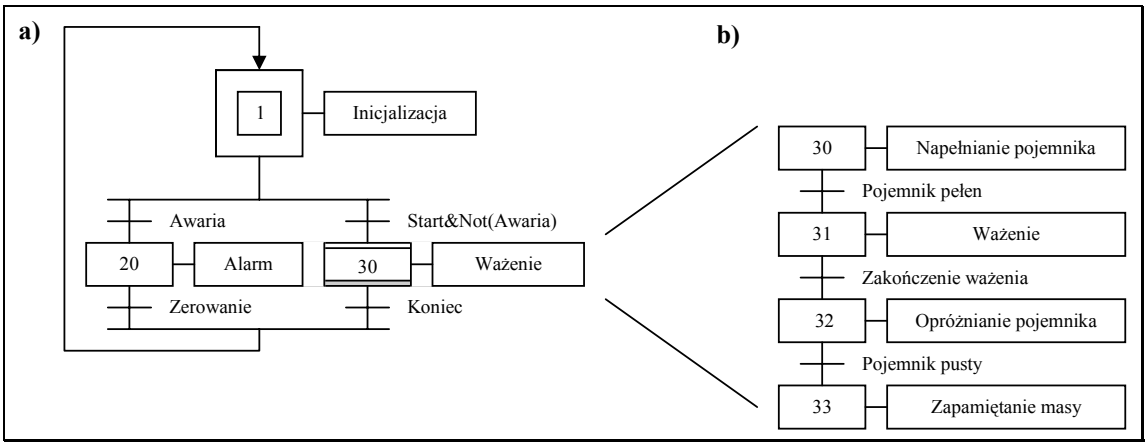
Podane w rozprawie przekształcenia dotyczą jedynie prostszych przypadków, gdzie odpowiedniość między siecią SFC a siecią Petriego jest oczywista. Złożone przypadki (priorytet, itp.) przedstawiono w [ADAMSKI,CHODAŃ00].

W przypadku ogólnym sieci SFC przekształcane są do postaci powiązanych sieci Petriego lub sieci Petriego z łukami zabraniającymi i zezwalającymi [ADAMSKI,MONTEIRO94].

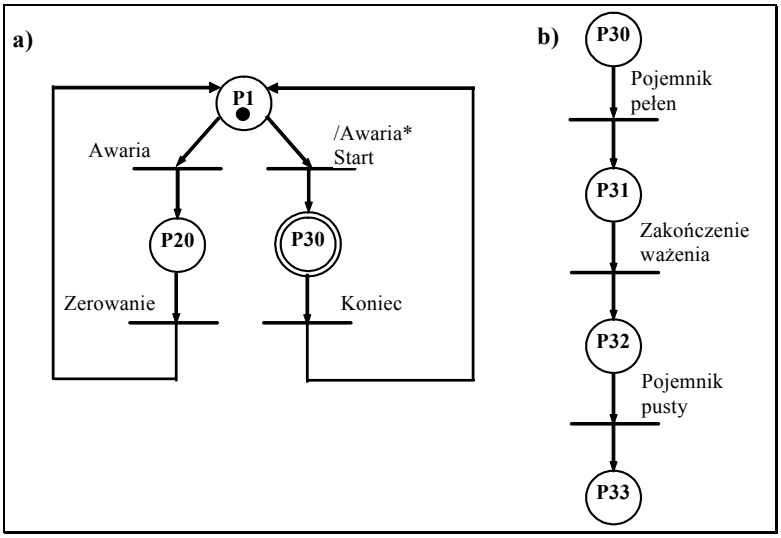
4.2.2. Przykład transformacji sieci SFC na sieć Petriego

Metoda transformacji modelu sieci SFC na sieć Petriego zostanie przedstawiona na przykładzie procesu ważenia substancji. Model opisanego procesu reprezentuje sieć SFC (Rys. 24). Sieć składa się z dwóch kroków i makroetapu (Rys. 24a), które reprezentują inicjalizację procesu, alarm oraz proces ważenia. Makroetap składa się z czterech kroków (Rys. 24b), które przedstawiają cały proces ważenia, czyli: napełnianie pojemnika, ważenie, opróżnianie pojemnika oraz zapamiętywanie masy [ADAMSKI,CHODAŃ00].

Na Rys. 25 pokazano interpretowaną sieć Petriego po translacji sieci SFC (Rys. 24) z wykorzystaniem powyżej przedstawionych zasad (Rys. 23).

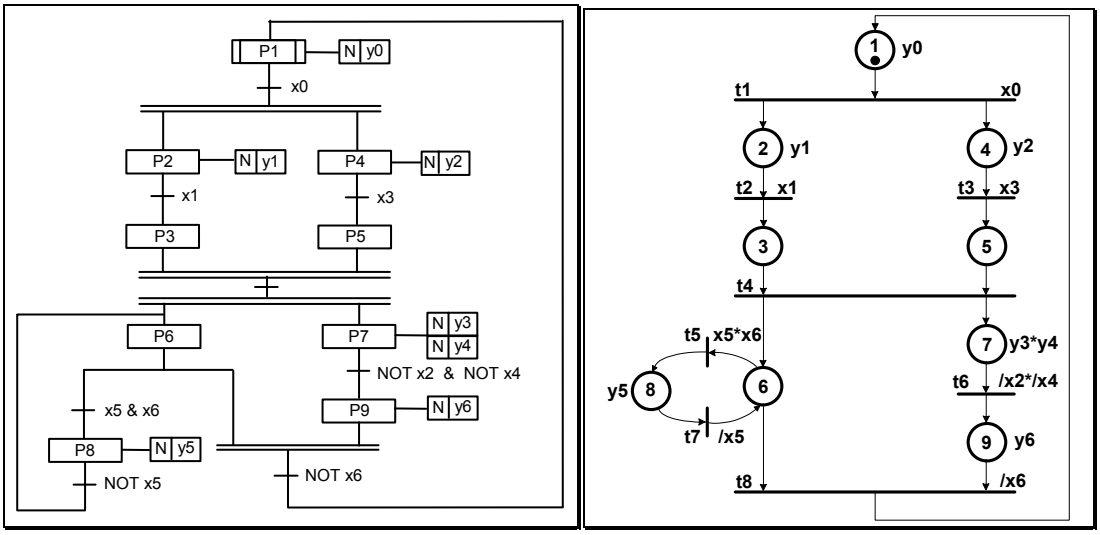


Rys. 24. Sieć SFC opisująca proces ważenia



Rys. 25. Sieć Petriego po translacji sieci SFC

Na Rys. 26 przedstawiono sieć SFC i odpowiadającą jej sieć Petriego [ADAMSKI98A]. Przykład zamieszczono w celu prezentacji sposobu translacji współbieżnej sieci SFC na sieć Petriego.



Rys. 26. Sieć SFC i odpowiadająca jej sieć Petriego

4.3. Podsumowanie

W rozdziale przedstawiono interpretowane automatowo sieci Petriego zwane również sieciami Petriego sterowania. Sieci te umożliwiają łatwą i poglądową specyfikację automatów cyfrowych w sposób podobny jak interpretowane grafy skierowane nazywane powszechnie grafami stanów. Sieć Petriego pozwala w sposób jawny opisywać procesy współbieżne oraz skomplikowane sposoby koordynacji tych procesów, odzwierciedlając precyzyjniej zależności przyczynowo-skutkowe między warunkami poszczególnych zdarzeń. Graf SFC potraktowano jako szczególny przypadek sieci Petriego.

5. Przegląd wybranych metod analizy sieci Petriego

Metody analizy sieci Petriego można podzielić na dwie klasy: metody bazujące na grafie znakowań oraz metody bazujące na strukturalnej analizie sieci Petriego [MURATA89].

Pierwsza klasa metod polega na tworzeniu pełnego grafu znakowań dla badanej sieci Petriego a następnie jego analizie [PETERSON81], [MURATA89]. Tak zbudowany graf znakowań daje pełną informację o własnościach statycznych i dynamicznych sieci. Wraz z liczbą procesów współbieżnych reprezentowanych przez sieć, bardzo silnie wzrasta złożoność. Istnieją również metody tworzenia grafu znakowań dla specyficznych fragmentów sieci Petriego [KARATKEVICH,ET.AL.00], jednakże i te metody charakteryzują się dużą złożonością obliczeniową. Dla pewnych klas sieci można zastosować bardziej efektywne metody bazujące na grafach znakowań, przedstawionych za pomocą równań charakterystycznych z wykorzystaniem grafów BDD [BILIŃSKI,ET.AL.95], [BILIŃSKI96], [PASTOR,ET.AL.94]. Złożoność obliczeniowa tych metod jest taka sama, jak metody analizy pełnego grafu znakowań, jednakże w przypadku grafów BDD można korzystać z uniwersalnego oprogramowania o dużej niezawodności, przeznaczonego do weryfikacji dużych układów cyfrowych. Zaletą metod opartych na grafie znakowań jest możliwość zlokalizowania miejsca sieci, w którym naruszono poprawność konstrukcyjną sieci, wadą konieczność przeglądania dużej liczby wierzchołków (eksplozja kombinatoryczna).

Druga klasa metod analizy sieci bazuje na analizie strukturalnej. Analiza ta polega na znalezieniu relacji pomiędzy zachowaniem sieci a jej strukturą. W ramach tej klasy metod można wyróżnić algorytmy bazujące na algebrze liniowej (metody wykorzystujące macierz incydencji sieci), które są niezależne od znakowania początkowego [ZAKREVSKIJ85], [ZAKREVSKIJ87], [MURATA89], [GIRAULT,VALK03] algorytmy opierające się na teorii grafów, czyli badaniu blokad i pułapek, oraz zależności między nimi [COMMONER72], [MURATA89], [BARKAOUI,MINOUX92].

5.1. Analiza przestrzeni stanów metodą grafu znakowań

Sieć Petriego może być przedstawiona w postaci grafu znakowań. Graf znakowań konstruowany jest dla każdego możliwego znakowania sieci, które może być osiągnięte ze znakowania początkowego i daje sekwencję realizacji tranzycji dla każdego znakowania. Graf znakowań jest na ogół konstruowany bez względu na ograniczenia nakładane przez interpretację sieci. W grafie tym węzły opisane są miejscami, które w danym momencie są aktywne, natomiast łuki wyjściowe węzła, opisane są tranzycjami, które muszą zostać zrealizowane, aby dane miejsca mogły być aktywne. Graf znakowań uwzględnia informacje o znakowaniu sieci Petriego, co pozwala na analizę sieci w zależności od początkowego znakowania sieci. Analiza grafu pozwala również na dokładniejszą lokalizację tego fragmentu sieci, w którym wystąpił błąd.

Algorytmy wykorzystujące graf znakowań mają charakter ogólny i są odpowiednie dla wszystkich klas sieci. Charakteryzują się one dużą złożonością obliczeniową i zajętością pamięci, szczególnie dla sieci z dużą liczbą gałęzi równoległych, stąd w niektórych przypadkach praktycznych są mało przydatne.

5.1.1. Analiza pełnego grafu znakowań

W rozdziale tym zostanie przedstawiona metoda analizy sieci Petriego na podstawie analizy pełnego grafu znakowań [PETERSON81], [MURATA89].

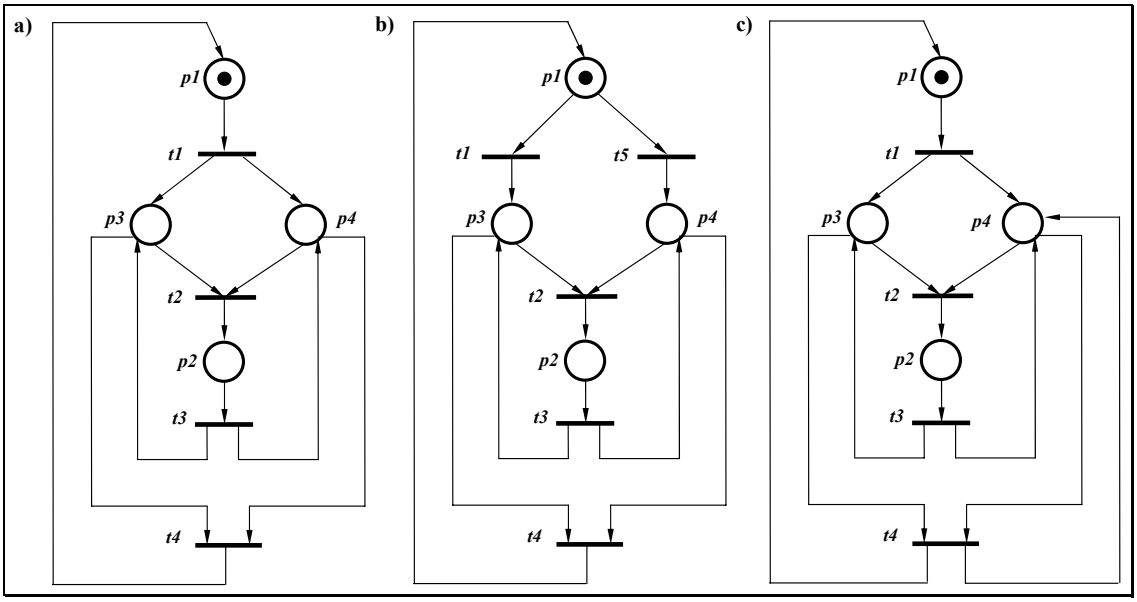
Analiza grafu znakowań rozpoczyna się od momentu jego konstruowania. W trakcie konstruowania można na bieżąco badać, czy sieć nie jest bezpieczna i w momencie wykrycia tego błędu można przerwać konstruowanie grafu. Dla każdego możliwego znakowania tworzony jest odrębny węzeł w grafie znakowań. Dlatego metoda ta jest mało efektywna dla sieci z dużą liczbą stanów współbieżnych.

Analiza pełnego grafu znakowań może zostać opisana w następujących krokach:

1. Podczas konstruowania grafu znakowań sprawdzane jest każde znakowanie, czy jest ono bezpieczne, jeżeli znalezione zostanie znakowanie niebezpieczne, to wówczas należy zakończyć konstruowanie grafu. Sieć Petriego, dla której konstruowany był graf, nie jest bezpieczna.
2. W kolejnym kroku, należy sprawdzić, czy skonstruowany graf znakowań jest silnie spójny. Jeżeli graf znakowań nie jest silnie spójny, wówczas sieć Petriego, dla której został skonstruowany graf nie jest żywa.
3. W ostatnim kroku sprawdzane jest występowanie każdej z tranzycji sieci Petriego w grafie znakowań. Jeżeli przynajmniej jedna tranzycja nie występuje w tym grafie, wówczas sieć Petriego nie jest żywa.

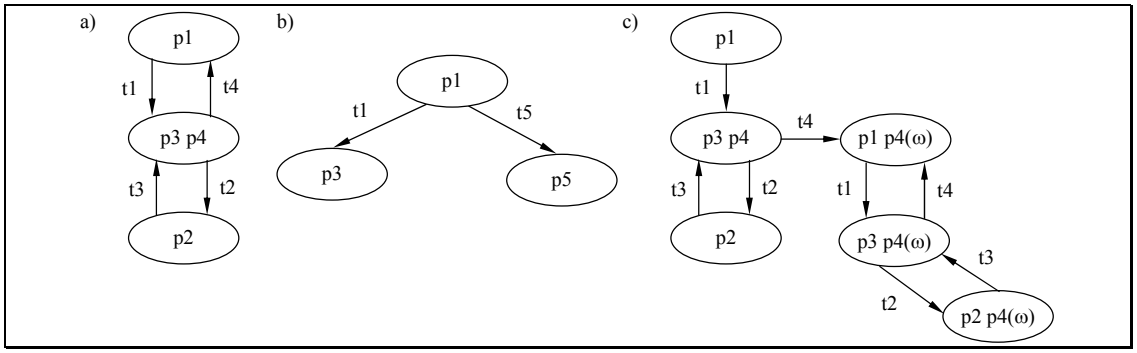
Złożoność obliczeniowa opisanego algorytmu jest wykładnicza, gdyż krok pierwszy algorytmu, czyli konstruowanie i sprawdzanie bezpieczeństwa sieci, ma złożoność obliczeniową wykładniczą [PASTOR,ET.AL.94].

W celu zilustrowania prezentowanej metody można rozpatrzyć przykład sieci z Rys. 27.



Rys. 27. Przykład sieci Petriego a) żywej, b) nieżywej, c) niebezpiecznej

Żywotność sieci Petriego można opisać również poprzez stwierdzenie, że sieć jest wówczas żywa, gdy w każdej silnie spójnej części grafu znakowań istnieje dla każdej tranzycji t∈T krawędź znakowana przez t [STARKE87].



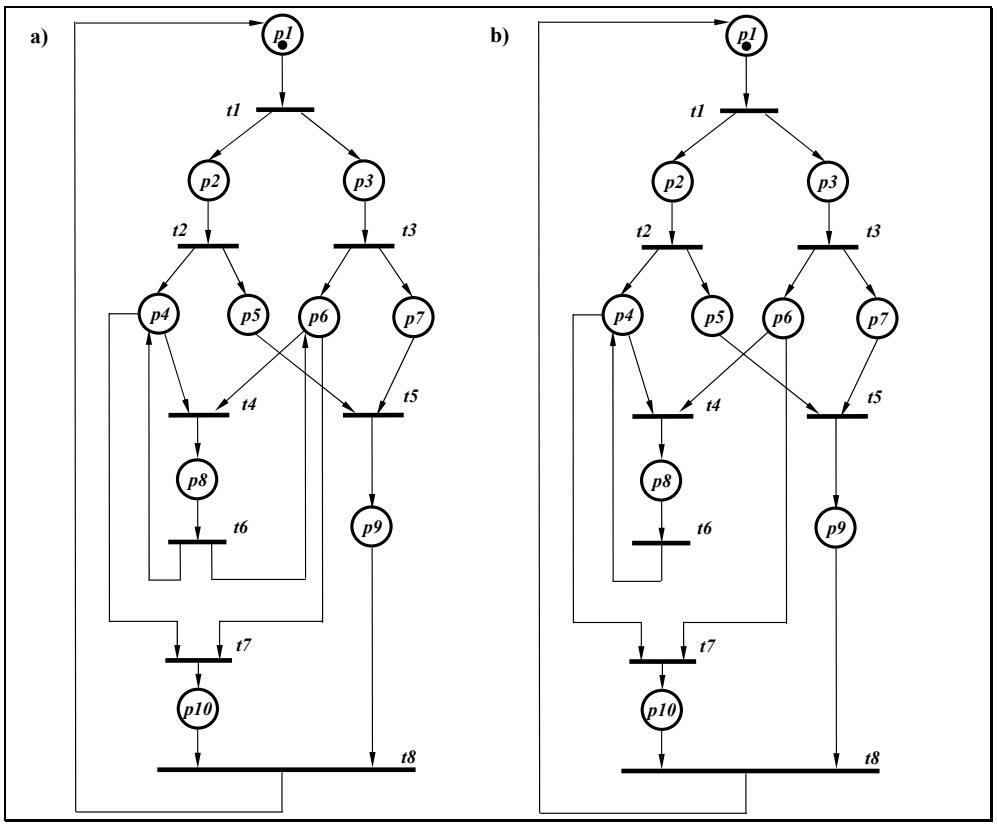
Rys. 28. Graf znakowań dla sieci z Rys. 27 a i b

Na podstawie wyznaczonego grafu znakowań (Rys. 28a) dla sieci z Rys. 27a można określić, że sieć Petriego, dla której został przygotowany ten graf jest siecią żywą i bezpieczną. Otrzymany graf jest silnie spójny oraz wszystkie tranzycje występują w nim. Przygotowany graf znakowań (Rys. 28b) dla sieci z Rys. 27b nie jest grafem silnie spójnym, gdyż występują w nim wierzchołki bez tranzycji wyjściowych, dlatego sieć ta nie jest żywa. Sieć z Rys. 27c nie jest siecią bezpieczną, gdyż po powtórnym wystąpieniu tranzycji *t1* miejsce *p4* jest podwójnie oznakowane i będzie gromadzić znaczniki: *p4* (ω).

5.1.2. Analiza zredukowanego grafu znakowań

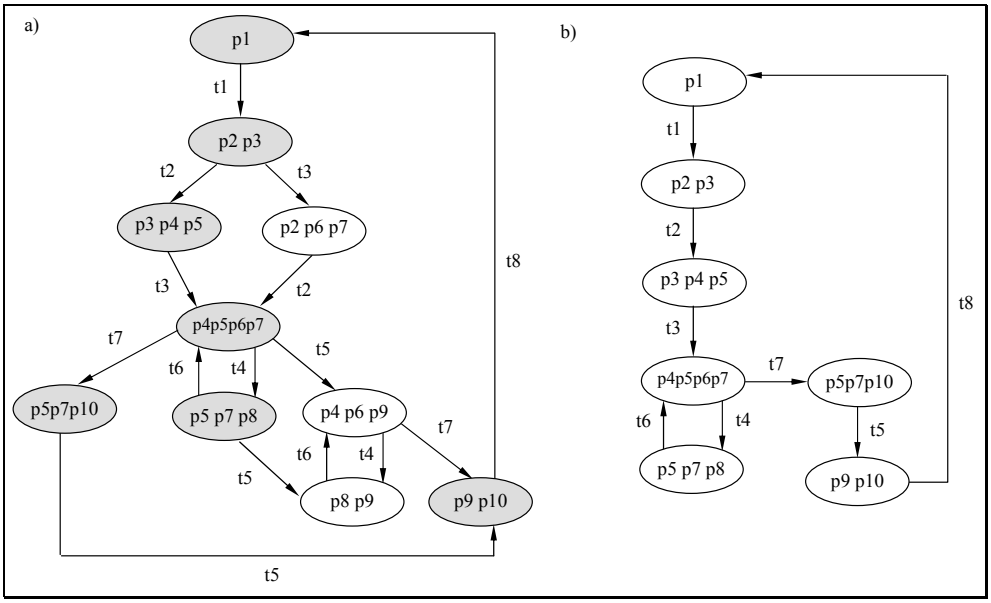
W podrozdziale tym zostanie przedstawiony algorytm badający własności sieci Petriego na podstawie analizy zredukowanego grafu znakowań [KARATKEVICH,ET.AL.00]. Metoda ta polega na tworzeniu podgrafu pełnego grafu znakowań (tzn. jego istotnej dla analizy, części), który jest następnie weryfikowany w podobny sposób jak pełny graf znakowań. W trakcie konstruowania grafu pominięte są te kombinacje znakowań, które uzyskiwane są na podstawie ewolucji gałęzi równoległych. Najpierw brane jest pod uwagę tylko znakowanie uzyskane na podstawie jednej z gałęzi, a następnie dla tego znakowania obliczane są wszystkie kombinacje ze znakowaniami z kolejnej gałęzi. Jeżeli po przeanalizowaniu tak sporządzonego podgrafu pełnego grafu znakowań stwierdzono, że sieć opisana tym podgrafem jest żywa i bezpieczna, to wówczas można stwierdzić, że cała sieć jest żywa i bezpieczna. Opisany algorytm zastosować można dla α-sieci. Opisywana metoda ma nadal złożoność obliczeniową wykładniczą, jednakże w porównaniu ze złożonością obliczeniową klasycznej metody analizy pełnego grafu znakowań, jest znacznie szybsza (szczególnie dla przypadków sieci z dużą liczbą miejsc współbieżnych) [KARATKEVICH,ET.AL.00].

W celu bardziej poglądowego zobrazowania metody, sieci z Rys. 27 zostały rozbudowane poprzez wprowadzenie dodatkowych miejsc oraz tranzycji (Rys. 29).

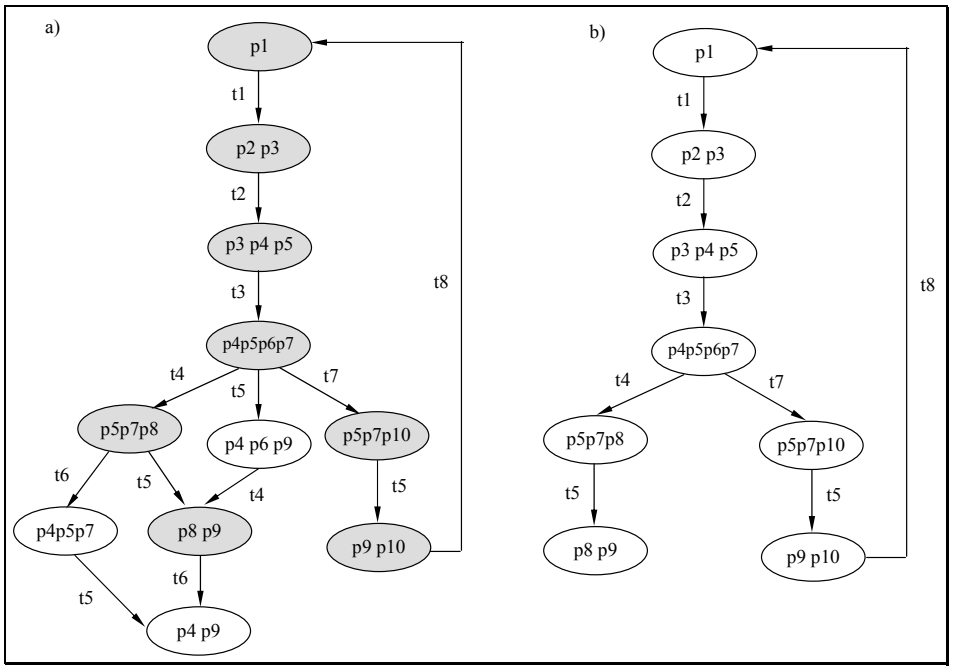


Rys. 29. Przykład sieci Petriego a) żywej, b)nieżywej

Na Rys. 30a i b przedstawione zostały pełny graf znakowań oraz podgraf znakowań dla sieci z Rys. 29a. Rys. 31a i b przedstawia pełny graf znakowań oraz podgraf znakowań dla sieci z Rys. 29b. W pełnych grafach znakowań zacienione zostały tylko te stany globalne, które występują w rozpatrywanych podgrafach znakowań wystarczających do analizy badanych sieci.



Rys. 30. a) Pełny graf znakowań b) podgraf znakowań, dla sieci z Rys. 29a



Rys. 31 a) Pełny graf znakowań b) podgraf znakowań, dla sieci z Rys. 29 b

Sieć (Rys. 29a), która jest analizowana na podstawie niepełnego grafu znakowań (Rys. 30b) jest siecią żywą, co potwierdza również bardziej złożony pełny graf znakowań, zamieszczony na Rys. 30a. Znacznie prostszy, niepełny graf znakowań sieci z Rys. 29b,

zamieszczony jako Rys. 31b, tak samo jak graf znakowań z Rys. 31a, pozwala już stwierdzić, że sieć ta nie jest żywa.

5.2. Analiza strukturalnych własności sieci metodami algebraicznymi

W podrozdziale tym zostaną przedstawione metody badania strukturalnej żywotności i ograniczoności sieci Petriego. Opisywane algorytmy operują na wyrażeniach logicznych oraz macierzach incydencji. W części 3.3 zaprezentowano metodę opierającą się na rozwiązywaniu wyrażeń logicznych opisujących sieci Petriego. Kolejne sekcje przedstawiają metody operujące na macierzy incydencji, czyli na wyszukiwaniu P-niezmienników (5.2.1) oraz wyznaczaniu minorów (5.2.2).

5.2.1. Metoda sprawdzania ograniczoności sieci z wykorzystaniem P-niezmienników

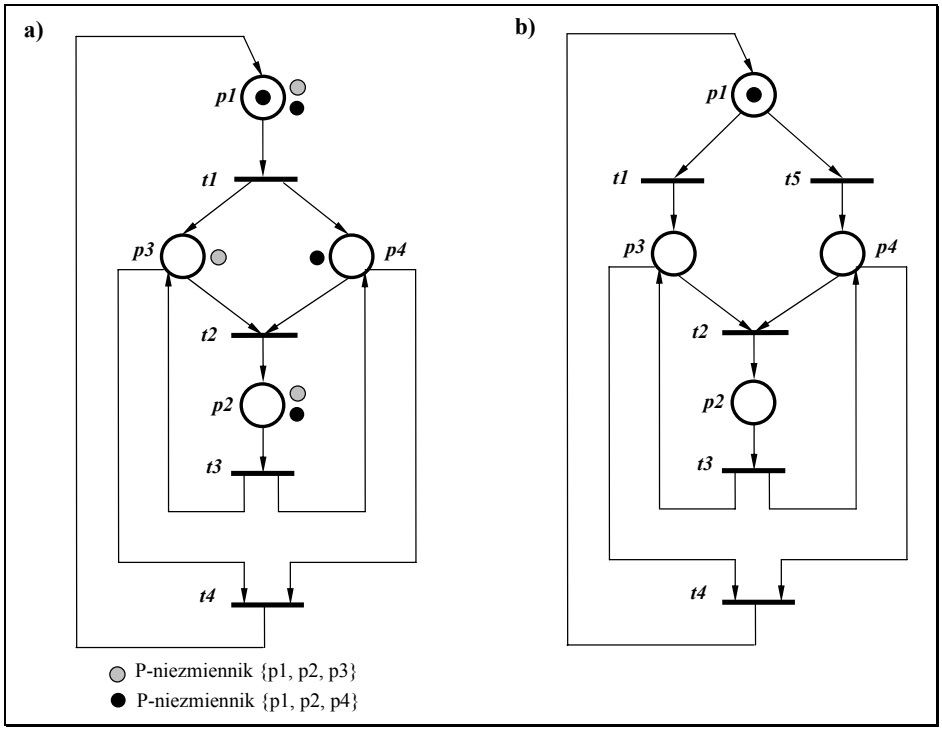
W rozdziale tym zostanie przedstawiona metoda analizy sieci Petriego badająca ograniczoność. Metoda polega na badaniu istnienia pokrycia sieci Petriego P-niezmiennikami [REISIG88].

W celu określenia, czy podana P/T sieć jest ograniczona należy zbadać czy sieć jest pokryta P-niezmiennikami. W przypadku gdy sieć pokryta jest P-niezmiennikami, oznacza to, że jest ona ograniczona [REISIG88]. Związane z tym zagadnieniem definicje znajdują się w rozdziale 2.1.

Poglądowym sposobem wyznaczania P-niezmienników jest porównywanie liczby znaczników wpływających i wypływających z podsieci. Jeżeli liczby te są równe, gwarantowana jest stała liczba znaczników w wyodrębnionej podsieci [REISIG88].

Występowanie poprawnie skonstruowanego P-niezmiennika określonego na danym zbiorze miejsc można także udowodnić formalnie. W tym celu należy wykazać równość, że iloczyn transponowanej macierzy incydencji sieci *C'* i wektora *I* określającego P-niezmiennik, równa się zero (C' * I = 0) [REISIG88].

Dla sieci z Rys. 27a, otrzymano dwa P-niezmienniki *{p1,p2,p3}* oraz *{p1,p2,p4}*. Na Rys. 32 przedstawiono pokrycie rozpatrywanych sieci P-niezmiennikami. Sieć z Rys. 27a została pokryta P-niezmiennikami, oznacza to, że jest ograniczona. Dla sieci z Rys. 27b, nie otrzymano żadnych P-niezmienników.



Rys. 32. Pokrycie sieci P-niezmiennikami

5.2.2. Metoda badania żywotności sieci z wykorzystaniem minorów

Metoda analityczno-macierzowa, opracowana przez Arkadiego Zakrzewskiego polega na sporządzeniu równań logicznych, opisujących blokady i pułapki w sieci Petriego, a następnie ich badaniu z wykorzystaniem macierzy trójwartościowych {0,1,-}[ZAKREVSKIJ85]. W celu wyjaśnienia działania algorytmu wprowadzono dodatkowe definicje i twierdzenia.

Niech zbiór $X=\{x_1, x_2, ..., x_n\}$, będzie zbiorem zmiennych boolowskich. Wartości tych zmiennych reprezentują podzbiór P_j, miejsc sieci P, w ten sposób, że $x_i=1 \Leftrightarrow p_i \in P_j$. Podzbiory zmiennych odpowiadających miejscom wejściowym i wyjściowym tranzycji t_i oznaczone są odpowiednio jako X_i i X'_i.

Twierdzenie 9. Równania opisujące pułapki [ZAKREVSKIJ85]

Wszystkie pułapki w sieci Petriego można zdefiniować przez rozwiązanie równania

$$\bigwedge_{i=1}^{m}(\exists[X_i] \rightarrow \exists[X'_i]) = 1$$

gdzie *m* jest liczbą określającą liczność zbioru tranzycji.

Twierdzenie 10. Równania opisujące blokady [ZAKREVSKIJ85]

Wszystkie blokady w sieci Petriego można zdefiniować przez rozwiązanie równania

$$\bigwedge_{i=1}^{m}(\exists[X'_i] \rightarrow \exists[X_i]) = 1$$

gdzie *m* jest liczbą określającą liczność zbioru tranzycji.

Twierdzenie 11. Minor odpowiadający pułapkom [ZAKREVSKIJ85]
Jeżeli minor kolumnowy macierzy opisującej pułapki, nie zawiera wierszy, w których nie występuje wartość „1”, ale występuje wartość „0”, to wtedy i tylko wtedy minor odpowiada pułapce.

Twierdzenie 12. Minor odpowiadający blokadom [ZAKREVSKIJ85]
Jeżeli minor kolumnowy macierzy opisującej blokady, nie zawiera wierszy, w których nie występuje wartość „1”, ale występuje wartość „0”, to wtedy i tylko wtedy minor odpowiada blokadzie.

Na podstawie sieci Petriego i z wykorzystaniem powyższych definicji, tworzone są równania, opisujące odpowiednio pułapki i blokady. Aby określić, czy sieć Petriego jest żywa, należy rozwiązać obydwa równania. Rozwiązanie można również znaleźć poprzez przekształcenia wyrażeń z twierdzenia 9 i 10, wykorzystując zależności przedstawione na Rys. 33. W celu uproszczenia otrzymanego równania wykorzystywana jest właściwość podana na Rys. 34. W celu określenia, czy badana sieć jest żywa, należy wykorzystać własność Commoner’a.

$$(x \to y)(z \to y) = x \vee z \to y$$
$$x \to y = \bar{x} \vee y$$
$$\overline{x \vee y} = \bar{x}\,\bar{y}$$

Rys. 33. Transformacja równań do formy koniunkcyjnej

$$(x \vee y) \to (x \vee z) \Leftrightarrow y \to x \vee z \Leftrightarrow \bar{y} \vee x \vee z$$

Rys. 34. Eliminacja implikacji

Na podstawie powyższych twierdzeń otrzymano dla sieci z Rys. 27a równania (a) i (b) opisujące odpowiednio blokady i pułapki.

$$(x_1 \to x_3 \vee x_4) \wedge (x_3 \vee x_4 \to x_2) \wedge (x_3 \vee x_4 \to x_1) \wedge (x_2 \to x_3 \vee x_4) = 1 \qquad \text{(a)}$$

$$(x_3 \vee x_4 \to x_1) \wedge (x_2 \to x_3 \vee x_4) \wedge (x_1 \to x_3 \vee x_4) \wedge (x_3 \vee x_4 \to x_2) = 1 \qquad \text{(b)}$$

Otrzymane równanie można zapisać w postać macierzy, w której kolumny odpowiadają argumentom równania, a wiersze implikacjom. Elementy macierzy przybierają wartość: „0”, gdy argument występuje po lewej stronie dysjunkcji, natomiast wartość „1”, gdy argument jest po prawej stronie dysjunkcji. W macierzy występuje wartość „-”, gdy argumentu nie ma w rozpatrywanej dysjunkcji.

Blokadom w sieci z Rys. 27a odpowiada macierz *R* przedstawiona poniżej:

$$R = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{bmatrix} 1 & - & 0 & 0 \\ - & 0 & 1 & 1 \\ - & 1 & 0 & 0 \\ 0 & - & 1 & 1 \end{bmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix}$$

Algorytm wyznaczania minorów określających blokady jest następujący. W pierwszym kroku z macierzy *R*, wybrana jest ta kolumna, która zawiera najmniejszą liczbę zer i największą liczbę jedynek. Zgodnie z twierdzeniem 10, minor opisujący blokady nie może zawierać takich wierszy, w których nie występują wartości „1". W związku z tym, należy kolejno rozpatrywać te wiersze z już wyznaczonej kolumny, które mają wartość „0" i dobierać dla nich takie kolumny, które dla rozpatrywanych wierszy będą miały wartość „1". Krok ten należy wykonywać tak długo, aż każdy wiersz, nie będzie zawierał negatywnych wartości. Na Rys. 35 przedstawiona została ilustracja algorytmu wyznaczania minoru macierzy odpowiadającej blokadom.

$$B1 = \begin{array}{c} \begin{matrix} 4 \end{matrix} \\ \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} \Rightarrow B1 = \begin{array}{c} \begin{matrix} 1 & 4 \end{matrix} \\ \begin{bmatrix} 1 & 0 \\ - & 1 \\ - & 0 \\ 0 & 1 \end{bmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} \Rightarrow B1 = \begin{array}{c} \begin{matrix} 1 & 2 & 4 \end{matrix} \\ \begin{bmatrix} 1 & - & 0 \\ - & 0 & 1 \\ - & 1 & 0 \\ 0 & - & 1 \end{bmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix}$$

Rys. 35. Wyznaczanie minoru macierzy R

Na podstawie macierzy R wyznaczono dwa minory *B1* i *B2*.

$$B1 = \begin{array}{c} \begin{matrix} 1 & 2 & 4 \end{matrix} \\ \begin{bmatrix} 1 & - & 0 \\ - & 0 & 1 \\ - & 1 & 0 \\ 0 & - & 1 \end{bmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} \quad B2 = \begin{array}{c} \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{bmatrix} 1 & - & 0 \\ - & 0 & 1 \\ - & 1 & 0 \\ 0 & - & 1 \end{bmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix}$$

Rozwiązania określone są przez wyznaczone kolumny minorów {1,2,4} i {1,2,3}, które nie zawierają negatywnych wartości wierszy. Minorom *B1* i *B2* odpowiadają więc blokady *{p1, p2, p4}* oraz *{p1, p2, p3}*.

Pułapkom w sieci z Rys. 27a odpowiada macierz *Q* przedstawiona na poniżej:

$$Q = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{bmatrix} 0 & - & 1 & 1 \\ - & 1 & 0 & 0 \\ - & 0 & 1 & 1 \\ 1 & - & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix}$$

Algorytm wyznaczania minorów macierzy odpowiadającej pułapkom, jest analogiczny, jak w przypadku algorytmu wyznaczania minorów dla macierzy

odpowiadającej blokadom. Na podstawie tego algorytmu wyznaczono minory *Q1* i *Q2*, którym odpowiadają pułapki *{p1, p2, p4}* oraz *{p1, p2, p3},* które oznakowane są w znakowaniu początkowym.

$$Q1 = \begin{array}{c} \begin{matrix} 1 & 2 & 4 \end{matrix} \\ \begin{bmatrix} 1 & - & 0 \\ - & 0 & 1 \\ - & 1 & 0 \\ 0 & - & 1 \end{bmatrix} \end{array} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} \qquad Q2 = \begin{array}{c} \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{bmatrix} 1 & - & 0 \\ - & 0 & 1 \\ - & 1 & 0 \\ 0 & - & 1 \end{bmatrix} \end{array} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array}$$

Po zbadaniu zależności pomiędzy otrzymanymi blokadami i pułapkami, z wykorzystaniem własności Commoner'a, można określić, że każda blokada zawiera pułapkę oznakowaną w znakowaniu początkowym, czyli badana sieć jest żywa.

Drugim rozpatrywanym przypadkiem jest sieć z Rys. 27b. Sieci tej odpowiadają równania (c) i (d) opisujące odpowiednio blokady i pułapki.

$$(x_1 \to x_3) \wedge (x_3 \vee x_4 \to x_2) \wedge (x_2 \to x_3 \vee x_4) \wedge (x_3 \vee x_4 \to x_1) \wedge (x_1 \to x_4) = 1 \quad \text{(c)}$$

$$(x_3 \to x_1) \wedge (x_2 \to x_3 \vee x_4) \wedge (x_3 \vee x_4 \to x_2) \wedge (x_1 \to x_3 \vee x_4) \wedge (x_4 \to x_1) = 1 \quad \text{(d)}$$

Na podstawie równania (3) opisującego blokady wyznaczono macierz R:

$$R = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{bmatrix} 1 & - & 0 & - \\ - & 0 & 1 & 1 \\ - & 1 & 0 & 0 \\ 0 & - & 1 & 1 \\ 1 & - & - & 0 \end{bmatrix} \end{array} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}$$

W wyniku działania algorytmu [ZAKREVSKIJ85] otrzymano dwa minory *B1* i *B2* dla blokad. Minorom *B1* i *B2* odpowiadają blokady *{p1, p2, p4}* oraz *{p1, p2, p3}:*

$$B1 = \begin{array}{c} \begin{matrix} 1 & 2 & 4 \end{matrix} \\ \begin{bmatrix} 1 & - & - \\ - & 0 & 1 \\ - & 1 & 0 \\ 0 & - & 1 \\ 1 & - & 0 \end{bmatrix} \end{array} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \qquad B2 = \begin{array}{c} \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{bmatrix} 1 & - & 0 \\ - & 0 & 1 \\ - & 1 & 0 \\ 0 & - & 1 \\ 1 & - & - \end{bmatrix} \end{array} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}$$

Natomiast pułapkom odpowiada macierz *Q*, która jest również minorem *Q1*. Macierzy tej odpowiada pułapka *{p1, p2, p3, p4}*, która jest oznakowana w znakowaniu początkowym *p1*.

$$Q = Q1 = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{bmatrix} 0 & - & 1 & - \\ - & 1 & 0 & 0 \\ - & 0 & 1 & 1 \\ 1 & - & 0 & 0 \\ 0 & - & - & 1 \end{bmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix}$$

Po analizie zależności między wygenerowanymi blokadami a pułapkami, można stwierdzić, że blokady nie zawierają pułapki oznakowanej w znakowaniu początkowym, czyli badana sieć nie jest żywa.

5.3. Metody symboliczne

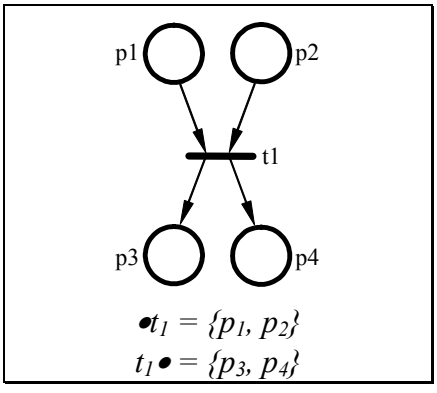
Struktura topologiczna sieci Petriego może być przedstawiona w postaci różnych wyrażeń logicznych. Jednym z możliwych sposobów jest opisanie pewnych, określonych właściwości strukturalnych sieci w postaci formuł Horna, a następnie zastosowanie efektywnych algorytmów obliczeniowych do analizy tych właściwości.

5.3.1. Formuły Horna a blokady w sieciach Petriego

W rozdziale tym zostanie przedstawiony sposób reprezentacji blokad w sieciach Petriego w postaci formuł Horna podobny do opisanego w pracy [MINOUX,BARKAOUI90]. Na podstawie definicji blokady (rozdział 2.3) dla fragmentu sieci podanego na Rys. 36 otrzymywane jest:

jeżeli $p_3 \in S$, to $p_1 \in S$ lub $p_2 \in S$ oraz (1)

jeżeli $p_4 \in S$, to $p_1 \in S$ lub $p_2 \in S$. (2)



Rys. 36. Fragment sieci

Przedstawiając miejsce p_i za pomocą zmiennej logicznej x_i zależności (1) i (2) można zapisać w postaci wyrażeń logicznych:

$$x_3 \rightarrow x_1 + x_2 \quad (3)$$

$$x_4 \rightarrow x_1 + x_2 \quad (4)$$

Stosując własności:

$$a + b \rightarrow c \quad \equiv \quad (a \rightarrow c) \ \wedge \ (b \rightarrow c) \tag{5}$$

otrzymuje się z (3) i (4), analogiczne wyrażenie do wyrażenia z twierdzenia 10 (rozdział 5.2.2) [ZAKREVSKIJ85]:

$$x_3 + x_4 \rightarrow x_1 + x_2 \tag{6}$$

Po lewej stronie znaku implikacji uzyskano sumę wszystkich zmiennych reprezentujących miejsca wyjściowe danej tranzycji t, natomiast po prawej stronie sumę zmiennych reprezentujących wszystkie miejsca wejściowe. Można to zapisać w postaci wyrażenia ogólnego dla danej tranzycji t:

$$\Sigma_{p_i \in t\bullet} x_i \rightarrow \Sigma_{p_j \in \bullet t} x_j \tag{7}$$

Dla całej sieci Petriego wyrażenie obejmujące wszystkie tranzycje sieci przyjmuje postać:

$$\Pi_{t \in T} \ (\Sigma_{p_i \in t\bullet} x_i \rightarrow \Sigma_{p_j \in \bullet t} x_j) \tag{8}$$

Przekształcając wyrażenie (8) i przy zastosowaniu własności:

$$a \rightarrow b \equiv /a + b \tag{9}$$

uzyskuje się eliminację implikacji przez zastąpienie jej negacją poprzednika zsumowanego logicznie z następnikiem:

$$\Pi_{t \in T} \ (\Pi_{p_i \in t\bullet} (/x_i \ + \ \Sigma_{p_j \in \bullet t} x_j)) \tag{10}$$

Wyrażenie (10) nie jest pozytywną formułą Horna, ale może być przekształcane do jej postaci poprzez zastosowanie podstawienia:

$$x_i = /y_i \tag{11}$$

Ostateczna postać wyrażenia dla danej sieci Petriego opisująca blokady:

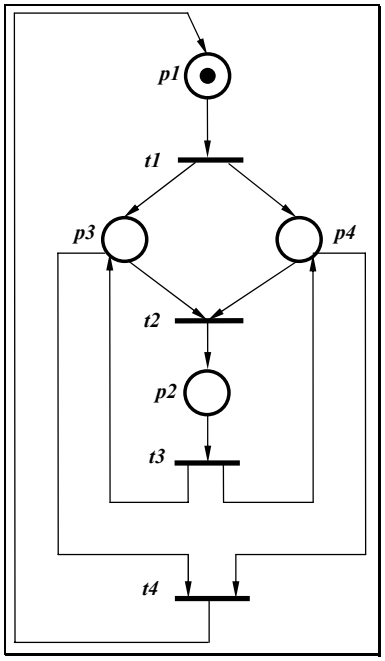
$$\Pi_{t \in T} \ \Pi_{p_i \in t\bullet} (y_i \ + \ \Sigma_{p_j \in \bullet t} /y_j) \tag{12}$$

gdzie:

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

Twierdzenie 13. Blokady w sieci Petriego [MINOUX, BARKAOUI90]

Wszystkie wektory, będące rozwiązaniem równania (12), odpowiadają blokadom w sieci Petriego.



Rys. 37. Przykład sieci Petriego

Rys. 37 przedstawia sieć Petriego, dla której, na podstawie równania (12), wyznaczono formułę Horna *HF* opisującą blokady:

HF = (/p1∨p3) ∧ (/p1∨p4) ∧ (/p3∨/p4∨p2) ∧ (/p2∨p3) ∧ (/p2∨p4) ∧ (/p3∨/p4∨ p1)

Wyznaczanie formuł Horna dla danej sieci Petriego przebiega w następujący sposób [MINOUX,BARKAOUI90]:

a) Wybierane jest miejsce oraz wyszukiwane są dla niego tranzycje wejściowe.
b) Dla każdej z tranzycji wejściowych wypisywana jest oddzielna implikacja zawierająca sumy zanegowanych literałów, odpowiadających miejscom wejściowym tych tranzycji oraz afirmacji literału opisującego rozpatrywane miejsce.

5.3.2. Formuły Horna a pułapki w sieciach Petriego

W podobny sposób, jak opisano tworzenie wyrażenia opisującego blokady, można wyprowadzić wyrażenie opisujące pułapki w sieci Petriego [MINOUX,BARKAOUI90].
Na podstawie definicji pułapki dla fragmentu sieci podanego na Rys. 36 można zapisać:

jeżeli $p_1 \in Q$, to $p_3 \in Q$ lub $p_4 \in Q$ oraz (13)

jeżeli $p_2 \in Q$, to $p_3 \in Q$ lub $p_4 \in Q$. (14)

Przedstawiając miejsce p_i za pomocą zmiennej logicznej x_i zależności (13) i (14) można zapisać w postaci wyrażeń logicznych:

$$x_1 \rightarrow x_3 + x_4 \quad (15)$$

$$x_2 \rightarrow x_3 + x_4 \quad (16)$$

Stosując własności:

$$a + b \rightarrow c \quad \equiv \quad a \rightarrow c \ \wedge \ b \rightarrow c \tag{17}$$

otrzymuje się z (15) i (16), analogiczne wyrażenie do wyrażenia z twierdzenia 9 (rozdział 5.2.2) [ZAKREVSKIJ85]:

$$x_1 + x_2 \rightarrow x_3 + x_4 \tag{18}$$

Po lewej stronie znaku implikacji uzyskano sumę wszystkich zmiennych reprezentujących miejsca wejściowe danej tranzycji t, natomiast po prawej stronie sumę zmiennych reprezentujących wszystkie miejsca wyjściowe. Można to zapisać w postaci wyrażenia ogólnego dla danej tranzycji t:

$$\Sigma_{p_i \in \bullet t} x_i \rightarrow \Sigma_{p_j \in t\bullet} x_j \tag{19}$$

Dla całej sieci Petriego wyrażenie przyjmuje postać:

$$\Pi_{t \in T} \ (\Sigma_{p_i \in \bullet t} x_i \rightarrow \Sigma_{p_j \in t\bullet} x_j) \tag{20}$$

Przekształcając wyrażenie (20) i stosując własności (5) uzyskuje się eliminację implikacji przez zastąpienie jej negacją poprzednika zsumowanego logicznie z następnikiem:

$$\Pi_{t \in T} \ (\Pi_{p_i \in \bullet t} (/x_i \ + \ \Sigma_{p_j \in t\bullet} x_j)) \tag{21}$$

Wyrażenie (21) można przekształcić do postaci formuły Horna stosując podstawienie (11). Ostateczna postać wyrażenia dla danej sieci Petriego opisująca blokady:

$$\Pi_{t \in T} \ \Pi_{p_i \in \bullet t} (y_i \ + \ \Sigma_{p_j \in t\bullet} /y_j) \tag{23}$$

gdzie:

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

Twierdzenie 14. Pułapki w sieci Petriego [MINOUX, BARKAOUI90]

Wszystkie wektory, będące rozwiązaniem równania (23), odpowiadają pułapkom w sieci Petriego.

Dla sieci z Rys. 37, na podstawie równania (23), wyznaczono formułę Horna *HF* opisującą pułapki:

```
HF = (p1∨/p3∨/p4) ∧ (p3∨/p2) ∧ (p4∨/p2) ∧ (p2∨/p3∨/p4) ∧
     (p3∨/p1) ∧ (p4∨/p1)
```

5.3.3. Algorytm SLT

Opisywany algorytm SLT (ang. *Structural Liveness Testing*) sprawdza strukturalną żywotność ograniczonych sieci Petriego z klasy EFC i NSC [BARKAOUI,MINOUX92]. W metodzie tej, badanie żywotności sieci polega na wyszukiwaniu silnie spójnych minimalnych blokad nie będących pułapkami. Algorytm SLT jest kombinacją dwóch metod, metody Tarjana wyszukującej silnie spójne podgrafy grafu skierowanego [TARJAN72] i metody LTUR [DOWLING,GALLIER76].

Algorytm LTUR wykorzystywany jest do wyszukiwania maksymalnej blokady. Natomiast z wykorzystaniem metody Tarjana wyszukiwane są silnie spójne komponenty w zorientowanym grafie.

W prezentowanej metodzie sieć Petriego opisana jest formułą Horna. W metodzie tej sprawdzane są tylko te miejsca, które mają dwie lub więcej tranzycje wyjściowe. Dla każdej pary, takiego miejsca i tranzycji szukana jest silnie spójna blokada, czy zawiera ona sprawdzane miejsce i nie zawiera miejsc wyjściowych sprawdzanej tranzycji. Jeżeli nie zostanie znaleziona taka blokada, wówczas sieć jest strukturalnie żywa. W przeciwnym przypadku sieć nie jest żywa. Złożoność obliczeniowa algorytmu SLT jest wielomianowa względem liczby łuków, miejsc i tranzycji [BARKAOUI,MINOUX92].

5.4. Metody bazujące na redukcji sieci Petriego

Techniki transformacji zamodelowanych układów opisanych sieciami Petriego pozwalają na przedstawienie danego modelu w postaci hierarchicznej, który następnie można weryfikować lub poddać syntezie. Istnieje wiele technik transformacji sieci Petriego. Podstawowe zasady redukcji sieci opisane są w rozdziale 2.4 na przykładach pracy [MURATA89] i [SURAJ,SZPYRKA99]. Zasady te można wykorzystać do sprawdzania żywotności i bezpieczeństwa sieci Petriego opisujących układy sterowania.

W rozdziale tym zostanie przedstawiona metoda redukcji sieć Petriego, zaczerpnięta z [ZAKREVSKIJ85]. Algorytm ten odpowiedni jest zwłaszcza dla α-sieci. Redukcja sieci polega na dwóch prostych operacjach: łączeniu szeregu tranzycji i eliminacji pętli. Sposób wykonania tych operacji opisany został w poniższych definicjach.

Definicja 42. Eliminacja pętli

Tranzycja t_i jest usuwana z sieci, jeżeli zbiór miejsc wejściowych $\bullet t_i$ tej tranzycji równy jest jej zbiorowi miejsc wyjściowych $t_i\bullet$ ($\bullet t_i = t_i\bullet$) i sieć zawiera ponadto inną tranzycję t_j, taką, że zbiór miejsc wejściowych tranzycji t_i zawiera się w zbiorze miejsc wejściowych

tranzycji t_j i zbiór miejsc wyjściowych tranzycji t_i zawiera się w zbiorze miejsc wyjściowych tranzycji t_j: $(\bullet t_i \subseteq \bullet t_j \wedge t_j \bullet \subseteq t_j \bullet)$.

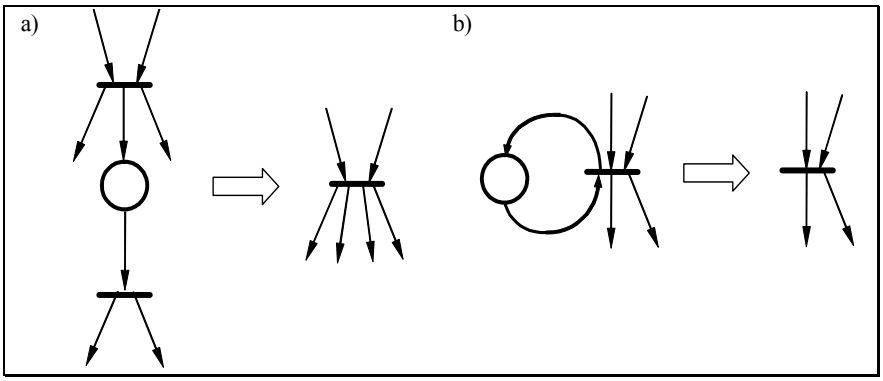
Definicja 43. Łączenie szeregu tranzycji

Niech nieoznakowany zbiór miejsc P (nie zawierający znakowania początkowego) spełnia następujące warunki dla każdej tranzycji t_i:

1. jeżeli $P \cap \bullet t_i \neq 0$ to $P = \bullet t_i$;
2. jeżeli $P \cap t_i \bullet \neq 0$ to $P \subseteq t_i \bullet$;
3. jeżeli $P = \bullet t_i$ i $P \subseteq t_j \bullet$ to $t_j \bullet \cap t_i \bullet = 0$.

Wtedy zbiór miejsc P i wszystkich tranzycji t_i, dla którego $P = \bullet t_i$ jest eliminowany, i każda tranzycja t_j, dla której $P \subseteq t_i \bullet$, jest zamieniana przez zbiór tranzycji otrzymany ze zbioru t_j przez zastąpienie zbioru miejsc P przez zbiór miejsc wyjściowych $t_i \bullet$ otrzymany z eliminacji tranzycji.

Ilustracja powyższych definicji została zamieszczona na Rys. 38a i Rys. 38b.



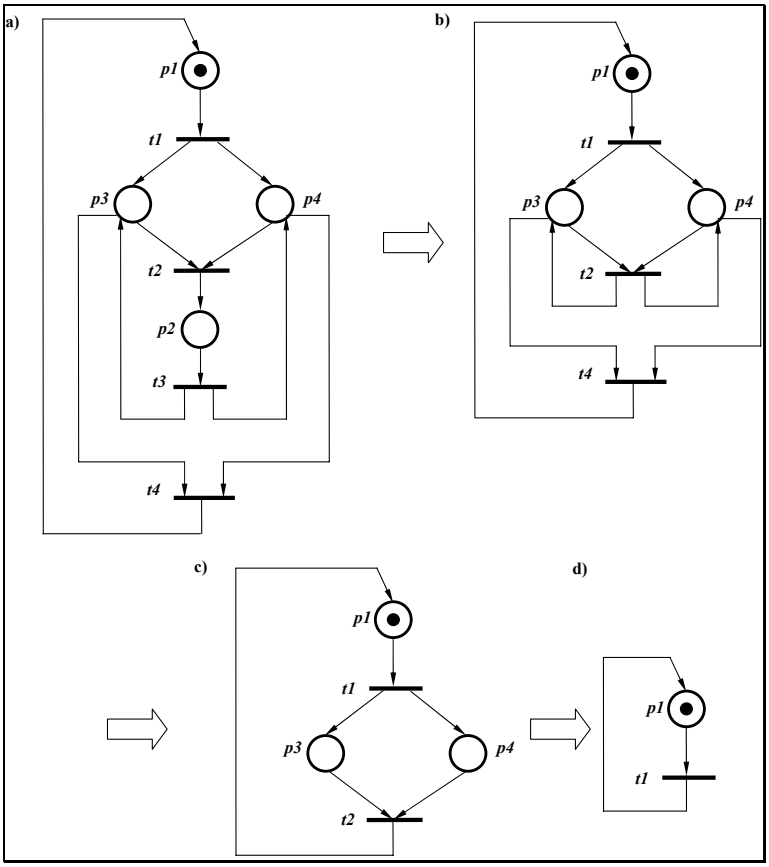
Rys. 38. Redukcja sieci Petriego; a) łączenie szeregu tranzycji b) eliminacja pętli

Jeżeli EFC sieć jest żywa i bezpieczna i posiada tylko jedno znakowanie początkowe, to redukując tą sieć operacjami łączenia szeregu tranzycji i eliminacji pętli otrzymywana jest sieć z jednym miejscem i jedną tranzycją.

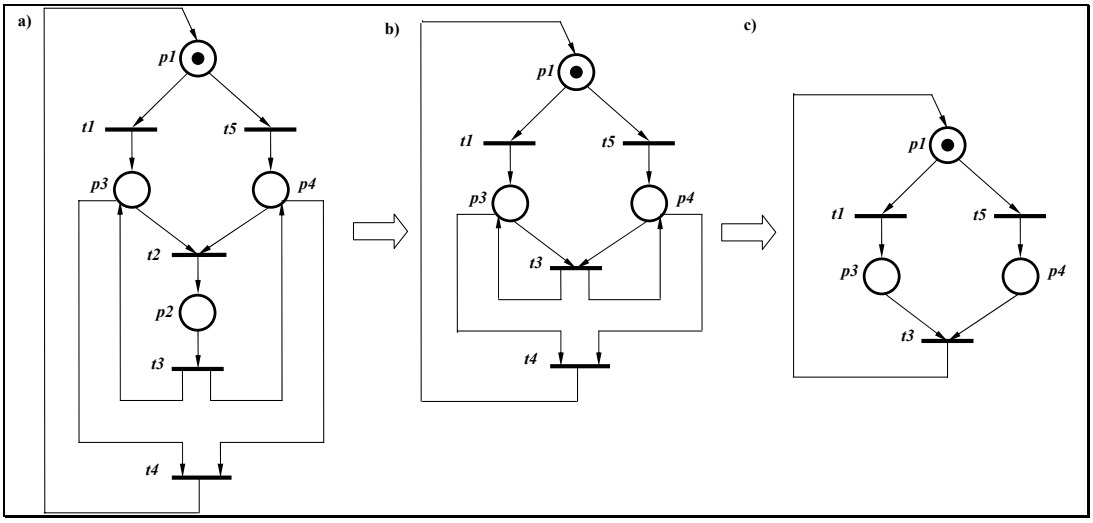
Na Rys. 39a, Rys. 39b, Rys. 39c oraz Rys. 39d zostały przedstawione kolejne etapy redukcji sieci Petriego z Rys. 27a, z wykorzystaniem opisanych metod redukcji. W wyniku otrzymano sieć z jednym miejscem i jedną tranzycją. Na tej podstawie można stwierdzić, że sieć z Rys. 27a jest żywa.

Na Rys. 40a, Rys. 40b oraz Rys. 40c zaprezentowano etap redukcji sieci z Rys. 27b. W wyniku redukcji otrzymano sieć z trzema miejscami i trzema tranzycjami.

Dalsza redukcja tej sieci, z wykorzystaniem opisanych metod, jest niemożliwa. Oznacza to, że badana sieć (Rys. 27b) nie jest żywa.



Rys. 39. Redukcja sieci z Rys. 27a



Rys. 40. Redukcja sieci z Rys. 27b

5.5. Porównanie wybranych metod analizy sieci Petriego

W rozdziale tym zostanie przedstawione porównanie metod analizy sieci Petriego (Tabela 2) opisanych w poprzednich podrozdziałach. Jako kryterium porównujące przyjęto:

- klasę sieci;

- złożoność obliczeniowa;
- rodzaj otrzymywanych wyników.

Metody analizy sieci Petriego można podzielić na cztery grupy: metoda grafu znakowań (osiągalności), metody macierzowe, metody symboliczne oraz techniki redukcji.

Tabela 2. Zestawienie wybranych metod analizy sieci Petriego

L.p.	*Metoda*	*Klasa sieci Petriego*	*Złożoność obliczeniowa*	*Wynik*
1	*Analiza pełnego grafu znakowań*	P/T sieci	wykładnicza	metoda podaje, czy badana sieć jest żywa i ograniczona
2	*Analiza zredukowanego grafu znakowań*	α-sieci	wykładnicza (szybsza od analizy pełnego grafu znakowań)	metoda podaje, czy badana sieć jest żywa i ograniczona
3	*Algorytm SLT*	EFC i NSC	wielomianowa	metoda podaje, czy badana sieć jest żywa
4	*Metoda P-niezmienników*	P/T sieci	wykładnicza	metoda podaje, czy badana sieć jest ograniczona
5	*Metoda bazująca na wyznaczaniu minorów*	P/T sieci	wykładnicza	metoda wyznacza wszystkie blokady i wszystkie pułapki
6	*Metoda bazująca na redukcji sieci Petriego*	P/T sieci	wielomianowa	metoda podaje, czy badana sieć jest żywa i ograniczona

Pierwsza grupa metod polega na wyliczeniu możliwych oznakowań i może być stosowana dla wszystkich rodzajów sieci Petriego (metody 1, 2 w tabeli Tabela 2) [PETERSON81], [MURATA89], [KARATKIEVICH,ET.AL.00]. Główną wadą tych metod jest to, że zastosowana do złożonej sieci Petriego, przedstawiającej dużą liczbę wzajemnie współbieżnych procesów sekwencyjnych, staje się bardzo nieefektywna [PASTOR,ET.AL.94].

Metody bazujące na redukcji (metoda 6) są bardziej efektywne, ale mogą być stosowane tylko do wybranych podklas sieci Petriego [BOUSSIN79], [MURATA89], [SURAJ,SZPYRKA99]. W przytoczonym algorytmie metoda ta jest odpowiednia dla α-sieci [ZAKREVSKIJ85]. W przypadku metody macierzowej wyznaczającej minory (metoda 5) badane mogą być P/T sieci, jednakże złożoność obliczeniowa tej metody jest wykładnicza [ZAKREVSKIJ85]. Efektywnymi metodami są metody symboliczne (metoda 4), jednak stosowane mogą być tylko dla sieci Petriego z klasy EFC i NSC [BARKAOUI,MINOUX92].

Dodatkowo większość metod podaje, czy badana sieć spełnia daną własność, bez dokładnego określenia, w którym miejscu sieci wystąpił defekt.

Podsumowując, metody analizy można podzielić na dwie klasy:

- uniwersalna, ale o małej efektywności;

- dla pewnej klasy podsieci, bardziej efektywna.

Bądź też z drugiej strony, jeżeli metoda podaje dokładne rozwiązania to jej złożoność obliczeniowa jest wykładnicza (metoda 1, 2, 5), jeżeli złożoność algorytmu jest wielomianowa, to otrzymywana jest jedynie odpowiedź, czy sieć spełnia badane własności (metoda 3, 6).

Na podstawie przeglądu wybranych metod analizy sieci Petriego (rozdział 5) stwierdzono, że warto rozwijać badania dotyczące symbolicznych metod analizy sieci Petriego pod kątem weryfikacji modelu współbieżnego automatu. Główną zaletą takiego podejścia jest możliwość zastosowania rozbudowanego aparatu logiki matematycznej wraz z opracowanymi już narzędziami, takimi jak Prolog lub oprogramowanie SAT [BIERE,ET.AL.99], [PENCZEK,ET.AL.02]. Dzięki takiemu podejściu istnieje możliwość weryfikacji opracowanej metody z wykorzystaniem tych narzędzi.

6. Modelowanie sieci Petriego w języku XML

Technologia XML daje możliwość projektowania języka znaczników przedstawicielom różnych dziedzin. Pozwala na wymianę danych i informacji bez konieczności troszczenia się o specjalizowane oprogramowanie. Dokumenty XML można łatwo pisać i odczytywać, więc jest on doskonałym formatem wymiany danych między różnymi aplikacjami. XML jest idealny dla dużych i złożonych dokumentów, gdyż dane w nim są ustrukturyzowane. Zawiera on również mechanizm włączania pozwalający integrować ze sobą dane z różnych źródeł i wyświetlać je jako pojedynczy dokument. W związku z powyżej opisanymi zaletami technologii XML, zdecydowano się na opracowanie nowego formatu opisu sieci Petriego w języku XML.

W rozdziale tym został zaprezentowany język XML, sposób wykorzystania tego języka do opisu sieci Petriego oraz przedstawienie możliwości nowo opracowanego formatu.

6.1. Język XML

Język XML (ang. *eXtensible Markup Language)* jest podzbiorem języka SGML *(*ang. *Standard Generalized Markup Language*) [HAROLD00], [w3.ORG], zdefiniowanego jako ISO 8879. Jest on zestawem reguł definiowania znaczników semantycznych, dzielących dokument na części. Ponadto reguły te jednoznacznie identyfikują poszczególne części.

XML jest językiem metaznaczników, który pozwala definiować składnię semantycznych, strukturalnych języków znacznikowych dla poszczególnych dziedzin zastosowań. Został on stworzony, aby ułatwić wymianę strukturalnych dokumentów poprzez sieć Internet. Wykorzystywany jest on również jako środek, umożliwiający różnorodnym systemom, platformom i aplikacjom, przetwarzanie danych. Pliki XML są nazwane dokumentami, ponieważ dane mogą być przesyłane w różnej formie (pliki, część strony internetowej, strumień tekstu itp.) [HAROLD00], [w3.ORG].

Do najważniejszych zalet języka XML należą [HAROLD00]:

- proste i bezpośrednie użycia języka SGML w sieci Internet;
- łatwe definiowanie typów dokumentów;

- łatwe zarządzanie i tworzenie dokumentów;
- łatwe przesyłanie i dzielenie dokumentów poprzez sieć.

Dokument XML jest dokumentem logicznie zbudowanym w oparciu na jednostki, które są odpowiednikami obiektów w programowaniu obiektowym. Każda jednostka może zawierać jeden lub więcej logicznych elementów (znaczników). Natomiast każdy element może posiadać atrybuty. Elementy mogą być zagnieżdżone, tworząc hierarchię zwaną drzewem informacyjnym.

Struktura dokumentu XML jest zdefiniowana za pomocą pliku lub zbioru plików zawierających definicje typu dokumentów - DTD (ang. *Document Type Definition*). Są one zapisane w składni deklaracji XML. Dokument DTD zawiera formalny opis danego typu dokumentu. Określa również, jakie nazwy mogą być użyte dla elementów typów, ich atrybuty i zależności między nimi [HAROLD00], [TRACZYK01], [W3.ORG].

Formatowanie dokumentu XML może być określone za pomocą arkusza stylu XSL (ang. *Extensible Stylesheet Language*). W arkuszu stylu elementom zdefiniowanym w dokumencie DTD przyporządkowane zostają atrybuty wyświetlania: czcionka, kolor, itp. Ten sam dokument XML można prezentować na różne sposoby w zależności od wyboru arkusza stylu XSL [HAROLD00].

Specyfikacja języka XSL składa się z dwóch części [HAROLD00], [TRACZYK01], [W3.ORG]:

- języka XSLT (ang. *XSL Transformation*) do konwersji dokumentów XML na np. język HTML;
- język XSL FO (ang. *XSL Formatting Objects*) do konwersji dokumentu XML w sformalizowaną postać dokumentu fo:xml. Zawiera on nie tylko dane z dokumentu pierwotnego, ale również formatowanie. Język XSL FO stosowany jest jako format przejściowy lub jako format podglądu pliku XML w specjalizowanych edytorach.

Język XSL umożliwia definicję zbioru reguł formatujących. Opisuje on transformację drzewa wejściowego, zapisanego w języku XML, na drzewo wyjściowe. Transformacja jest dokonywana przy pomocy wzorów i szablonów. Wzór służy do opisania transformacji elementów w drzewie wejściowym. Natomiast szablon jest częścią drzewa wyjściowego. Drzewo wyjściowe jest odseparowane od wejściowego, przy czym ich struktura może być całkowicie różna. Podczas tworzenia drzewa wyjściowego, elementy z drzewa wejściowego mogą być filtrowane, przepisywane oraz może zostać dodana nowa struktura [HAROLD00].

6.2. Język PNSF3

W trakcie badań nad możliwościami modelowania i analizy układów współbieżnych z wykorzystaniem sieci Petriego powstało wiele tekstowych formatów opisu sieci, na przykład Petri Net Specification Format (PNSF) [KOZŁOWSKI,ET.AL.95]. Podobną postać regułową wykorzystano w języku CONPAR [FERNANDES,ET.AL.97] Oba formaty zostały następnie rozszerzone o możliwość opisu również kolorowanych sieci Petriego (PNSF2) [WĘGRZYN98]. Wprowadzenie nowych technologii wymusza modyfikację opracowanych formatów. Z tego względu format PNSF2, zorientowany na formalną specyfikację sieci P/T interpretowanych, kolorowanych i hierarchicznych został zmodyfikowany i zapisany w języku znaczników XML. Powstały format PNSF3 jest również tekstowym formatem zapisu interpretowanej, synchronicznej, hierarchicznej i kolorowanej sieci Petriego [WĘGRZYN,BUBACZ01A], [WĘGRZYN02]. Jest on zbiorem reguł opisujących strukturę sieci Petriego. Reguły mają bezpośrednie odzwierciedlenie w elementach struktury sieci. Z wykorzystaniem formatu PNSF3 opisywany jest algorytm sterowania danego sterownika. Format PNSF3 pozwala na dokładne opisanie kolejnych elementów struktury - miejsc i tranzycji sieci oraz połączeń wewnątrz grafu. Przechowuje także informacje o sygnałach zegarowych, wejściowych i wyjściowych kontrolera i ich rodzaju. Ponadto, pozwala na opisanie warunków realizacji kolejnych tranzycji oraz następstw ich wykonania. Format PNSF3 nie przechowuje jednak informacji o sposobie graficznego rozmieszczenia elementów sieci w obrazie grafu.

Przedstawiony format PNSF3 posiada własny słownik i strukturę znaczników XML, wykorzystywanych do zapisu informacji o sieci. W dokumentacji formatu zdefiniowano szereg ogólnych znaczników grupujących pewne charakterystyczne informacje o modelu. Każdy zdefiniowany znacznik zawiera inne informacje i opisuje inny element grafu. Poszczególne znaczniki posiadają natomiast listę argumentów lub znaczników potomnych, umożliwiających dokładne wyspecyfikowanie informacji o danym elemencie grafu.

Bardzo ważnym zagadnieniem, przy tworzeniu nowego formatu, jest wyznaczenie pewnych zasad i ujednolicenie sposobu zapisu wszystkich tworzonych dokumentów w tym formacie. Wykorzystanie języka XML do zapisu sieci Petriego, znacznie ułatwia proces weryfikacji dokumentu, gdyż istnieje obecnie wiele parserów pozwalających na porównanie zawartości dokumentu z zawartością szablonu wzorcowego. Właśnie, dlatego, w celu poprawnej walidacji tworzonych dokumentów został opracowany zewnętrzny opis DTD definiujący strukturę, składnię i semantykę poszczególnych znaczników. Opis ten, definiuje także kolejność i liczbę powtórzeń znaczników potomnych oraz argumentów

danego znacznika. Każdy poprawnie sformułowany dokument formatu PNSF3, powinien być zgodny z opisem DTD. Ogólny schemat poprawnego dokumentu przedstawia Rys. 41.

Poprawny dokument formatu PNSF3, rozpoczyna się od instrukcji przetwarzania samego języka XML. Instrukcje te, definiują wersję języka, stronę kodową oraz ścieżkę dostępu do szablonu wzorcowego ***pnsf3.dtd.*** Po tych instrukcjach musi wystąpić główny znacznik <**PNSF3**> obejmujący klamrą zawartość całego dokumentu.

```
<?xml version="1.0" encoding="ISO-8859-2" standalone="no"?>
<!DOCTYPE PNSF3 SYSTEM "pnsf3.dtd">
<PNSF3>
      <GLOBAL>
            deklaracja sygnałów globalnych
      </GLOBAL>
      <MACRO_PLACE>
            deklaracja makromiejsca
      </MACRO_PLACE>
      </MACRO_TRANSITION>
            deklaracja makrotranzycji
      </MACRO_TRANSITION>
      <PART>
            deklaracja modułu sterownika
      </PART>
</PNSF3>
```

Rys. 41. Ogólny schemat dokumentu w formacie PNSF3

Jak przedstawiono na Rys. 41 opis sieci Petriego składa się z czterech głównych bloków - znaczników:

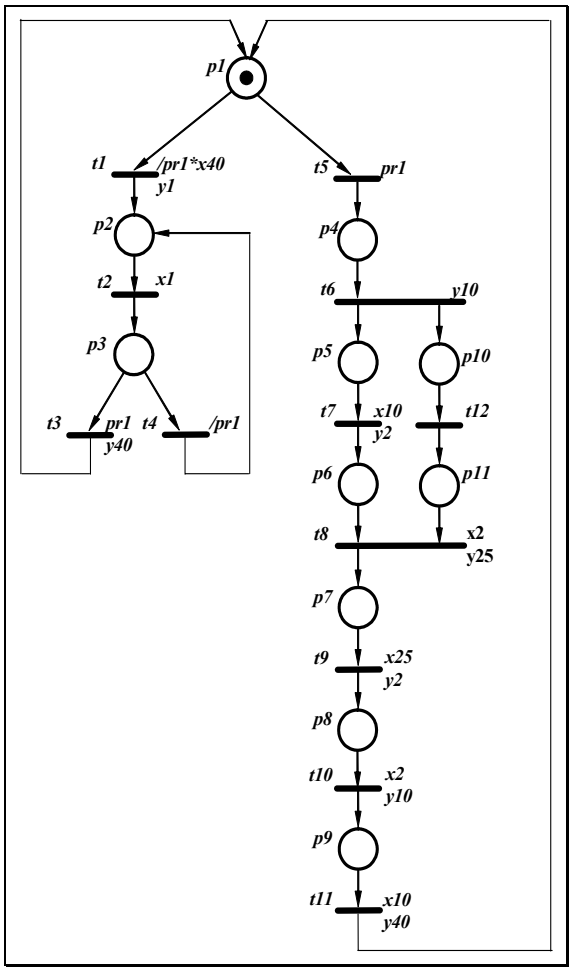
<GLOBAL> - Znacznik może wystąpić w dokumencie tylko jeden raz. Pozwala na zdefiniowanie sygnałów globalnych modelowanego układu, obowiązujących w zakresie całego opisu sieci Petriego.

<PART> - Znacznik musi wystąpić w dokumencie przynajmniej jeden raz. Pozwala na zdefiniowanie pewnego pseudo-niezależnego modułu projektowanego układu. Każdy znacznik **<PART>** może opisywać cały projektowany układ lub tylko jego fragment. Znacznik ten pozwala na zdefiniowanie lokalnych sygnałów modelowanego układu, obowiązujących w zakresie danego modułu oraz pozwala na opis grafu połączeń sieci.

<MACRO_PLACE> - Znacznik wykorzystywany jest do opisu hierarchicznych sieci Petriego. Występuje tylko wtedy gdy w opisie nadrzędnego modułu **<PART>** wystąpi deklaracja makrowęzła. Znacznik ten, stanowi rozwinięcie opisu grafu wewnątrz danego makromiejsca. Znacznik ten pozwala na zdefiniowanie lokalnych sygnałów modelowanego układu, obowiązujących w zakresie danego makrowęzła oraz pozwala na opis grafu połączeń sieci. Wewnętrzne znaczniki potomne specyfikują identyczne parametry jak opisane dla znacznika **<PART>.** Ponadto dodatkowo za pomocą argumentów tego znacznika można zdefiniować: **<MACRO_TRANSITION>.**

Znacznik **<MACRO_TRANSITION>** wykorzystywany jest do opisu hierarchicznych sieci Petriego. Występuje tylko wtedy, gdy w opisie nadrzędnego modułu **<PART>** wystąpi deklaracja makrowęzła. Znacznik ten, stanowi rozwinięcie opisu grafu wewnątrz danej makrotranzycji. Pozwala on na zdefiniowanie lokalnych sygnałów modelowanego układu, obowiązujących w zakresie danego makrowęzła oraz pozwala na opis grafu połączeń sieci. Wewnętrzne znaczniki potomne specyfikują identyczne parametry jak opisane dla znacznika **<PART>.**

Rys. 42 przedstawia sieć Petriego, dla której przygotowano zapis w formacie PNSF3 (Rys. 43). Dokładny opis formatu PNSF3 oraz dokumentu DTD dla tego formatu znajduje się w [WĘGRZYN01A].



Rys. 42. Sieć Petriego

Obecnie na świecie istnieje wiele grup tworzących własne pakiety oprogramowania i narzędzi wspomagających tworzenie i symulację modeli sieci Petriego. Utworzyły one również własne formaty opisu sieci. Proponowany format PNSF3 może być przydatny w wymianie danych pomiędzy różnymi narzędziami. Największymi zaletami nowego

formatu są bez wątpienia: duża elastyczność, kompleksowy opis modelu, niezależność od systemu, łatwość zapisu, przejrzystość oraz łatwość weryfikacji i translacji.

```
<?xml version="1.0" encoding="ISO-8859-2" standalone="no"?>
<?xml-stylesheet type="text/xsl" href="pnsf3_v2.xsl"?>
<!DOCTYPE PNSF3 SYSTEM "pnsf3_v2.dtd">
<PNSF3>
  <CLOCKS>    <CLOCK ID="clk1"> clk1 </CLOCK>   </CLOCKS>
  <INPUTS>
  <INPUTS>    <INPUT ID="x1"> x1   </INPUT>
              <INPUT ID="x2"> x2   </INPUT>
              <INPUT ID="x10"> x10 </INPUT>
              <INPUT ID="x25"> x25 </INPUT>
              <INPUT ID="x40"> x40 </INPUT>
              <INPUT ID="pr1"> pr1 </INPUT>     </INPUTS>
  <OUTPUTS>   <OUTPUT ID="y1"> y1   </OUTPUT>
              <OUTPUT ID="y2"> y2   </OUTPUT>
              <OUTPUT ID="y10"> y10 </OUTPUT>
              <OUTPUT ID="y25"> y25 </OUTPUT>
              <OUTPUT ID="y40 y40   </OUTPUT>  </OUTPUTS>
  <REG_OUTPUTS> <REG_OUTPUT ID_OUTPUT="y1"> y1   </REG_OUTPUT>
                <REG_OUTPUT ID_OUTPUT="y2"> y2   </REG_OUTPUT>
                <REG_OUTPUT ID_OUTPUT="y10"> y10 </REG_OUTPUT>
                <REG_OUTPUT ID_OUTPUT="y25"> y25 </REG_OUTPUT>
                <REG_OUTPUT ID_OUTPUT="y40"> y40 </REG_OUTPUT>  </REG_OUTPUTS>
  <PLACES>       <PLACE ID="p1" MARKING="yes"> p1 </PLACE>
                 <PLACE ID="p2" > p2 </PLACE>
                 <PLACE ID="p3" > p3 </PLACE>
                 <PLACE ID="p4" > p4 </PLACE>
                 <PLACE ID="p5" > p5 </PLACE>
                 <PLACE ID="p6" > p6 </PLACE>
                 <PLACE ID="p7" > p7 </PLACE>
                 <PLACE ID="p8" > p8 </PLACE>
                 <PLACE ID="p9" > p9 </PLACE>
                 <PLACE ID="p10" > p10 </PLACE>
                 <PLACE ID="p11" > p11 </PLACE>    </PLACES>
  <PREDICATES>   <PREDICATE ID="pred1"> /pr1*x40 </PREDICATE>
                 <PREDICATE ID="pred2"> /pr1     </PREDICATE>     </PREDICATES>
  <TRANSITIONS>
      <TRANSITION ID="t1" ID_INPUTS="pred1" ID_OUTPUTS="y1" > t1 </TRANSITION>
      <TRANSITION ID="t2" ID_INPUTS="x1"                    > t2 </TRANSITION>
      <TRANSITION ID="t3" ID_INPUTS="pr1"   ID_OUTPUTS="y4O"> t3 </TRANSITION>
      <TRANSITION ID="t4" ID_INPUTS="pred2"                 > t4 </TRANSITION>
      <TRANSITION ID="t5" ID_INPUTS="pr1"                   > t5 </TRANSITION>
      <TRANSITION ID="t6"                   ID_OUTPUTS="y10"> t6 </TRANSITION>
      <TRANSITION ID="t7" ID_INPUTS="x10"   ID_OUTPUTS="y2" > t7 </TRANSITION>
      <TRANSITION ID="t8" ID_INPUTS="x2"    ID_OUTPUTS="y25"> t8 </TRANSITION>
      <TRANSITION ID="t9" ID_INPUTS="x25"   ID_OUTPUTS="y2" > t9 </TRANSITION>
      <TRANSITION ID="t10" ID_INPUTS="x2"   ID_OUTPUTS="y10"> t10 </TRANSITION>
      <TRANSITION ID="t11" ID_INPUTS="x10"  ID_OUTPUTS="y40"> t11 </TRANSITION>
      <TRANSITION ID="t12"                                  > t12 </TRANSITION>
    </TRANSITIONS>
     <NET>
      <ARC ID_TRANSITION="t1"  IN_ID_PLACES="p1" OUT_ID_PLACES="p2"> </ARC>
      <ARC ID_TRANSITION="t2"  IN_ID_PLACES="p2" OUT_ID_PLACES="p3"> </ARC>
      <ARC ID_TRANSITION="t3"  IN_ID_PLACES="p3" OUT_ID_PLACES="p1"> </ARC>
      <ARC ID_TRANSITION="t4"  IN_ID_PLACES="p3" OUT_ID_PLACES="p2"> </ARC>
      <ARC ID_TRANSITION="t5"  IN_ID_PLACES="p1" OUT_ID_PLACES="p4"> </ARC>
      <ARC ID_TRANSITION="t6"  IN_ID_PLACES="p4" OUT_ID_PLACES="p5 p10"> </ARC>
      <ARC ID_TRANSITION="t7"  IN_ID_PLACES="p5" OUT_ID_PLACES="p6"> </ARC>
      <ARC ID_TRANSITION="t8"  IN_ID_PLACES="p6 p11" OUT_ID_PLACES="p7"> </ARC>
      <ARC ID_TRANSITION="t9"  IN_ID_PLACES="p7" OUT_ID_PLACES="p8"> </ARC>
      <ARC ID_TRANSITION="t10" IN_ID_PLACES="p8" OUT_ID_PLACES="p9"> </ARC>
      <ARC ID_TRANSITION="t11" IN_ID_PLACES="p9" OUT_ID_PLACES="p1"> </ARC>
      <ARC ID_TRANSITION="t12" IN_ID_PLACES="p10" OUT_ID_PLACES="p11"> </ARC>
    </NET>
</PNSF3>
```

Rys. 43. Format PNSF3 dla sieci z Rys. 42

W literaturze znane są już formaty opisu sieci Petriego w formacie XML [LYNGSO,MAILUND98], [WEBER,KINDLER02], jednakże większość z nich opisuje atrybuty graficzne każdego elementu sieci. Przykładem takiego systemu jest Design/CPN opracowany na Uniwersytecie w Aarchus (Dania) [JENSEN92], [JENSEN94], [JENSEN97]. System ten szerzej opisany został w rozdziale 8. W formacie PNSF3 pominięte zostały atrybutów graficzne, co pozwala na ręczne przygotowywanie opisu modeli sieci, przy czym analiza składni takiego formatu jest łatwiejsza. Dodatkowym atutem braku atrybutów graficznych jest możliwość pełnej sprawdzalności formatu poprzez dokument DTD. Formaty przechowujące dane graficzne nie zawierają tego typu dokumentu, gdyż nie ma możliwości zapisu w nim wymagań, dotyczących rozmieszczenia elementów na arkuszu.

Zalety opracowanego języka specyfikacji sieci Petriego są następujące:

- łatwość tworzenia i modyfikacji modeli w tym formacie;
- łatwość analizowania składni poprzez ogólnodostępne narzędzia;
- łatwa konwersja formatu do różnych reprezentacji;
- możliwość wymiany danych pomiędzy różnymi systemami;
- możliwość weryfikacji stworzonego modelu w różnych systemach;
- możliwość symulacji zamodelowanej sieci poprzez wykorzystanie formatu SVG.

6.3. Zastosowania formatu PNSF3 w modelowaniu sterowników logicznych

Wykorzystanie języka XML do opisu sieci Petriego daje wiele nowych możliwości. W rozdziale tym zostaną przedstawione jedynie dwie przykładowe. Upowszechnianie sposobu zapisu informacji w postaci dokumentu XML spowodowało konieczność przystosowania narzędzi do tworzenia systemów informacyjnych z bazami danych, do technologii XML. Atutem zapisu sieci Petriego w formacie PNSF3 jest łatwe przekształcenie informacji zgromadzonej w bazie danych w postaci relacyjnej na język XML i odwrotnie [WĘGRZYN,BUBACZ01], [WĘGRZYN,BUBACZ01A].

Jak już wcześniej wspomniano, zastosowanie języka XML pozwala również na łatwą konwersję formatu opartego na tym języku na różne reprezentacje, np. na format grafiki wektorowej SVG. Dzięki takiemu podejściu, format tekstowy opisu sieci można transformować do formatu graficznego, a następnie sieć poddać symulacji.

6.3.1. Modelowanie sieci Petriego z wykorzystaniem narzędzi do obsługi baz danych

W systemach z bazami danych język XML zastosowany jest do reprezentacji danych poza bazą oraz do przechowywania złożonych struktur w bazie danych. Pierwszym krokiem producentów systemów zarządzania bazami danych było skonstruowanie interfejsów umożliwiających przekształcenie informacji zgromadzonej w bazie danych w postaci relacyjnej na język XML i odwrotnie. Jednakże w przypadku, gdy język XML wykorzystywany jest do reprezentacji złożonej informacji, trudno jest taką strukturę przekształcić do postaci relacyjnej.

Istnieją trzy sposoby przechowywania dokumentów XML w bazie danych [BOURRET02]:

- zapis w pełni tekstowy,
- częściowa strukturalizacja,
- reprezentacja generyczna.

W przypadku zapisu czysto tekstowego w bazie danych, struktura danych jest bardzo elastyczna, ale nieczytelna i nieefektywna. Zapis danych w bazie w postaci częściowej strukturalizacji pozwala na gromadzenie danych z dokumentów złożonych i o niejednolitej strukturze. Stałe elementy struktury XML są rozkładane do postaci relacyjnej, natomiast elementy o zmiennej strukturze są przechowywane jako teksty ze znacznikami. Ten sposób reprezentacji danych daje duże możliwości przetwarzania, lecz jest on nieelastyczny. Natomiast reprezentacja generyczna wykorzystywana jest w przypadku dokumentów o jednolitej strukturze, jednakże w przypadku złożonych dokumentów jest ona nieefektywna oraz nieczytelna.

Format PNSF3 należy do reprezentacji niezłożonych informacji, które w łatwy sposób przekształcane są do postaci relacyjnej. Zdecydowano się na częściową strukturalizację danych, tzn. część znaczników z dokumentu reprezentowana jest w postaci kolumny w tabeli, natomiast bardziej rozbudowane i opisowe elementy, w postaci tekstów ze znacznikami.

6.3.2. Symulacja sieci Petriego z wykorzystaniem technik graficznych

Grafika wektorowa opiera się na matematycznym opisie takich elementów, jak: koło, wielokąt, linia. Format SVG (ang. *Scalable Vector Graphics*) opisuje dwuwymiarową grafikę wektorową [W3.ORG]. Zaletą tego formatu jest to, że zapisywany jest on w postaci tekstowej i zbudowany na bazie języka XML. Elementy graficzne SVG, w łatwy sposób są modyfikowane i animowane za pomocą zróżnicowanych technik

[W3.ORG]. Najważniejsza korzyść, jaką daje technologia SVG, wynika z jej powiązania z technologią XML, czyli bezpośrednie generowanie kodu, na podstawie informacji pobieranych np. z bazy danych.

Obrazki w formacie SVG są w pełni skalowalne, mają niewielkie rozmiary. Ich elementy mogą być w łatwy sposób modyfikowane, ponieważ wszystkie informacje - zarówno o sposobie wyświetlania, jak i obsłudze związanych z nimi zdarzeń, zawarte są w językach skryptowych. Informacje te umieszczane są, identycznie jak w przypadku języka HTML, w obrębie dokumentu lub w zewnętrznym pliku. W stylach definiować można wszystkie charakterystyczne dla grafiki wektorowej właściwości takie, jak: kolor i rodzaj wypełnienia, kolor i grubość obrysu lub poziom przezroczystości, z warstw zaś - tworzyć osobne klasy. Reguły stosowania stylów są takie same, jak w przypadku języka HTML i języka XML - style mogą być definiowane w nagłówku pliku, w obrębie pojedynczego znacznika lub w zewnętrznym zbiorze.

Zapis sieci Petriego w formacie XML pozwala na łatwą konwersję na format SVG. Natomiast zapis sieci w formacie SVG pozwala na jej weryfikację poprzez symulację. Symulacja sieci jest najprostszą metodą weryfikacji. Daje ona możliwość obserwacji kolejnych stanów układu. Podczas symulacji sprawdzane jest, czy system opisany interpretowaną siecią Petriego zachowuje się prawidłowo. Jeżeli występują błędy, wówczas już we wczesnym stadium projektowania są one rozpoznawane i usuwane. Metoda ta jest nieefektywna dla dużych modeli, gdyż częściowa symulacja układu może wykryć obecność błędów, ale nigdy nie może wykazać ich braku. Pełna symulacja układu cyfrowego jest możliwa tylko dla najprostszych układów. Jednakże w takim przypadku można zastosować hierarchię sieci, i makrosieć oraz każdą z podsieci składowych analizować oddzielnie.

Dynamiczny plik SVG zawiera postać graficzną modelu sieci Petriego, opisanego w sposób tekstowy formatem PNSF3. Plik ten, przy wykorzystaniu standardowej przeglądarki WWW, umożliwia symulację działania projektowanego układu. Generowany dynamiczny plik SVG, zawiera informacje o wszystkich elementach sieci takich, jak: miejsca, tranzycje, znaczniki oraz o sposobie połączeń wewnątrz grafu. Ponadto, zawiera także dodatkowe informacje o rozmieszczeniu tych elementów na ekranie oraz sposobie animacji. Oprócz tego, dodany został blok paneli kontrolnych umożliwiających animacje, zadawanie sygnałów wejściowych, obserwacje stanu wyjść układu i sposobu pokolorowania grafu.

Dynamiczny plik SVG składa się z trzech części:

1. **Definicja grafu sieci**

Blok definicji grafu sieci generowany jest na podstawie informacji zawartych w opisie PNSF3 w znacznikach **<PLACES>** i **<TRANSITIONS>.** Każdemu miejscu, tranzycji i znacznikowi zostaje przypisany odpowiedni symbol graficzny i położenie na ekranie. Ponadto, na podstawie danych w znaczniku **<NET>** zostaje wyrysowana sieć połączeń grafu.

2. **Definicja paneli kontrolnych**

Blok definicji paneli kontrolnych generowany jest na podstawie informacji zawartych w opisie PNSF3 w znacznikach **<INPUTS>**, **<OUTPUTS>**, **<COLOURS>**, **<PART>**. Blok paneli kontrolnych składa się z kilku paneli umożliwiających: animację sieci, przełączanie pomiędzy kolejnymi modułami sterownika PLC, wymuszanie sygnałów wejściowych, obserwację stanu wyjść i sposobu pokolorowania grafu. Ponadto, zawiera kopie opisów sieci w formacie PNSF3, do wglądu dla użytkownika.

3. **Skrypt animujący**

Skrypt animujący umożliwia animację i symulację modelu sieci Petriego oraz umożliwia interaktywną komunikację programu z użytkownikiem. Skrypt ten stanowi zbiór funkcji, o składni podobnej np. do języka C++ lub JAVA. Ten fragment pliku SVG generowany jest głównie na podstawie informacji zawartych w opisie PNSF3, w znacznikach **<NET>**.

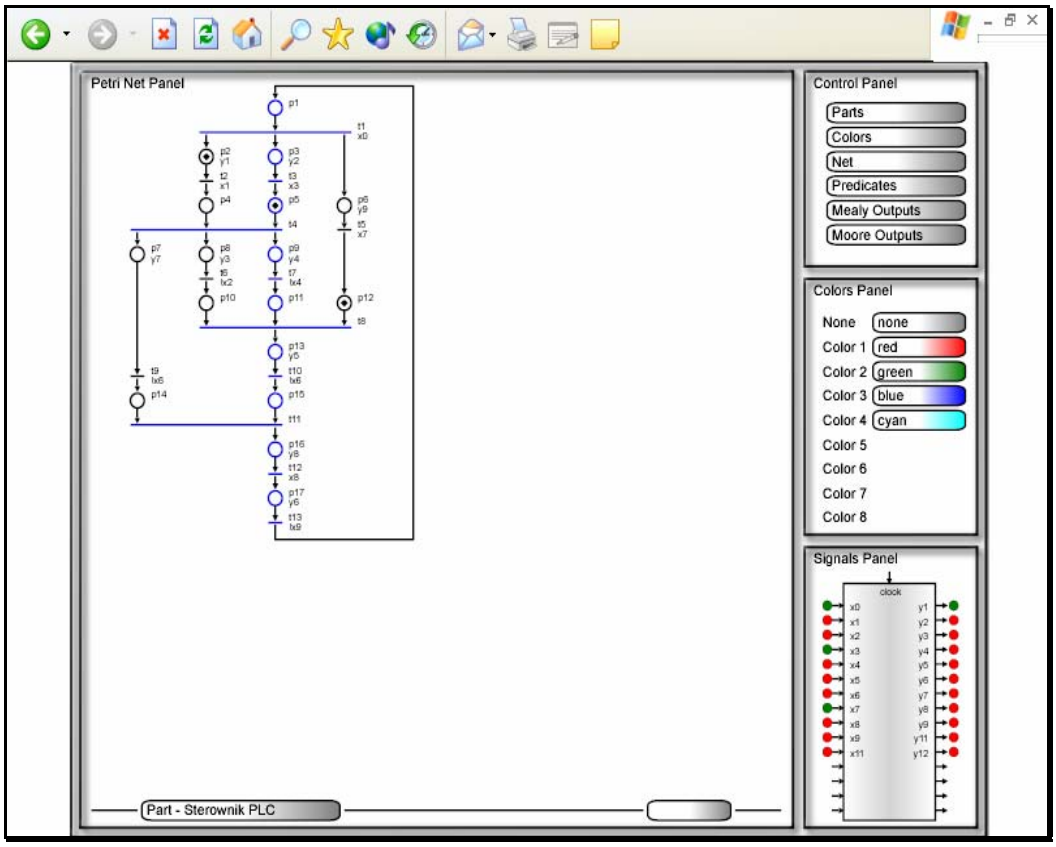
Przykładowy, wygenerowany w wyniku translacji plik SVG, pokazano na Rys. 44 przy użyciu standardowej przeglądarki stron WWW – Internet Explorer 6.0. Rysunek ten przedstawia model formalny sieci Petriego w trakcie procesu symulacji dla układu sterownika.

6.4. PNSF3 a formuły Horna

Format PNSF3 pozwala na szybkie i łatwe wygenerowanie regułowego opisu topologicznej struktury sieci Petriego, w postaci formuł Horna. Z drugiej strony zachowuje odpowiedniość pomiędzy uzyskanymi w drodze analizy podsieciami i modelu w języku XML. Ważną zaletą tego formatu jest możliwość zachowania interpretacji sieci w każdej chwili.

Do analizy modelu automatu współbieżnego zapisanego w postaci formatu PNSF3, wykorzystywany jest jedynie ten fragment modelu, który opisuje poszczególne tranzycje oraz zbiory miejsc wejściowych i wyjściowych dla poszczególnych tranzycji (znacznik <NET>). Rys. 45 przedstawia fragment specyfikacji sieci w PNSF3 (Rys. 43) opisujący

znacznik <NET>. Takie podejście wynika z faktu, że analizowana jest struktura sieci, bez interpretacji. Nie ma również znaczenia, w którym miejscu sieci jest znakowanie początkowe.



Rys. 44. Przykład sieci Petriego w formacie SVG

Sieć Petriego zapisywana jest w postaci równań, formuł Horna. Algorytm wyznaczania formuł Horna opisany został w rozdziale 7.2. Z algorytmu tego wynika, że inne elementy sieci nie są brane pod uwagę.

```
<NET>
 <ARC ID_TRANSITION="t1"  IN_ID_PLACES="p1" OUT_ID_PLACES="p2"> </ARC>
 <ARC ID_TRANSITION="t2"  IN_ID_PLACES="p2" OUT_ID_PLACES="p3"> </ARC>
 <ARC ID_TRANSITION="t3"  IN_ID_PLACES="p3" OUT_ID_PLACES="p1"> </ARC>
 <ARC ID_TRANSITION="t4"  IN_ID_PLACES="p3" OUT_ID_PLACES="p2"> </ARC>
 <ARC ID_TRANSITION="t5"  IN_ID_PLACES="p1" OUT_ID_PLACES="p4"> </ARC>
 <ARC ID_TRANSITION="t6"  IN_ID_PLACES="p4" OUT_ID_PLACES="p5 p10"> </ARC>
 <ARC ID_TRANSITION="t7"  IN_ID_PLACES="p5" OUT_ID_PLACES="p6"> </ARC>
 <ARC ID_TRANSITION="t8"  IN_ID_PLACES="p6 p11" OUT_ID_PLACES="p7"> </ARC>
 <ARC ID_TRANSITION="t9"  IN_ID_PLACES="p7" OUT_ID_PLACES="p8"> </ARC>
 <ARC ID_TRANSITION="t10" IN_ID_PLACES="p8" OUT_ID_PLACES="p9"> </ARC>
 <ARC ID_TRANSITION="t11" IN_ID_PLACES="p9" OUT_ID_PLACES="p1"> </ARC>
 <ARC ID_TRANSITION="t12" IN_ID_PLACES="p10" OUT_ID_PLACES="p11"> </ARC>
</NET>
```

Rys. 45. Fragment formatu PNSF3

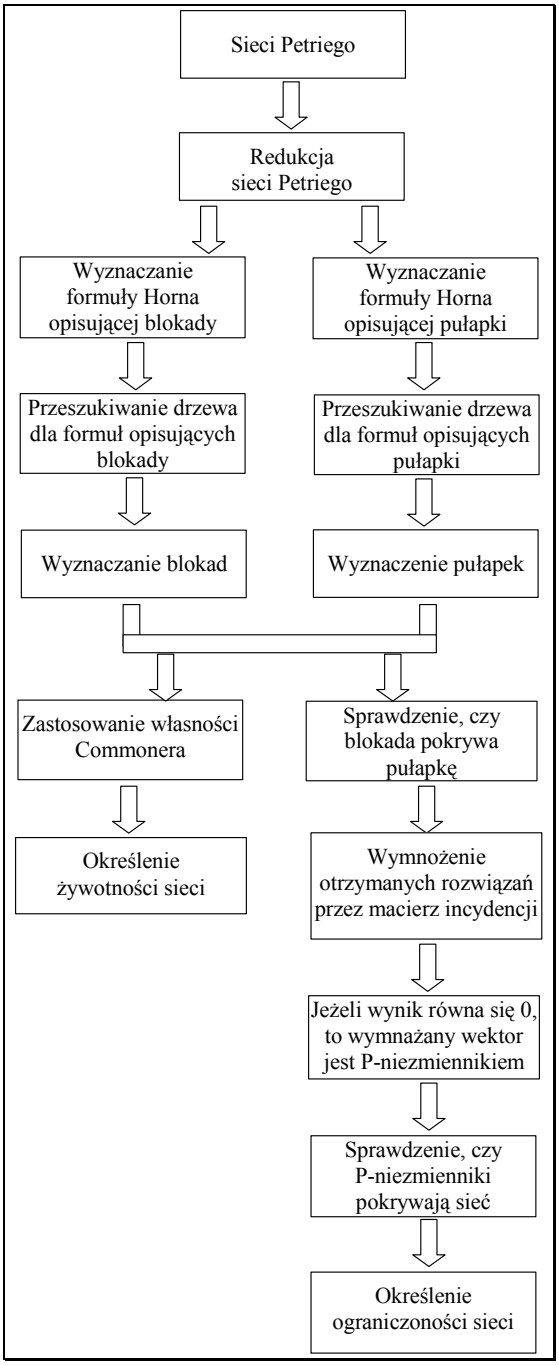
7. Metoda symbolicznej analizy hierarchicznej sieci Petriego

Wyznaczanie wszystkich blokad i pułapek w sieci opisującej układ sterowania, pozwala na zlokalizowanie tych fragmentów sieci, w których występują defekty z punktu widzenia poprawnego przepływu sterowania w sterownikach logicznych.

Metoda opisana w tym rozdziale bazuje na algebrze Boole'a i symbolicznym przetwarzaniu danych. Z wykorzystaniem tej metody można sprawdzić strukturalne własności sieci Petriego. Większość metod analizuje modele płaskie sieci. Natomiast usunięcie tych fragmentów sieci (poprzez zastąpienie ich makromiejscem, bądź makrotranzycją), które nie wpływają na proces analizy, może w znacznym stopniu przyspieszyć ten proces, dlatego zastosowano redukcję sieci.

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 [GIRAULT,VALK03]. W celu otrzymania rozwiązań, należy otrzymane 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. Zaletą tego algorytmu jest liniowa zajętość pamięci. 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. Opracowany algorytm polega na wyznaczaniu wszystkich blokad i pułapek dla podanej sieci Petriego. Ponieważ ich liczba może być bardzo duża, pierwszym zalecanym etapem algorytmu jest przygotowanie zredukowanej sieci Petriego (rozdział 7.1), czyli usunięcie tych fragmentów sieci, które nie mają wpływu na jej żywotność. Następnie sieć Petriego przedstawiana jest w postaci dwóch formuł Horna (jedna opisująca blokady oraz druga opisująca pułapki) (rozdział 7.2). Kolejnym krokiem w proponowanym algorytmie jest

przygotowanie drzewa Thelena wyznaczającego wszystkie rozwiązania dla podanej formuły Horna (rozdział 7.3). Na Rys. 46 przedstawiono schemat blokowy algorytmu symbolicznej metody badania sieci Petriego.



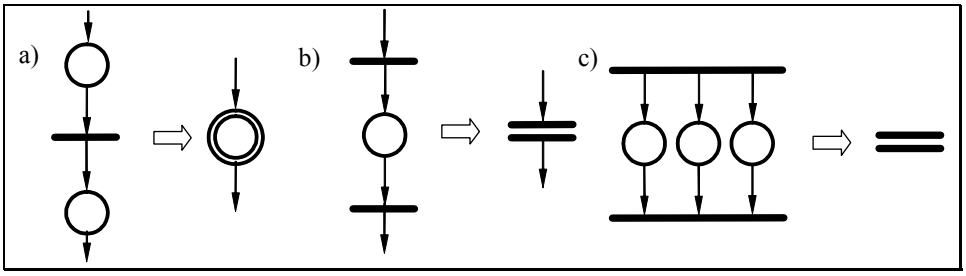
Rys. 46. Schemat blokowy symbolicznej metody badania sieci Petriego

W celu sprawdzenia żywotności i ograniczoności sieci, należy zbadać relacje pomiędzy blokadami i pułapkami w zależności od badanej klasy sieci, co zostało szerzej opisane w rozdziale 7.4 i 7.5. Przykłady analizy sieci zostaną zamieszczone w odrębnych podrozdziałach, w celu zwiększenia czytelności opisywanej metody oraz analizowanych przykładów.

7.1. Redukcja sieci Petriego

Redukcja sieci ma na celu usunięcie z sieci tych fragmentów, które podczas procesu analizy nie wpłyną na jej ostateczny wynik, a ich obecność jedynie zwiększa czas procesu analizy. Wraz z upraszczaniem struktury sieci nadal zachowane są jej własności takie, jak: żywotność i ograniczoność [MURATA89], [SURAJ,SZPYRKA99]. Istnieje wiele sposobów redukcji sieci (opisane zostały w rozdziale 2.4, oraz w [MURATA89], [SURAJ,SZPYRKA99]), jednakże w tym algorytmie zostaną wykorzystane tylko trzy z nich.

Zaproponowany algorytm redukcji sieci polega na tworzeniu makromodułów. Na początku sprawdzane jest występowanie podsieci takich, jak na Rys. 47a i b oraz ich zamiana odpowiednio na makromiejsca (Rys. 48) i makrotranzycje (Rys. 49). Następnie wyszukiwane są takie miejsca lub makromiejsca równoległe, które mają wspólną tranzycję wejściową i wyjściową. Każda z gałęzi równoległych może być podsiecią, tak jak to pokazano na Rys. 47c (Rys. 49). Gałęzie te mogą być zastąpione przez makrotranzycje. Powyższe kroki wykonywane są tak długo, aż wszystkie takie podsieci zostaną zastąpione przez makromoduły [WĘGRZYN,WĘGRZYN00].



Rys. 47. Metody redukcji sieci

```
function makromiejsce(siećPetriego);
  begin
   for t in T do
   begin
     if  |•t|=1 and |t•|=1  then
     begin
          p' := t•;
          p'':= •t;
          P := P ∪ {pm};
          •pm := •p';
          pm• := p''•;
          P := P \ {p', p''};
          T := T \ {t};
     end;
  end;
end; {function makromiejsce}
```

Rys. 48. Algorytm wyznaczania makromiejsc

```
function makrotranzycja(siećPetriego);
  begin
   for p in P do
   begin
     if  |((•p)•)•|=1 and |•((p•)•)|=1  then
     begin
          t' := p•;
          t'':= •p;
          T := T ∪ {tm};
          •tm := •t';
          tm• := t''•;
          P := P \ ((•p)•);
          T := T \ {t', t''}
     end;
  end;
end; {function makrotranzycja}
```

Rys. 49. Algorytm wyznaczania makrotranzycji

Opis symboli użytych w opisie algorytmów wyznaczania makrotranzycji i makromiejsca podano w tabeli (Tabela 3).

Tabela 3. Opis symboli użytych w opisie algorytmu (Rys. 48, Rys. 49)

Symbol	*Opis*
p	miejsce sieci
P	zbiór miejsc sieci
t	tranzycja sieci
T	zbiór tranzycji sieci
•t	zbiór miejsc wejściowych do danej tranzycji t
t•	zbiór miejsc wyjściowych z danej tranzycji t
•p	zbiór tranzycji wejściowych do danego miejsca p
p•	zbiór tranzycji wyjściowych z danego miejsca p
pm	makromiejsce
tm	makrotranzycja

7.2. Wyznaczanie formuł Horna opisujących blokady i pułapki

Kolejnym etapem algorytmu jest przedstawienie struktury sieci Petriego w postaci odpowiadającym jej formułom Horna. Metoda wynika z równań podanych w rozdziale 5.3.1 i 5.3.2 zaproponowanych przez [BARKAOUI,MINOUX92]. Jednakże w tym przypadku algorytm wyznaczania równań zorientowany jest na miejsca, odwrotnie niż w podejściu Autorki, gdzie algorytm zorientowany jest na tranzycje, co wynika z faktu wykorzystywania opisu logicznego sieci Petriego zorientowanego na tranzycje.

W opracowanej metodzie, podczas tworzenia poszczególnych klauzul występujących w formule Horna, przeszukiwane są kolejno wszystkie tranzycje sieci oraz

określone ich miejsca wejściowe i wyjściowe. Informację dotyczącą miejsc wejściowych i wyjściowych tranzycji można uzyskać wprost z reprezentacji macierzowej sieci (rozdział 2.1).

W pierwszym kroku sprawdzane są wszystkie miejsca wyjściowe każdej tranzycji. Miejscom tym odpowiada literał pozytywny w klauzuli Horna. Z definicji formuł Horna wynika, że w danej klauzuli nie może występować więcej niż jeden pozytywny literał. Dlatego też liczba miejsc wyjściowych dla rozpatrywanej tranzycji odpowiada liczbie klauzul wyznaczonych dla tej tranzycji. Następnie wyszukiwane są wszystkie miejsca wejściowe dla tej tranzycji, miejscom tym odpowiada literał negatywny w klauzuli Horna. W ten sposób przeszukiwane są wszystkie tranzycje w sieci Petriego. Opisany algorytm wyznacza formułę Horna opisującą blokady w sieci Petriego (Rys. 50).

```
function formułaHorna(siećPetriego);
  begin
     FH := "";
     for  t  in  T  do
        begin
           K := "";
           for  p  in  t•  do
             begin
                K := p;
                for  q  in  •t  do
                     K := K + not(q);
                FH := FH * (K);
             end;
        end;
     formułaHorna := FH;
end; {function formułaHorna}
```

Rys. 50. Algorytm wyznaczania formuły Horna opisującej blokady

Opis symboli użytych w opisie algorytmu wyznaczania formuły Horna zamieszczono w tabeli (Tabela 4).

Tabela 4. Opis symboli użytych w opisie algorytmu (Rys. 50)

Symbol	*Opis*
FH	formuła Horna
K	klauzula
p	jedno z miejsc wyjściowych z tranzycji *t*
t	tranzycja sieci
T	zbiór tranzycji sieci
•t	zbiór miejsc wejściowych do danej tranzycji *t*
t•	zbiór miejsc wyjściowych z danej tranzycji *t*
q	jedno z miejsc wejściowych do tranzycji *t*

Algorytm wyznaczania formuł Horna opisujących pułapki jest analogiczny, jednakże miejscom wejściowym odpowiada literał pozytywny w klauzuli Horna, a miejscom wyjściowym literał negatywny.

7.3. Metoda wyznaczania rozwiązań

Kolejnym krokiem rozpatrywanej metody analizy jest wyznaczenie wszystkich rozwiązań dla równań opisujących odpowiednio blokady albo pułapki. Aby otrzymać wszystkie rozwiązania określające blokady i pułapki, należy wcześniej otrzymane formuły wymnożyć. Szukane są rozwiązania, dla których formuła Horna równa jest **1**. W tym celu zastosowana jest symboliczna metoda wyznaczania wszystkich rozwiązań, tzn. wszystkich implikantów prostych, bazująca na metodzie Thelena [THELEN88], [MATHONY89]. Metoda ta wykorzystuje metodę przekształcenia funkcji logicznej *f* przedstawionej w postaci koniunkcyjnej, w postać dysjunkcyjną przez algebraiczne wymnażanie poszczególnych członów z uwzględnieniem tożsamości boolowskich przedstawionych poniżej:

```
a * a       = a
a + a * b   = a
a * /a      = 0
a * 0       = 0
a + 0       = a
```

Jednakże zamiast czasochłonnego wymnażania kolejno wszystkich wyrażeń, przeszukiwane jest drzewo, którego konstrukcję, dla funkcji zadanej koniunkcją, można przedstawić następująco:

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

Przeszukiwanie drzewa odbywa się zgodnie z algorytmem przeszukiwania grafu wgłąb (ang. *depth first search*, DFS). Jest ono obcinane według czterech reguł [THELEN88], [MATHONY89]:

R1: łuk jest obcinany, jeżeli iloczyn literałów dla poprzedniego wierzchołka zawiera uzupełnienie (negację) literału przypisanego do danego łuku;

R2: łuk jest obcinany, jeżeli istnieje na wyższym poziomie nie przeszukany jeszcze łuk opisany tym samym literałem;

R3: jeżeli wśród łuków wychodzących z danego wierzchołka istnieje łuk (łuki) opisany literałem występującym w iloczynie opisującym dany wierzchołek, to obcinane są wszystkie, za wyjątkiem jednego, opisanego tym powtarzającym się literałem;

R4: łuk *j* jest obcinany, jeżeli rozwinięta jest już gałąź z łukiem *k* na wyższym poziomie, opisana takim samym literałem, oraz jeżeli reguła *R2* nie została zastosowana w poddrzewie od łuku *k* w odniesieniu do łuku na tym samym poziomie co łuk *k*, który prowadzi do łuku *j*.

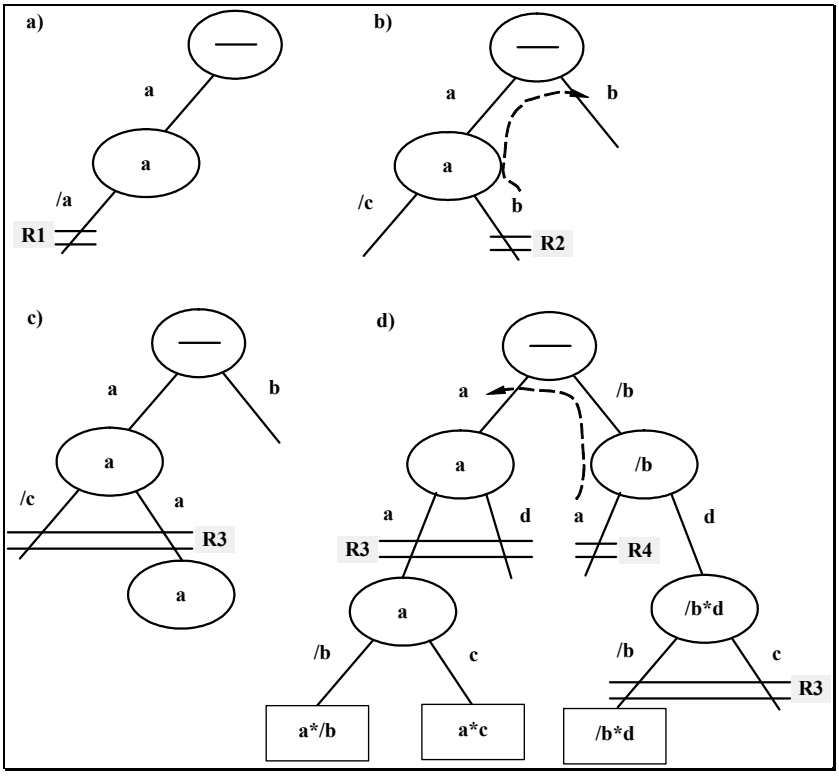
UWAGA:

Łuki obcięte nie są dalej przeszukiwane.

Początkowo, algorytm przeszukiwania drzewa wykorzystywał jedynie trzy pierwsze reguły. W celu wyznaczania tylko implikantów prostych, należało znalezione implikanty porównywać z wcześniej wygenerowanymi, a pochłaniane implikanty eliminować. Reguła R2 gwarantuje, że implikanty mogą być pochłaniane tylko przez już wcześniej wyznaczone, gdyż pierwszy otrzymany implikant jest zawsze implikantem prostym [THELEN88]. Reguła R4, która została dodana przez Mathony'ego powoduje, że nie jest już konieczne dalsze sprawdzanie wygenerowanych już implikantów, gdyż zapewnia ona, że zawsze zostaną wyznaczone implikanty proste [MATHONY89].

Rys. 51 przedstawia ilustrację reguł, odpowiednio a) reguła R1, b) reguła R2, c) reguła R3 oraz d) reguła R4.

Kolejność stosowania reguł nie wpływa na poprawność generowania implikantów prostych. Jednakże badania wykazały, że odpowiednie uszeregowanie wykonywania reguł może przyspieszyć działanie algorytmu. Zalecana kolejność reguł to: R3, R1, R2, R4.



Rys. 51. Ilustracja reguł R1-R4

Zaletą algorytmu opracowanego przez Thelena jest liniowa zależność zajmowanej pamięci, która zależy od liczby zmiennych *n*. W pracy [THELEN88] udowodniono, że maksymalny rozmiar pamięci jest proporcjonalny do *(2n-1)*.

Jeżeli dla formuły Horna opisującej blokady (pułapki) zostały znalezione rozwiązania, wówczas sieć Petriego opisana taką formułą zawiera blokady (pułapki). Jeżeli nie otrzymano żadnego rozwiązania, wówczas sieć Petriego opisana takimi formułami nie zawiera blokad (pułapek). Otrzymane implikanty proste przedstawiają rozwiązania analizowanego równania. Rozwiązania przedstawione są w postaci wektorów wartościowych (0,1,-). Do zbioru miejsc opisujących blokady (pułapki) należą miejsca reprezentowane przez literał negatywny, co wynika z równania (11) w rozdziale 5.3.1.

Iloczyn negatywnych literałów reprezentujących wszystkie miejsca, oznacza, że to rozwiązanie odpowiada całej sieci, czyli cała sieć jest blokadą (pułapką).

UWAGA:

Z wykorzystaniem opisanej metody można wyznaczać blokady i pułapki dla sieci Petriego należących do różnych klas. Jednakże w celu określenia żywotności wykorzystuje się własność Commoner'a, za pomocą której bada się sieci należące do klasy EFC.

Na podstawie eksperymentów można stwierdzić, że ograniczenie w postaci koniunkcyjnej funkcji boolowskiej do postaci klauzulowej, a w szczególności do iloczynu klauzul Horna, powoduje że drzewo ma charakter zwarty w porównaniu z ogólnym grafem BDD, oraz że gałęzie nie wpływające na wynik obcinane są wcześnie. W związku z tym, że algorytm wykorzystywany jest dla sieci Petriego modelujących rzeczywiste procesy, nie wystąpi sytuacja, że drzewo nie będzie skracane, oraz że proces tworzenia i przeszukiwania drzewa nie będzie skończony. Po wyznaczeniu implikantu prostego, odpowiednia gałąź z pamięci jest usuwana, co pozwala na przetwarzanie dużej ilości węzłów w drzewie. Liczba rozwiązań (określających blokady i pułapki) dotycząca układów rzeczywistych nie jest duża, w związku z tym obszar pamięci operacyjnej wystarcza, a przetwarzanie wykonywane jest w realnym czasie.

7.4. Badanie żywotności

Algorytm wyznaczania wszystkich blokad i pułapek można zastosować do sprawdzenia żywotności sieci Petriego. Aby określić, czy dana sieć jest żywa, należy sprawdzić zależności między blokadami a pułapkami. Dla klasy sieci FC i EFC można zastosować własność Commonera [COMMONER72], natomiast dla sieci z klasy AC i NSC można zastosować twierdzenie 5 z rozdziału 2.3.

Po wyznaczeniu rozwiązań dla formuł Horna opisujących blokady i pułapki, należy sprawdzić, zgodnie z własnością Commoner'a [COMMONER72], czy każda wyznaczona blokada zawiera oznakowaną pułapkę. Z definicji 25 (rozdział 2.3) wynika, że każda blokada zawiera blokadę minimalną bądź jest blokadą minimalną. Stąd wynika własność, że jeżeli każda minimalna blokada zawiera oznakowaną pułapkę, to każda blokada zawiera oznakowaną pułapkę. Dlatego, aby sprawdzić żywotność sieci Petriego wystarczy sprawdzić jedynie minimalne blokady, gdyż są one zawarte w każdej innej blokadzie. Podobnie jest w przypadku pułapek, wystarczy sprawdzenie zawierania się minimalnych pułapek, gdyż jeżeli blokada zawiera nieminimalne pułapki, to tym samym zawiera minimalne. Z powyższych rozważań wynika własność, że **aby sprawdzić żywotność sieci Petriego z klasy FC i EFC, należy rozpatrywać zawieranie się minimalnych pułapek oznakowanych w znakowaniu początkowym w minimalnych blokadach**. Należy tu zwrócić uwagę, że z wykorzystaniem takiej własności można badać tylko sieci spójne. Takie podejście jest wystarczające w przypadku sieci modelujących układy sterowania, a znacznie przyspiesza analizę sieci.

W pierwszym kroku należy przeanalizować, czy każda blokada jest oznakowana w znakowaniu początkowym. Jeżeli, któraś z blokad nie jest oznakowana, oznacza to, że sieć nie jest żywa. Wynika to z faktu, że aby sieć była żywa, to każda blokada musi zawierać oznakowaną pułapkę, co wiąże się z tym, że każda blokada również musi być oznakowana. Kolejnym krokiem jest sprawdzenie, czy każda minimalna blokada zawiera minimalną oznakowaną pułapkę. Jeżeli znajdzie się choć jedna blokada, która nie zawiera takiej pułapki, oznacza to, że sieć Petriego opisana takimi formułami Horna jest nieżywa, w przeciwnym przypadku badana sieć jest żywa.

Z wykorzystaniem symbolicznej metody analizy sieci Petriego można badać sieci:

- z klasy FC i EFC:
  - spójne,
  - bez tranzycji typu źródło (tzn. $t \in T$, takich że $\bullet t=0$),
  - bez tranzycji typu ujście (tzn. $t \in T$, takich że $t\bullet=0$);

  stosując zmodyfikowaną własność Commoner'a (opisaną powyżej).
- z klasy AC i NSC:
  - spójne,
  - ograniczone,
  - bez tranzycji typu źródło (tzn. $t \in T$, takich że $\bullet t=0$),
  - bez tranzycji typu ujście (tzn. $t \in T$, takich że $t\bullet=0$)

  stosując twierdzenie 5 z rozdziału 2.3, czyli badanie, czy każda minimalna blokada jest równa pułapce.

7.5. Badanie ograniczoności

Ograniczoność sieci Petriego jest własnością, która podobnie jak żywotność, zależy od występowania blokad i pułapek w sieci.

Wykorzystując algorytm wyznaczania wszystkich blokad i pułapek w sieci Petriego opisanych formułami Horna, można określić, czy badana sieć jest ograniczona. Podzbiory miejsc, które spełniają w tym samym czasie równania Horna opisujące zarówno blokady jak i pułapki, związane są z P-niezmiennikami [MINOUX,BARKAOUI90].

P-niezmienniki otrzymywane mogą być w dwojaki sposób [MINOUX,BARKAOUI90]:

- należy porównać otrzymane zbiory miejsc określające blokady i pułapki; jeżeli dwa takie zbiory odpowiadają sobie jeden do jednego, wówczas zbiór ten może odpowiadać P-niezmiennikowi,
- należy wyznaczyć formułę Horna, składającą się zawsze z formuły opisującej blokady jak i z formuły opisującej pułapki. Następnie taką formułę przyrównać do 1 i rozwiązać, a otrzymane wyniki mogą odpowiadać P-niezmiennikom.

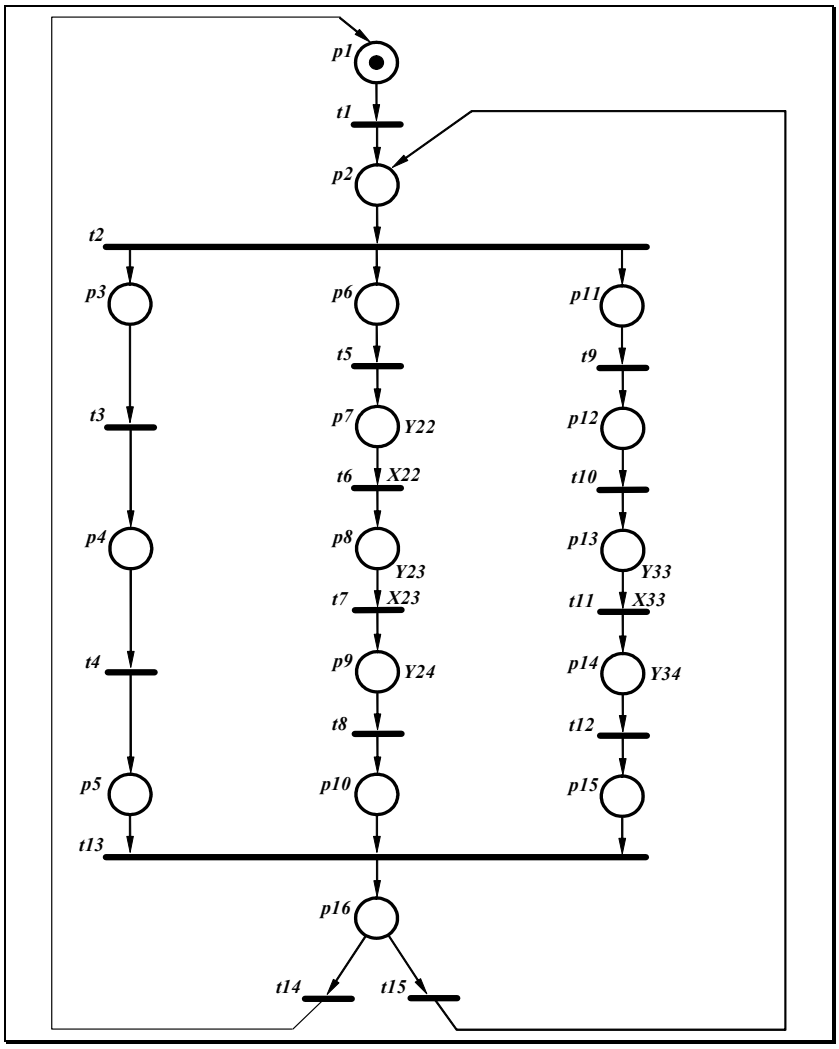
Aby stwierdzić, które z rozwiązań są P-niezmiennikami, należy każdy wektor wymnożyć przez macierz incydencji. Zgodnie z definicją z rozdziału 2.1, jeżeli wynikiem będzie wektor zerowy, wówczas badany wektor jest P-niezmiennikiem [REISIG88]. Kolejnym krokiem jest sprawdzenie, czy wyznaczone P-niezmienniki pokrywają całą sieć. W przypadku pokrycia całej sieci można stwierdzić, że jest ona ograniczona. W przeciwnym przypadku nie można stwierdzić, że sieć nie jest ograniczona [REISIG88].

7.6. Przykład analizy sieci żywej

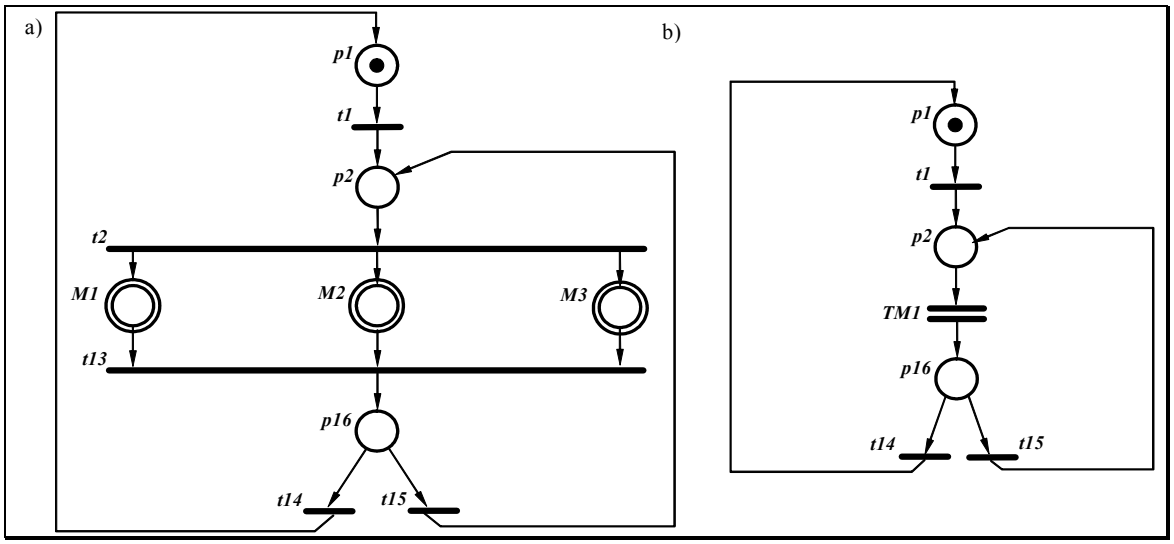
Zaproponowana metoda symbolicznej analizy sieci Petriego zostanie przedstawiona na przykładzie analizy sieci wzorowanej na sieci z [FERRARINI92]. Sieć ta należy do klasy sieci FC. Rys. 52 przedstawia sieć Petriego, opisującą algorytm działania stanowiska do wiercenia. Dla sieci tej zastosowano algorytm redukcji. Kolejne etapy redukcji sieci zostały przedstawione na Rys. 53 a i b.

Miejscom *p3*, *p4*, *p5* odpowiada makromiejsce *M1*, miejscom *p6*, *p7*, *p8*, *p9*, *p10* odpowiada makromiejsce *M2*, natomiast miejscom *p11*, *p12*, *p13*, *p14*, *p15* odpowiada makromiejsce *M3*.

Makromiejscom *M1*, *M2*, *M3* oraz tranzycjom *t2* i *t13* odpowiada makrotranzycja *TM1*.



Rys. 52. Zmodyfikowany model sieci Petriego opisujący stanowisko do wiercenia



Rys. 53. Pierwszy i drugi etap redukcji sieci Petriego z Rys. 52

Na Rys. 54 zamieszczono kolejne formuły Horna *HF1*, *HF2*, *HF3* opisujące blokady w sieciach Petriego, odpowiednio z Rys. 52, Rys. 53a i b.

```
HF1 = (/p1 + p2) * (/p2 + p3) * (/p2 + p6) * (/p2 + p11) *
      (/p3 + p4) * (/p4 + p5) * (/p6 + p7) * (/p7 + p8) *
(/p8 + p9) * (/p9 + p10) * (/p11 + p12) * (/p12 + p13) *
(/p13 + p14) * (/p14 + p15) * (/p5 + /p10 + /p15 + p16) *
(/p16 + p1) * (/p16 + p2)

HF2 = (/p1 + p2) * (/p2 + M1) * (/p2 + M2) * (/p2 + M3) *
      (/M1 + /M2 + /M3 + p16) * (/p16 + p1) * (/p16 + p2)

HF3 = (/p1 + p2) * (/p2 + p16) * (/p16 + p1) * (/p16 + p2)
```

Rys. 54. Formuła Horna opisująca blokady w sieci Petriego

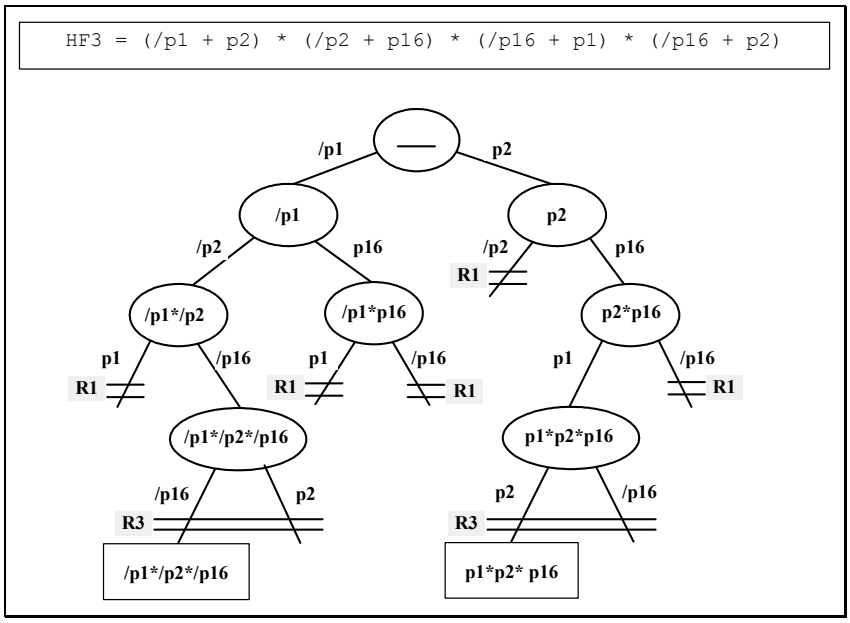
Formuła *HF1* składa się z 18 klauzul. Po pierwszym etapie redukcji sieci, formuła *HF2* składa się tylko z 8 klauzul. Natomiast po kolejnej redukcji sieci, formuła *HF3* składa się już tylko z 4 klauzul. Redukcja sieci może więc w znacznym stopniu uprościć formułę Horna.

Na Rys. 55 zamieszczono formułę Horna *HF3'*, opisującą pułapki w sieci Petriego z Rys. 53b.

```
HF3′ = (p1 + /p2) * (p2 + /p16) * (p16 + /p1) * (p16 + /p2)
```

Rys. 55. Formuła Horna opisująca pułapki w sieci Petriego z Rys. 53b

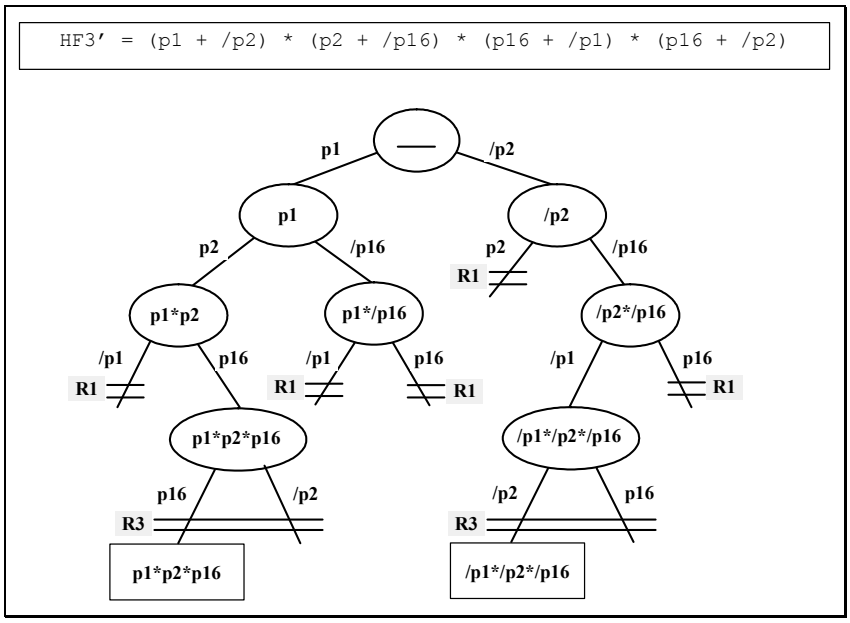
Zgodnie z regułami podanymi w tym rozdziale 7.3, zostały przygotowane drzewa Thelena dla formuł Horna opisujących blokady i pułapki. Na Rys. 56 przedstawione jest drzewo dla formuły *HF3* z Rys. 54.



Rys. 56. Drzewo Thelena dla formuły Horna HF3

Wygenerowane drzewo Thelena, dla formuły Horna *HF3*, zawiera dwa rozwiązania. Wektor `{p1,p2,p16}` otrzymany na podstawie implikantu prostego `/p1*/p2*/p16`, pokrywa całą sieć, oznacza to, że cała sieć jest blokadą.

Następnie generowane jest drzewo dla formuły Horna opisującej pułapki (Rys. 55). Konstrukcja drzewa jest analogiczna.



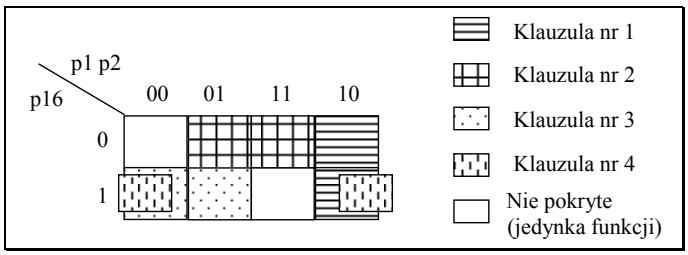
Rys. 57. Drzewo Thelena dla formuły Horna z Rys. 55

Wygenerowane dla formuły Horna z Rys. 55 drzewo Thelena, zawiera dwa rozwiązania. Podobnie jak w przypadku wyznaczania blokad, wygenerowany został wektor `{p1,p2,p16}` otrzymany na podstawie implikantu prostego `/p1*/p2*/p16`. Wektor ten pokrywa całą sieć, oznacza to, że cała sieć jest pułapką. Na podstawie uzyskanych dwóch wyników można stwierdzić, że sieć Petriego z Rys. 52 jest żywa, gdyż wygenerowana blokada zawiera pułapkę oznakowaną w znakowaniu początkowym. Na podstawie twierdzenia z rozdziału 2.4, można stwierdzić, że odpowiadająca jej sieć płaska również jest żywa.

W celu zilustrowania otrzymanych wyników, dla formuły Horna *HF3*, gdzie klauzule mają odpowiednio numery porządkowe:

1) `(/p1 + p2)`
2) `(/p2 + p16)`
3) `(/p16 + p1)`
4) `(/p16 + p2)`

przygotowano siatkę Karnaugh (Rys. 58).

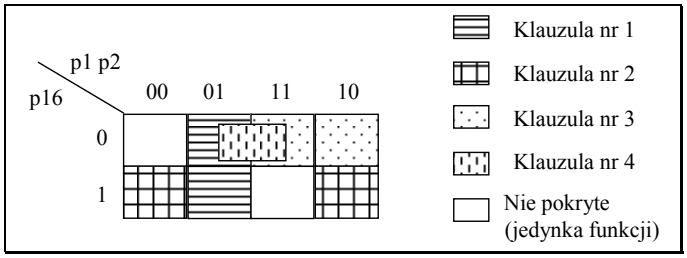


Rys. 58. Siatka Karnaugh dla formuły Horna HF3

Natomiast dla formuły *HF3'*, gdzie klauzule mają odpowiednio numery porządkowe:

1) `(p1 + /p2)`
2) `(p2 + /p16)`
3) `(p16 + /p1)`
4) `(p16 + /p2)`

siatka Karnaugh przedstawiona jest na Rys. 66.



Rys. 59. Siatka Karnaugh dla formuły Horna HF3'

W obu przypadkach niepokryte zostały dwa pola oznaczające rozwiązania: afirmacja dla każdego literału i negacja dla każdego literału. Czyli sieć opisana formułami *HF3* i *HF3'* zawiera odpowiednio blokadę i pułapkę, które odpowiadają całej sieci.

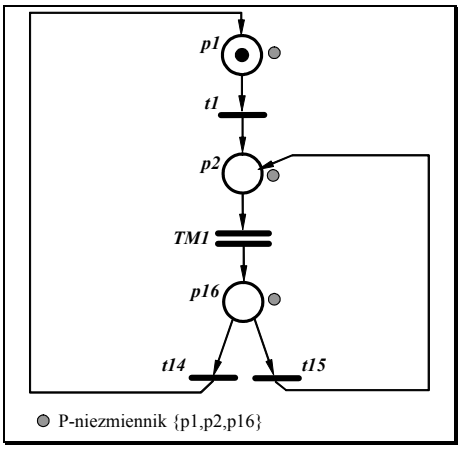
Następnie, aby wyznaczyć P-niezmienniki należy sprawdzić, które z wektorów przedstawiających blokady i pułapki, pokrywają się. W rozpatrywanym przypadku jest to wektor `{p1,p2,p16}`, który można przedstawić w postaci [1,1,1]. Wektor ten należy wymnożyć przez macierz incydencji z Rys. 60.

```
      p1 p2 p16
t1   -1  1  0
TM1   0 -1  1
t14   0  1 -1
t15   1  0 -1
```

Rys. 60. Macierz incydencji

Po wymnożeniu uzyskano wynik [0,0,0,0], co oznacza, że badany wektor `{p1,p2,p16}` jest P-niezmiennikiem sieci. Cała sieć pokryta jest przez ten P-niezmiennik, czyli jest siecią ograniczoną. Na Rys. 61 została przedstawiona badana sieć pokryta P-niezmiennikiem. Cała sieć została pokryta P-niezmiennikiem, oznacza to, że

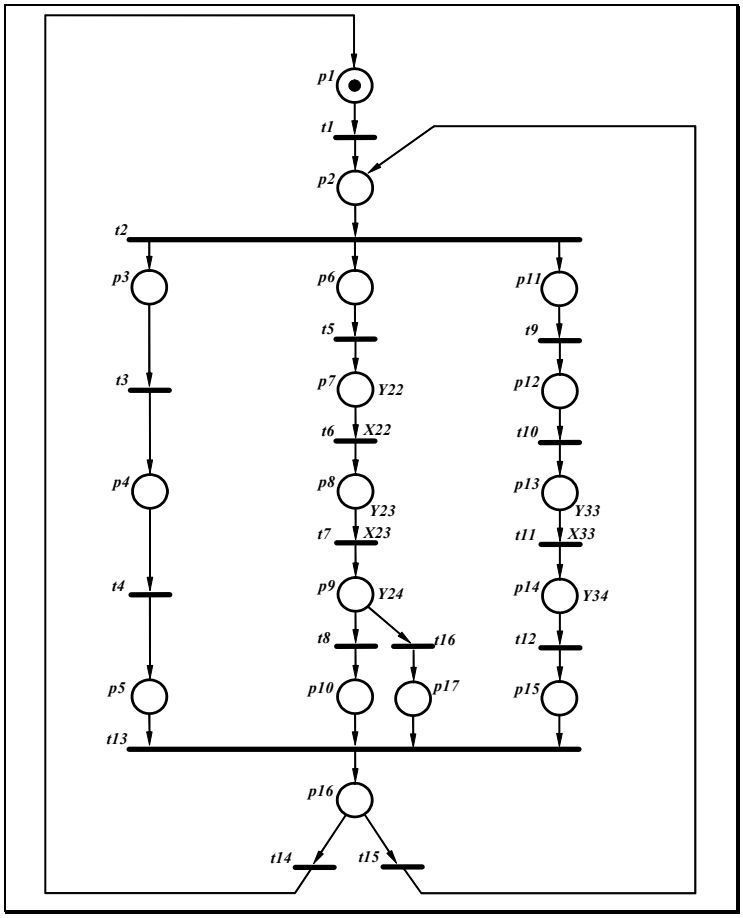
sieć jest ograniczona. Na podstawie twierdzenia z rozdziału 2.4, można stwierdzić, że odpowiadająca jej sieć płaska również jest ograniczona.



Rys. 61. Sieć Petriego pokryta P-niezmiennikiem

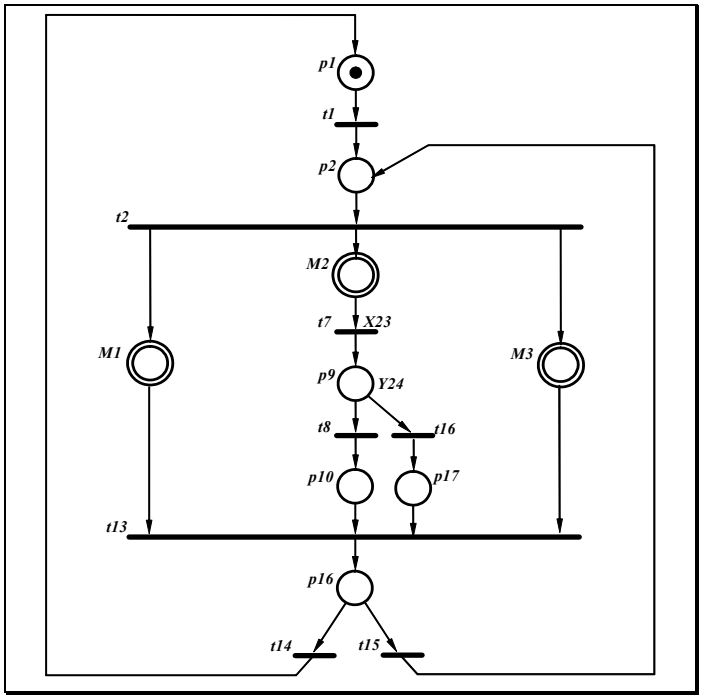
7.7. Przykład analizy sieci nieżywej

W celu zaprezentowania działania metody dla sieci nieżywej, sieć z Rys. 52 zmodyfikowano dodając miejsce *p17* i tranzycję *t16* (Rys. 62).



Rys. 62. Model sieci Petriego z Rys. 52, z dodatkowym miejscem i tranzycją

Rys. 63 przedstawia zredukowaną sieć Petriego z Rys. 62. Miejscom *p3*, *p4* oraz *p5* odpowiada makromiejsce *M1*, miejscom *p6*, *p7*, *p8*, makromiejsce *M2*, a miejscom *p11*, *p12*, *p13*, *p14*, oraz *p15*, makromiejsce *M3*.



Rys. 63. Sieć Petriego z Rys. 62 z wyznaczonymi makromiejscami

Rys. 64 przedstawia formuły Horna opisujące blokady w sieciach Petriego, odpowiednio z Rys. 62 oraz Rys. 63. Formuła *HF11* składa się z 18 klauzul. Natomiast formuła Horna *HF22*, składa się z 10 klauzul. W tym przypadku również w znaczny sposób formuła Horna została uproszczona.

```
HF11 = (/p1 + p2) * (/p2 + p3) * (/p2 + p6) * (/p2 + p11)
*   (/p3 + p4) * (/p4 + p5) * (/p6 + p7) * (/p7 + p8)
*   (/p8 + p9) * (/p9 + p10) * (/p11 + p12) * (/p12 + p13)
*   (/p13 + p14) * (/p14 + p15) * (/p5 + /p10 + /p15 + /p17
+ p16) * (/p9 + p17) * (/p16 + p1) * (/p16 + p2)

HF22 = (/p1 + p2) * (/p2 + M1) * (/p2 + M2) * (/p2 + M3) *
     (/M2 * p9) + (/p9 * p10) + (/p9 * p17) + (/M1 + /p10
+ /p17 + /M3 + p16) * (/p16 + p1) * (/p16 + p2)
```

Rys. 64. Formuły Horna opisujące blokady

Na Rys. 65 przedstawiono formułę Horna *HF22'*, opisującą pułapki w sieci Petriego z Rys. 63.

```
HF22′ = (p1 + /p2) * (p2 + /M1 + /M2 + /M3) * (/p9 + M2) *
(p9 + /p10) * (p9 * /p17) + (M1 * /p16) + (p10 * /p16) +
(p17 + /p16) * (M3 + /p16) * (p16 + /p1) + (p16 + /p2)
```

Rys. 65. Formuła Horna opisująca pułapki w sieci Petriego z Rys. 63

Ze względu na złożoność drzew Thelena dla formuły Horna opisującej blokady (Rys. 64) oraz formuły opisującej pułapki (Rys. 65), nie zamieszczono rysunków. W przypadku pierwszego drzewa otrzymano 57 węzłów, natomiast rozwiązania są przedstawione na Rys. 66.

```
RESULT_HF22 = (/p1 * /p2 * p9 * p10 * /p16 * p17 * /M3) +
              (/p1 * /p2 * p9 * p10 * /p16 * p17 * /M1) +
              (/p1 * /p2 * p10 * /p16 * p17 * /M2 * /M3) +
              (/p1 * /p2 * p10 * /p16 * p17 * /M1 * /M2) +
              (/p1 * /p2 * /p9 * /p10 * /p16 * /M2) +
              (/p1 * /p2 * /p9 * /p16 * /M2 * /M3) +
              (/p1 * /p2 * /p9 * /p16 * /M1 * /M2) +
              (/p1 * /p2 * /p9 * /p16 * /p17 * /M2) +
              (p1 * p2 * p9 * p10 * 16 * p17 * M1 * M2 * M3)
```

Rys. 66. Blokady dla sieci z Rys. 62

W przypadku drugiego drzewa otrzymano 158 węzłów, a rozwiązania są przedstawione na Rys. 67.

```
RESULT_HF22' = (p1 * p2 * p9 * p10 * p16 * p17 * M1 * M2 * M3) +
               (/p1 * /p2 * /p9 * /p10 * /p16 * /p17 * /M3) +
               (/p1 * /p2 * /p10 * /p16 * /p17 * M2 * /M3) +
               (/p1 * /p2 * p9 * /p16 * M2 * /M3) +
               (/p1 * /p2 * /p9 * /p10 * /p16 * /p17 * /M1) +
               (/p1 * /p2 * /p10 * /p16 * /p17 * /M1 * M2) +
               (/p1 * /p2 * p9 * /p16 * /M1 * M2) +
               (/p1 * /p2 * /p9 * /p10 * /p16 * /p17 * /M2)
```

Rys. 67. Pułapki dla sieci z Rys. 62

Na podstawie rozwiązań (Rys. 66, Rys. 67), wyznaczony został zbiór wszystkich blokad i zbiór wszystkich pułapek, jak w tabeli (Tabela 5). Blokady i pułapki zaznaczone szarym kolorem, to blokady i pułapki minimalne.

Na podstawie zaprezentowanych wyników można stwierdzić, że badana sieć Petriego zawiera blokady i pułapki. Jednakże, nie można na tym etapie określić, czy sieć jest żywa.

Kolejnym etapem jest sprawdzenie na podstawie uzyskanych wyników, czy badana sieć jest żywa. W pierwszym kroku należy sprawdzić, czy wszystkie minimalne blokady są oznakowane w znakowaniu początkowym. Dla analizowanego przykładu nie znaleziono minimalnej blokady nieoznakowanej w znakowaniu początkowym. Następnie wyznaczane są pułapki minimalne ze zbioru wszystkich pułapek oraz sprawdzane, które z nich są oznakowane w znakowaniu początkowym. W prezentowanym przykładzie otrzymano trzy pułapki minimalne. Dla analizowanego przykładu (Rys. 63) w znakowaniu początkowym oznakowane jest miejsce *p1*. Wśród zbioru rozwiązań (Tabela 5) opisującego pułapki minimalne, nie występuje żadna pułapka nie oznakowana w znakowaniu początkowym. Dwa zbiory miejsc reprezentujące blokadę `{p1,p2,p9,p10,p16,M2}` oraz

{p1,p2,p9,p16,p17,M2}, nie zawierają oznakowanej pułapki w znakowaniu początkowym *p1*. Oznacza to, że sieć Petriego nie jest żywa.

Tabela 5. Zbiór wszystkich blokad i pułapek dla sieci z Rys. 63

Blokady	*Pułapki*
{p1,p2,p16,M3}	{p1,p2,p9,p10,p16,p17}
{p1,p2,p16,M3,M1}	{p1,p2,p9,p10,p16,p17,M3}
{p1,p2,p16,M3,M1,M2}	{p1,p2,p9,p10,p16,p17,M1,M3}
{p1,p2,p16,M3,M2}	{p1,p2,p9,p10,p16,p17,M2,M3}
{p1,p2,p16,M1}	{p1,p2,p9,p10,p16,p17,M1,M2,M3}
{p1,p2,p16,M1,M2}	{p1,p2,p10,p16,p17,M3}
{p1,p2,p9,p16,M2,M3}	{p1,p2,p10,p16,p17,M1,M3}
{p1,p2,p9,p16,M3,M1,M2}	{p1,p2,p16,M3}
{p1,p2,p9,p16,M1,M2}	{p1,p2,p16,p10,M3}
{p1,p2,p9,p10,p16,M2}	{p1,p2,p16,p17,M3}
{p1,p2,p9,p10,p17,p16,M2}	{p1,p2,p16,M1,M3}
{p1,p2,p9,p10,p16,M1,M2}	{p1,p2,p16,p10,M1,M3}
{p1,p2,p9,p10,p16,M2,M3}	{p1,p2,p16,p17,M1,M3}
{p1,p2,p9,p10,p16,p17,M2,M1}	{p1,p2,p9,p10,p16,p17,M1}
{p1,p2,p9,p10,p16,p17,M2,M1,M3}	{p1,p2,p9,p10,p16,p17,M1,M2}
{p1,p2,p9,p10,p16,p17,M2,M3}	{p1,p2,p10,p16,p17,M1}
{p1,p2,p9,p10,p16,M1,M2,M3}	{p1,p2,p16,M1}
{p1,p2,p9,p16,p17,M2,M3}	{p1,p2,p10,p16,M1}
{p1,p2,p9,p16,p17,M1,M2,M3}	{p1,p2,p16,p17,M1}
{p1,p2,p9,p16,p17,M1,M2}	{p1,p2,p9,p10,p16,p17,M2}
{p1,p2,p9,p16,p17,M2}	

Na podstawie wyznaczonych blokad i pułapek można wyznaczyć P-niezmienniki sieci. P-niezmiennikiem sieci jest ten zbiór miejsc, który odpowiada zarówno blokadzie jak i pułapce, czyli blokada równa jest pułapce. W tabeli Tabela 6 zostały zamieszczone wszystkie blokady, które odpowiadają pułapkom, dla sieci z Rys. 63.

Tabela 6. Blokady pokrywające pułapki dla sieci z Rys. 63

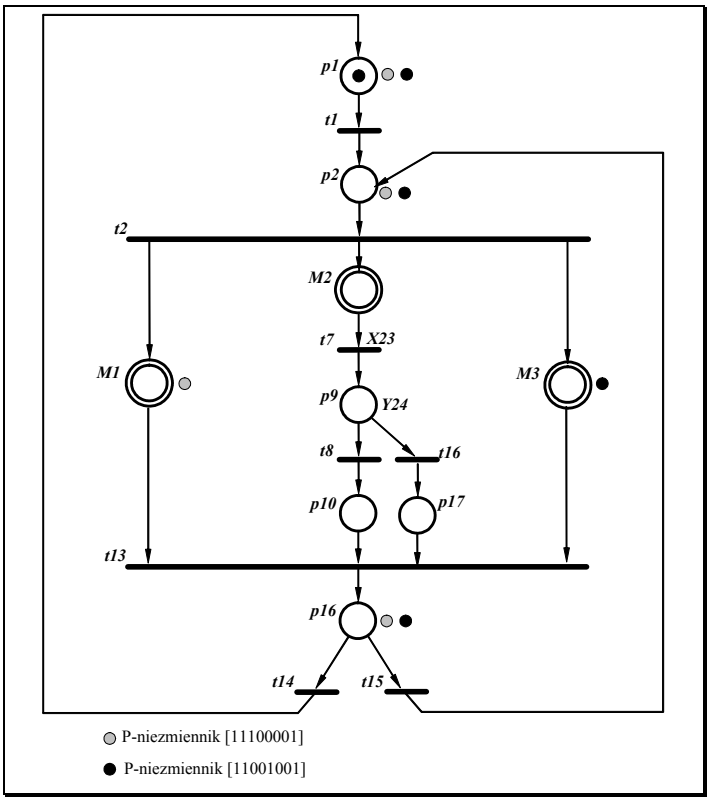
Blokady pokrywające pułapki
{p1,p2,p16,M3}
{p1,p2,p16,M1}
{p1,p2,p16,M3,M1}
{p1,p2,p9,p10,p17,p16,M2}
{p1,p2,p9,p10,p16,p17,M2,M1}
{p1,p2,p9,p10,p16,p17,M2,M1,M3}
{p1,p2,p9,p10,p16,p17,M2,M3}

Zbiorom z tabeli (Tabela 6) odpowiadają poniżej przedstawione wektory: [110010001], [111000001], [111010001], [110101111], [111101111], [111111111], [110111111]. Następnie każdy wektor wymnożono przez macierz incydencji z Rys. 68.

	p1	p2	M1	M2	M3	p9	p10	p17	p16
t1	-1	1	0	0	0	0	0	0	0
t2	0	-1	1	1	1	0	0	0	0
t7	0	0	0	-1	0	1	0	0	0
t8	0	0	0	0	0	-1	1	0	0
t16	0	0	0	0	0	-1	0	1	0
t13	0	0	-1	0	-1	0	-1	-1	1
t14	1	0	0	0	0	0	0	0	-1
t15	0	1	0	0	0	0	0	-1	0

Rys. 68. Macierz incydencji

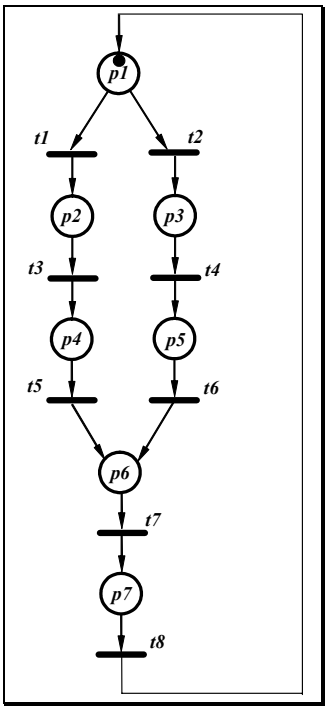
Na podstawie opracowanej metody nie można określić, czy badana sieć jest ograniczona. Wynika to z faktu, że zostały wyznaczone tylko dwa P-niezmienniki `[110010001]` oraz `[111000001]`, które nie pokrywają sieci. Stąd można wywnioskować, że sieć nie jest kolorowalna, czyli posiada defekt. Na Rys. 69 przedstawiono pokrycie sieci P-niezmiennikami. P-niezmiennikowi `[111000001]` odpowiada kropka szara, natomiast P-niezmiennikowi `[110010001]` kropka czarna.



Rys. 69. Pokrycie sieci Petriego P-niezmiennikami

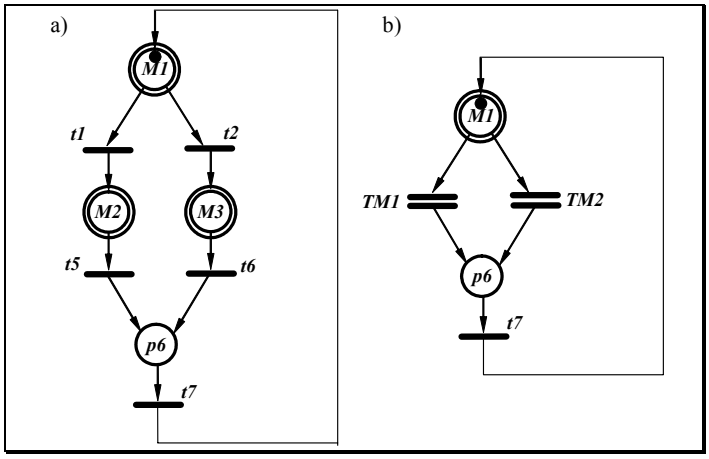
7.8. Przykład analizy sieci automatowej

W rozdziale tym zostanie przedstawiony przykład analizy sieci automatowej z rozgałęzieniem z klasy FC (Rys. 70).



Rys. 70. Automatowa sieć Petriego

Miejscom *p1*, *p7* odpowiada makromiejsce *M1*, miejscom *p2*, *p4* odpowiada makromiejsce *M2*, natomiast miejscom *p3*, *p5* odpowiada makromiejsce *M3* (Rys. 71a).



Rys. 71. Automatowa sieć Petriego –a) pierwszy, b) drugi poziom hierarchii

Makromiejscom *M2*, *M3* oraz tranzycjom *t1*, *t5* i *t2*, *t6* odpowiada odpowiednio makrotranzycja *TM1* i *TM2* (Rys. 71b).

Na Rys. 72 zamieszczono formuły Horna *HF3* i *HF3′* opisujące odpowiednio blokady i pułapki w automatowej sieci Petriego z Rys. 71b.

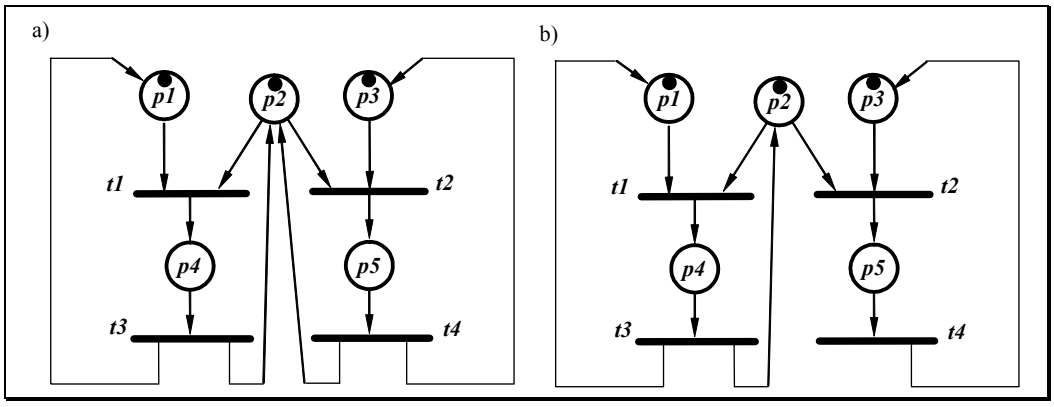
```
HF3 = (/M1 + p6) * (/M1 + p6) * (/p6 + M1)

HF3' = (M1 + /p6) * (M1 + /p6) * (p6 + /M1)
```

Rys. 72. Formuły Horna opisujące blokady i pułapki

Na podstawie drzew Thelena dla formuł Horna opisujących blokady i pułapki otrzymano dwa rozwiązania, dla blokad `{M1,p6}` oraz dla pułapek `{M1,p6}`. Oznacza to, że cała sieć jest zarówno blokadą jak i pułapką, czyli otrzymana blokada pokrywa się z oznakowaną pułapką. Wyznaczony również został jeden P-niezmiennik sieci pokrywający całą sieć `{M1,p6}`. Na podstawie otrzymanych wyników można stwierdzić, że analizowana sieć Petriego jest żywa i ograniczona oraz jest typu SM (rozdział 2.1).

7.9. Przykład analizy sieci z klasy AC

W rozdziale tym zostanie przedstawiony przykład analizy sieci żywej i nieżywej z klasy AC (Rys. 73a i b). Ze względu na prosty przykład pominięto redukcję obu sieci. Kolejnym etapem jest wyznaczanie formuł Horna opisujących blokady i pułapki. Dla sieci z Rys. 73a obie formuły *HF* i *HF'* odpowiadające odpowiednio blokadom i pułapkom, zamieszczono na Rys. 74, natomiast dla sieci z Rys. 73b formuły *HF* i *HF'* odpowiadające odpowiednio blokadom i pułapkom, zamieszczono na Rys. 75.



Rys. 73. Przykład sieci a) żywej, b) nieżywej z klasy AC

```
HF = (/p1 + /p2 + p4) * (/p2 + /p3 + p5) * (/p4 + p1) *
     (/p4 + p2) * (/p5 + p2) * (/p5 + p3)

HF' = (p1 + /p4) * (p2 + /p4) * (p2 + /p5) * (p3 + /p5) *
      (p4 + /p1 + /p2) * (p5 + /p2 + /p3)
```

Rys. 74. Formuły Horna opisujące blokady i pułapki dla sieci żywej

```
HF = (/p1 + /p2 + p4) * (/p2 + /p3 + p5) * (/p4 + p1) *
     (/p4 + p2) * (/p5 + p3)

HF' = (p1 + /p4) * (p2 + /p4) * (p2 + /p5) * (p3 + /p5) *
      (p4 + /p1 + /p2) * (p5 + /p3)
```

Rys. 75. Formuły Horna opisujące blokady i pułapki dla sieci nieżywej

Dla powyższych formuł wyznaczono rozwiązania, a na ich podstawie blokady i pułapki. W tabeli (Tabela 7) zamieszczono wszystkie blokady i pułapki dla sieci z Rys.

73a. Blokady i pułapki minimalne zaznaczono kolorem szarym. Każda blokada minimalna jest pułapką, co oznacza, że sieci z Rys. 73a jest żywa. Na podstawie uzyskanych wyników można wyznaczyć wektory wśród których są P-niezmienniki sieci. Po wymnożeniu poszczególnych wektorów przez macierz incydencji otrzymano P-niezmienniki: `{p1,p4}`, `{p3,p5}`, `{p2,p4,p5}`, które pokrywają całą sieć, na podstawie czego można stwierdzić, że sieć jest ograniczona.

Tabela 7. Zbiór wszystkich blokad i pułapek dla sieci z Rys. 73a

Blokady	*Pułapki*
{p1, p4}	{p1, p4}
{p3, p5}	{p3, p5}
{p2, p4, p5}	{p2, p4, p5}
{p1, p3, p4, p5}	{p1, p3, p4, p5}

W tabeli (Tabela 8) zamieszczono wszystkie blokady i pułapki dla sieci z Rys. 73b. Blokady i pułapki minimalne zaznaczono kolorem szarym. Blokada minimalna `{p2,p4}` nie jest równa żadnej z wyznaczonych pułapek, co oznacza, że sieć z Rys. 73b nie jest żywa.

Tabela 8. Zbiór wszystkich blokad i pułapek dla sieci z Rys. 73b

Blokady	*Pułapki*
{p1, p4}	{p1, p4}
{p3, p4}	{p3, p4}
{p2, p4}	{p2, p3, p4, p5}
{p2, p4, p5}	{p1, p3, p4, p5}
{p1, p3, p4, p5}	

7.10. Wyniki eksperymentalne

Konstrukcja drzewa Thelena jest zależna od kolejności klauzul. W tym celu badano wpływ zależności na rozmiar drzewa. W rozdziale tym zostaną zaprezentowane wyniki doboru kolejności klauzul według następujących kryteriów:

- długości klauzul,
- podobieństwa,
- indeksu podobieństwa,
- naturalnej kolejności w opisie sieci Petriego, zorientowanym na tranzycje.

Sortowanie według długości klauzul polega na uporządkowaniu klauzul w formule według ich długości, czyli liczby literałów występujących w klauzuli. Najkrótsze klauzule umieszczane są na początku, najdłuższe na końcu formuły. Porządek klauzul o tej samej długości nie jest określany.

W sortowaniu według podobieństwa, klauzule ustawiane są tak, aby obok siebie występowały klauzule najmniej różniące się od siebie (w których występuje jak najwięcej takich samym literałów). Proces sortowania rozpoczyna się od wybrania najkrótszej klauzuli w formule. Jeżeli występuje więcej klauzul o tej samej długości, to wybierana jest pierwsza napotkana. Jako druga wybierana jest klauzula najmniej różniąca się od pierwszej, a jako trzecia, najmniej różniąca się od drugiej i tak dalej, aż posortowana zostanie cała formuła.

Sortowanie według indeksu podobieństwa polega na ustawieniu klauzul według częstości występowania literałów w formule. Klauzule zawierające najczęściej występujące literały, umieszczane są na początku formuły, natomiast klauzule z rzadko występującymi literałami, na końcu.

Sortowanie według długości i indeksu podobieństwa jest połączeniem obu metod sortowania. W pierwszej kolejności wykonywane jest sortowanie według długości, a następnie klauzule o tej samej długości są ustawiane według indeksu podobieństwa. Indeks podobieństwa wyznaczany jest na podstawie częstości występowania literałów w całej formule, a nie tylko w grupie klauzul o tej samej długości.

Sortowanie klauzul przedstawiono na przykładzie formuły Horna dla sieci z Rys. 63. W przypadku gdy, kolejność klauzul w formule jest taka, jak po wygenerowaniu na podstawie opisu sieci Petriego zorientowanego na tranzycje (Rys. 76), otrzymano 57 węzłów w drzewie Thelena.

```
HF_TSP = (/p1 + p2) * (/p2 + M1) * (/p2 + M2) * (/p2 + M3) *
         (/M2  +p9)  *  (/p9  +  p10)  *  (/p9  +  p17)  *
         (/M1  +  /p10  +  /p17  +  /M3  +  p16)*  (/p16  +  p1)  *
         (/p16 +p2)
```

Rys. 76. Porządek klauzul zgodny z kolejnością tranzycji

W przypadku gdy, posortowano klauzule według długości (Rys. 77), otrzymano 47 węzłów w drzewie Thelena.

```
HF_DL = (/p1 + p2) * (/p2 + M1) * (/p2 + M2) * (/p2 + M3) *
        (/M2 +p9) * (/p9 + p10) * (/p9 + p17) * (/p16 + p1) *
        (/p16 +p2) * (/M1 + /p10 + /p17 + /M3 + p16)
```

Rys. 77. Sortowanie klauzul według długości

W przypadku sortowania według podobieństwa (Rys. 78), otrzymano 46 węzłów w drzewie Thelena.

```
HF_CW = (/p1 + p2) * (/p16 +p2) * (/p16 + p1) * (/p2 + M1) *
        (/p2 + M2) * (/p2 + M3) * (/M2 +p9) * (/p9 + p10) *
        (/p9 + p17) * (/M1 + /p10 + /p17 + /M3 + p16)
```

Rys. 78. Sortowanie klauzul według podobieństwa

W przypadku sortowania według indeksu podobieństwa (Rys. 79), otrzymano 45 węzłów w drzewie Thelena.

```
HF_IP = (/p2 + M1) * (/p2 + M2) * (/p2 + M3) * (p2 + /p16) *
        (/p1 + p2) * (p9 + /M2) * (/p9 + p10) * (/p9 + p17) *
        (/p10 + p16 + /p17 + /M1 + /M3) * (p1 + /p16)
```

Rys. 79. Sortowanie klauzul według indeksu podobieństwa

Tabela 9 przedstawia zestawienie wyników dla formuł Horna, opisujących blokady i pułapki dla sieci z różnych rozdziałów pracy. Podano w niej również stosunek liczby węzłów w drzewie dla formuły po sortowaniu do liczby węzłów w drzewie dla formuły otrzymanej zgodnie z kolejnością tranzycji. Dla rozpatrywanych przypadków, najlepszym uporządkowaniem klauzul jest sortowanie według długości klauzul (średnio 78,24% w stosunku do naturalnej kolejności w opisie sieci Petriego, zorientowanym na tranzycje).

W tabeli (Tabela 10) zaprezentowano wyniki, dla sieci wygenerowanych losowo, przez program opracowany na uniwersytecie w Mińsku. Program generuje sieci z klasy EFC, na podstawie trzech parametrów:

- liczba miejsc w sieci;
- liczba tranzycji w sieci;
- liczba tranzycji, mających wspólne miejsca wejściowe.

Przykładowo, w tabeli przeanalizowano przykład sieci z dwudziestoma miejscami, piętnastoma tranzycjami oraz dziesięcioma tranzycjami mającymi wspólne miejsca wejściowe (*20x15x10*), *_01* oznacza numer przykładu, dla podanych parametrów.

Tabela 9. Zestawienie wyników dla sieci z różnych rozdziałów pracy

Sieć Petriego	***TSP***	***DL***	***CW***	***Losowo***
rys.93b rozdz.9 blokady	42	83,33%	119,05%	150,00%
rys.93b rozdz.9 pułapki	68	57,35%	106,00%	103,00%
rys.26a rozdz.5 blokady	25	80,00%	80,00%	96,00%
rys.26a rozdz.5 pułapki	26	76,92%	76,92%	96,00%
rys.26b rozdz.5 blokady	25	80,00%	80,00%	84,00%
rys.26b rozdz.5 pułapki	21	90,00%	90,00%	76,00%
rys.1 rozdz.1 blokady	40	67,50%	60,00%	88,00%
rys.1 rozdz.1 pułapki	52	51,92%	50,00%	58,00%
rys.52a rozdz.7 blokady	10	100,00%	90,00%	100,00%
rys.52a rozdz.7 pułapki	10	100,00%	90,00%	90,00%
rys.62 rozdz.7 blokady	57	82,46%	78,95%	204,00%
rys.62 rozdz.7 pułapki	158	47,47%	34,18%	77,00%
ŚREDNIA		***76,41%***	***79,59%***	***101,83%***

gdzie:

DL – sortowanie według długości klauzul (od najkrótszej do najdłuższej);

CW – sortowanie według częstości występowania poszczególnych literałów w formule (klauzule, w których występują literały często występujące w całej formule, ustawione są na początku formuły);

Losowo – losowe uporządkowanie klauzul;

TSP – rozmiar drzewa dla uporządkowania klauzul wygenerowanych zgodnie z kolejnością tranzycji.

Podobnie, jak w przypadku przykładów sieci Petriego z pracy, uporządkowanie klauzul dla losowo wygenerowanych sieci, najlepsze jest gdy klauzule sortowano według długości.

Na podstawie przeprowadzonych badań, nie uzyskano jednoznacznej odpowiedzi, które z kryterium należy wybrać dla podanej formuł. Dlatego wprowadzono podejście heurystyczne, które polega na oszacowaniu rozmiaru drzewa dla poszczególnego rodzaju sortowania, a następnie wyborze odpowiedniego sortowania. Dzięki takiemu podejściu możliwe jest zredukowanie rozmiaru drzewa, czego skutkiem jest znaczne skrócenie czasu obliczeń i zmniejszenie zapotrzebowania na wielkość pamięci operacyjnej.

Tabela 10. Zestawienie wyników dla sieci wygenerowanych losowo

Sieć Petriego	*TSP*	*DL*	*CW*	*Losowo*
20x15x10_01	10972	81%	92%	173%
20x15x10_02	6263	79%	45%	222%
20x15x10_03	16010	50%	39%	103%
20x18x15_01	3332	114%	89%	140%
20x18x15_02	765	182%	153%	2199%
20x18x15_03	700	144%	111%	1984%
20x18x15_04	1101	101%	115%	734%
20x18x15_05	1021	111%	105%	997%
20x20x15_01	1919	110%	118%	415%
20x20x15_02	1294	33%	68%	778%
20x20x15_03	770	136%	118%	639%
25x20x15_01	5836	41%	63%	358%
25x20x15_02	34272	84%	107%	385%
25x20x15_03	15356	65%	74%	311%
25x20x15_04	1919	67%	164%	149%
35x20x15	1702508	44%	48%	215%
ŚREDNIA		***90,13%***	***94,31%***	***612,63%***

7.11. Zastosowania opracowanego algorytmu w projektowaniu sterowników

Dekompozycja automatu współbieżnego na sekwencyjne automaty składowe wykorzystywana jest często w celu ułatwienia procesu syntezy układu cyfrowego

[ADAMSKI90], [LI,WOODSIDE95]. W przypadku zamodelowania takiego układu siecią Petriego, problem dekompozycji sprowadza się do wyodrębnienia podsieci typu automatowego, czyli podsieci zawierających tylko jeden znacznik.

W celu wyjaśnienia metody dekompozycji sieci na podsieci automatowe, poniżej zostaną podane podstawowe pojęcia [BANASZAK,ET.AL.93].

Dwa różne miejsca p_i i p_j nazywa się współbieżnymi, jeżeli istnieje takie oznakowanie tej sieci, w którym p_i oraz p_j są oznakowane równocześnie. Dwa różne miejsca p_i i p_j nazywa się niewspółbieżnymi (sekwencyjnymi), jeżeli p_i oraz p_j nigdy nie są oznakowane równocześnie.

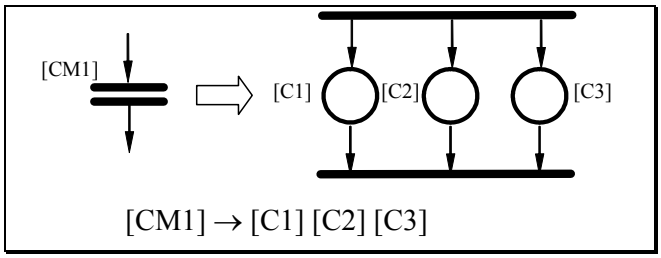
W jednej z metod, opisanych między innymi w pracy [BANASZAK,ET.AL.93], w celu dekompozycji sieci na składowe automatowe, należy sieć kolorować. Kolorowanie polega na przypisaniu miejscom sieci minimalnej liczby kolorów w taki sposób, aby dwa wierzchołki współbieżne miały zawsze różne kolory. Kolorowanie sieci w taki sposób przedstawia się jako przypisanie miejscom określonego koloru zgodnie z następującymi zasadami, uzupełnionymi w pracach doktorskich [WĘGRZYN98], [BILIŃSKI96]:

- każda tranzycja i miejsce muszą mieć co najmniej jeden kolor;
- jeżeli miejsce ma dany kolor, to każda tranzycja wejściowa i wyjściowa ma taki sam kolor;
- miejsca wejściowe każdej tranzycji muszą mieć różne kolory;
- miejsca wyjściowe każdej tranzycji muszą mieć różne kolory;
- zbiory kolorów wejściowych i wyjściowych dla danej tranzycji są sobie równe;
- nie istnieją dwa (lub więcej) miejsca o początkowym znakowaniu tym samym kolorem;
- każdy kolor musi mieć swój znacznik, tzn. liczba kolorów dla oznakowania jest równa całkowitej liczbie kolorów.

Pokolorowana sieć Petriego, oprócz informacji o współbieżności i sekwencyjności zdarzeń, niesie informację o przynależności miejsc do poszczególnych procesów sekwencyjnych [WĘGRZYN98]. Każda składowa automatowa reprezentowana jest przez P-niezmiennik.

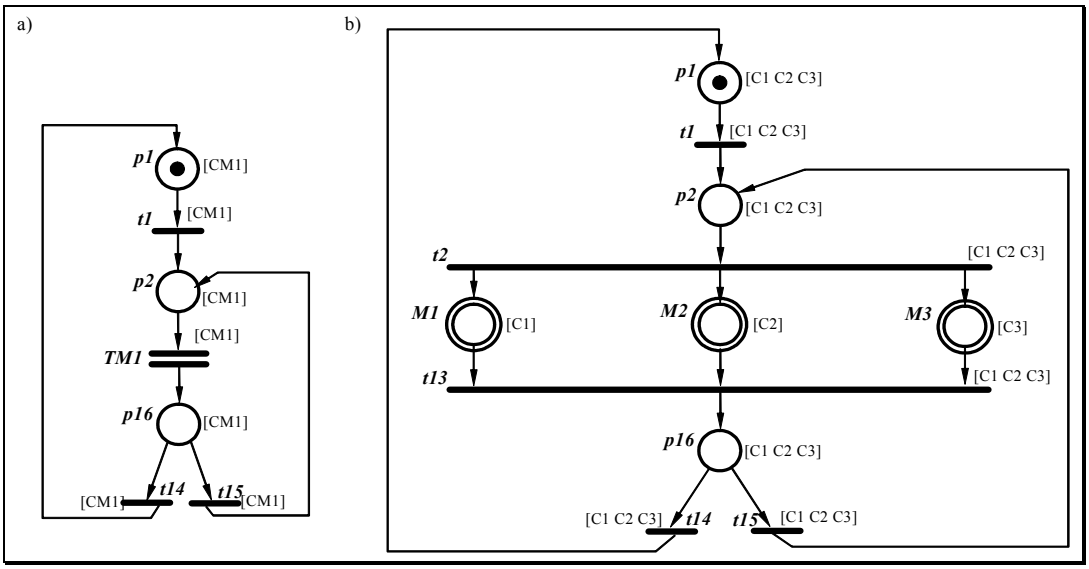
Na podstawie opracowanego algorytmu opisanego w rozdziale 7, generowane są P-niezmienniki sieci. Na ich podstawie można dokonać dekompozycji sieci Petriego na składowe automatowe. W tym celu należy pokolorować każdy z automatów oddzielnym kolorem, jak zostało to opisane powyżej. Liczba kolorów, jakimi należy pokolorować sieć zależy od liczby wyznaczonych P-niezmienników sieci. W przypadku kolorowania sieci hierarchicznej nie można przewidzieć liczby gałęzi równoległych, które zostały zredukowane w trakcie tworzenia sieci hierarchicznej. W związku z tym liczba kolorów

dla sieci płaskiej nie wynika jednoznacznie z liczby P-niezmienników dla sieci hierarchicznej. Zależy natomiast od liczby równoległych gałęzi w sieci płaskiej (Rys. 80). Przykładowo, kolor [CM1] makromiejsca reprezentowany jest przez trzy kolory [C1 C2 C3] w rozwinięciu tego makromiejsca.



Rys. 80. Rozwinięcie kolorów

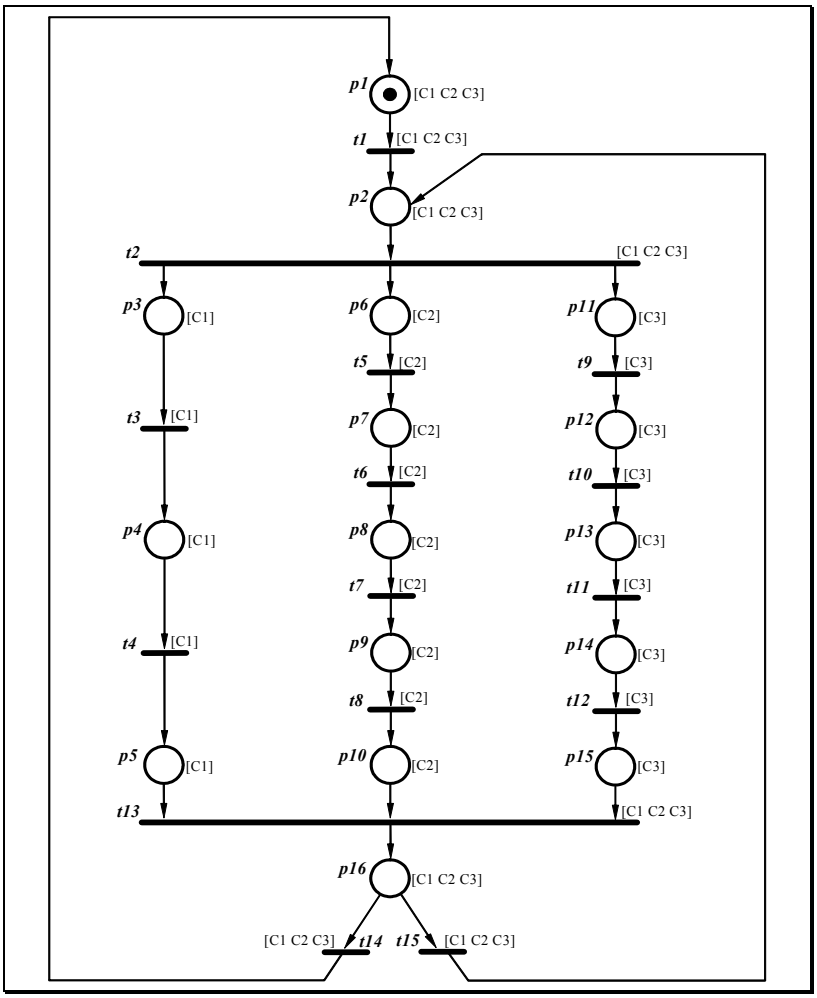
Dla przykładów z rozdziału 7.6 i rozdziału 7.7, możliwe jest pokolorowanie jedynie sieci z Rys. 52. Sieć ta została pokolorowana jednym kolorem [CM1] (Rys. 81a), jednakże kolor ten należy traktować jako potrójny (z trzema odcieniami [C1], [C2], [C3]), gdyż w przypadku rozwinięcia makrotranzycji, otrzymywane są trzy miejsca równoległe.



Rys. 81. P-kolorowana sieć Petriego na poszczególnych poziomach hierarchii

Na Rys. 81b została przedstawiona sieć Petriego, otrzymana po rozwinięciu makrotranzycji *TM1*. Sieć została pokolorowana dodatkowymi kolorami [2], [3].

Natomiast na Rys. 82 przedstawiono płaską, pokolorowaną sieć Petriego.



Rys. 82. Kolorowanie płaskiej sieci Petriego

8. Weryfikacja eksperymentalna symbolicznej metody analizy sieci Petriego

Sieci typu P/T są podzbiorem kolorowanych sieci Petriego, z tego względu możliwe jest wykorzystanie bardziej uniwersalnego narzędzia do analizy kolorowanych sieci Petriego, w celu weryfikacji algorytmu analizy, opisanego w niniejszej pracy.

W rozdziale tym zostanie zaprezentowany sposób weryfikacji, opracowanej przez autorkę, symbolicznej metody analizy sieci Petriego, z wykorzystaniem systemu Design/CPN, do modelowania i analizy kolorowanych sieci Petriego. W celu wykorzystania tego systemu do modelowania interpretowanych sieci Petriego opisanych w rozdziale 4.1, należało zaproponować sposób ich opisu w systemie Design/CPN (rozdział 8.1). Natomiast do weryfikacji symbolicznej metody analizy, wykorzystany został moduł tego systemu do analizy sieci Petriego na podstawie grafu znakowań (rozdział 8.2).

System Design/CPN posiada możliwość zapisu sieci Petriego z wykorzystaniem języka XML. Taka forma zapisu pozwala na wymianę danych pomiędzy opracowanym formatem PNSF3 a formatem XML w systemie Design/CPN. Dodatkowo, ze względu na opracowaną metodę automatycznego P-kolorowania sieci Petriego, sieci takie w sposób bezpośredni można przenieść do systemu Design/CPN. Kolejną zaletą jest fakt, opracowywania standardu zapisu sieci Petriego w języku XML. Norma takiego zapisu sieci będzie obejmowała również system Design/CPN, dzięki czemu po zastosowaniu takiego formatu również w opracowanym systemie eksperymentalnym (rozdział 9), będzie możliwość bezpośredniej wymiany danych pomiędzy obydwoma systemami.

System Design/CPN jest oprogramowaniem opracowanym przez firmę Meta Software Corporation przy współpracy grupy naukowców z Uniwersytetu w Aarhus [JENSEN92], [JENSEN94], [JENSEN97]. Jest to interaktywne narzędzie składające się z trzech części:

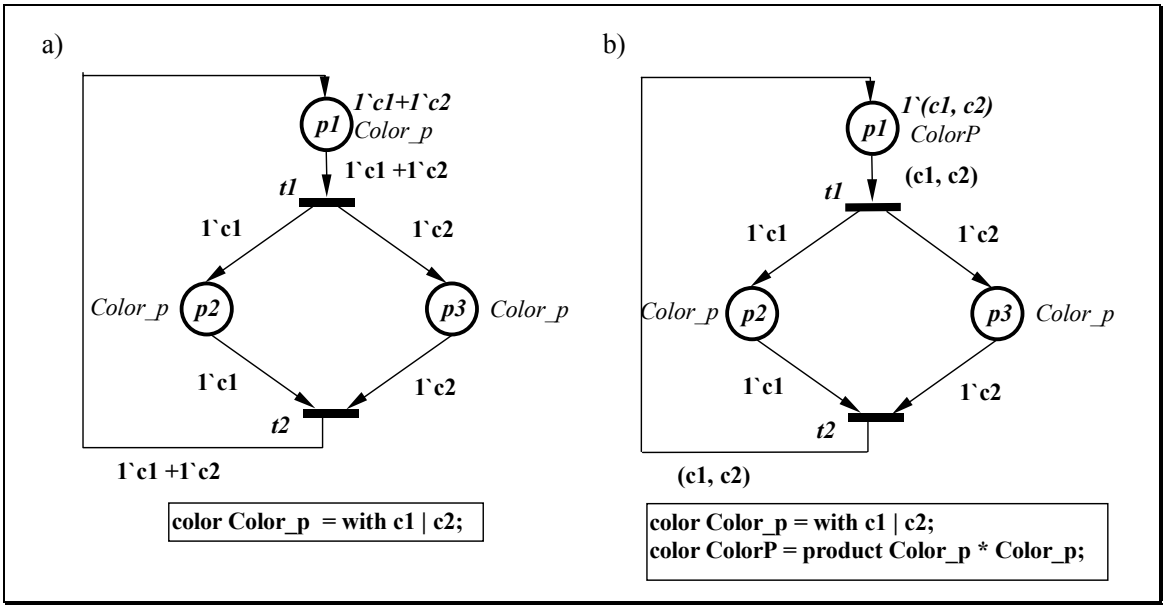
- edytora do rysowania i sprawdzania poprawności modelu kolorowanej sieci Petriego,
- symulatora do interaktywnej lub automatycznej symulacji modelu kolorowanej sieci,
- narzędzia do analizy – konstruowania i analizy grafu znakowań dla kolorowanej sieci.

Wszystkie trzy części pakietu mogą być wykorzystywane dla płaskich, hierarchicznych i czasowych sieci Petriego. Design/CPN wykorzystuje zmodyfikowany język ML (zwany CPN ML) do oprogramowania atrybutów węzłów i łuków w sieci Petriego.

8.1. Metodologia modelowania sterowników cyfrowych w systemie Design/CPN

System Design/CPN można wykorzystać do realizacji kilku sposobów modelowania sterowników logicznych [WĘGRZYN,KARATKIEVICH00] [WĘGRZYN,WĘGRZYN98]. W tym rozdziale zostaną przedstawione tylko cztery z nich. Metody zostaną pokazane na prostych sieciach, tylko w celu zademonstrowania mechanizmu przepływu znaczników wraz z odpowiednią interpretacją kolorów.

W pierwszej metodzie (Rys. 83a) znacznik początkowy jest przedstawiony jako suma dwóch znaczników, *1`c1+1`c2*, które po realizacji tranzycji *t1* generują znakowanie dla miejsc *p1* i *p2*, odpowiednio *1`c1* i *1`c2*. Następnie po realizacji tranzycji *t2*, oba znaczniki łączą się, tak jak przy miejscu *p1*.



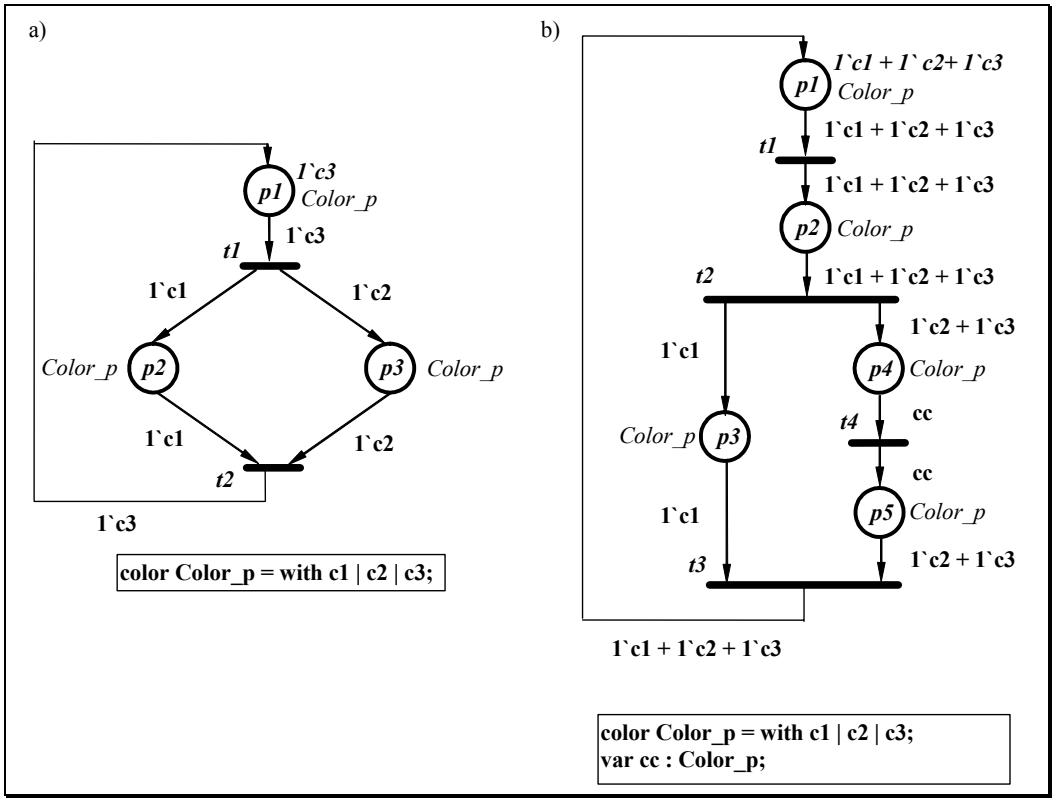
Rys. 83. a) Pierwsza, b) druga metoda modelowania w CPN

W drugiej metodzie (Rys. 83b) wykorzystywana jest deklaracja typu złożonego (ang. *tuple*). Typ złożony tworzy się jako produkt kartezjański dwóch zbiorów kolorów, np.: *ColorP = product Color_p * Color_p.* Znakowaniem początkowym dla miejsca *p1* jest element złożony, *1`(c1, c2)*. Po realizacji tranzycji *t1*, miejsca *p2* i *p3* otrzymują

odpowiednio znakowanie *1`c1* i *1`c2*. Po realizacji tranzycji *t2* znaczniki łączą się w znakowanie *1`(c1,c2)*.

W obu metodach kolor może być reprezentowany (interpretowany) jako proces sekwencyjny - każdy kolor reprezentuje jeden proces. Tak zamodelowana sieć może być dekomponowana na automaty sekwencyjne. Kolory obserwowane w czasie symulacji lub przeglądania wyników analizy ułatwiają śledzenie poszczególnych procesów sekwencyjnych w modelowanym układzie.

W trzeciej metodzie (Rys. 84a) w różnych podsieciach wykorzystywane są różne kolory charakteryzujące te podsieci. Znacznikiem początkowym jest *1`c3*. Po realizacji tranzycji *t1* generowane są znaczniki o kolorach: *1`c1* i *1`c2* odpowiednio dla miejsc *p2* i *p3*. Następnie po realizacji tranzycji *t2* generowany jest ponownie znacznik *1`c3*.

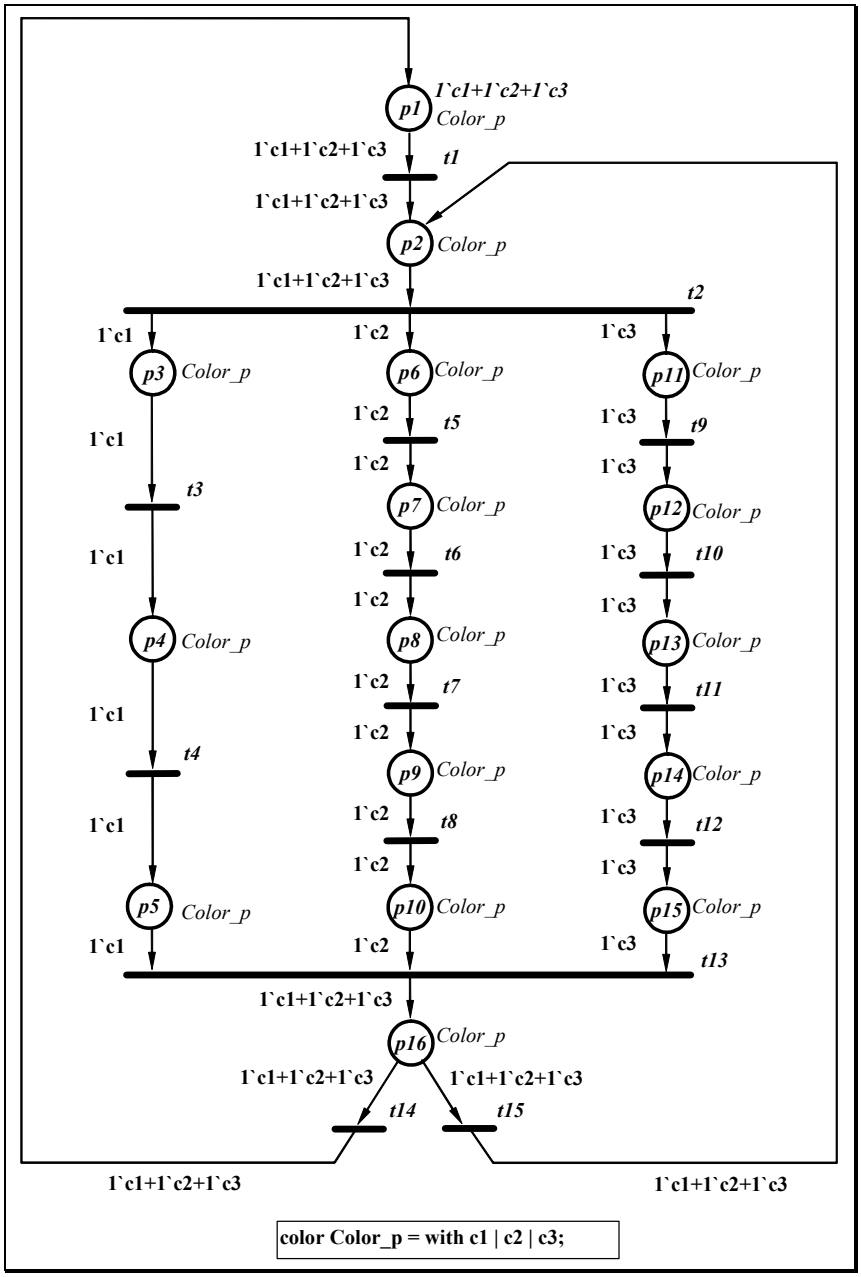


Rys. 84. a) Trzecia, b) czwarta metoda modelowania w CPN

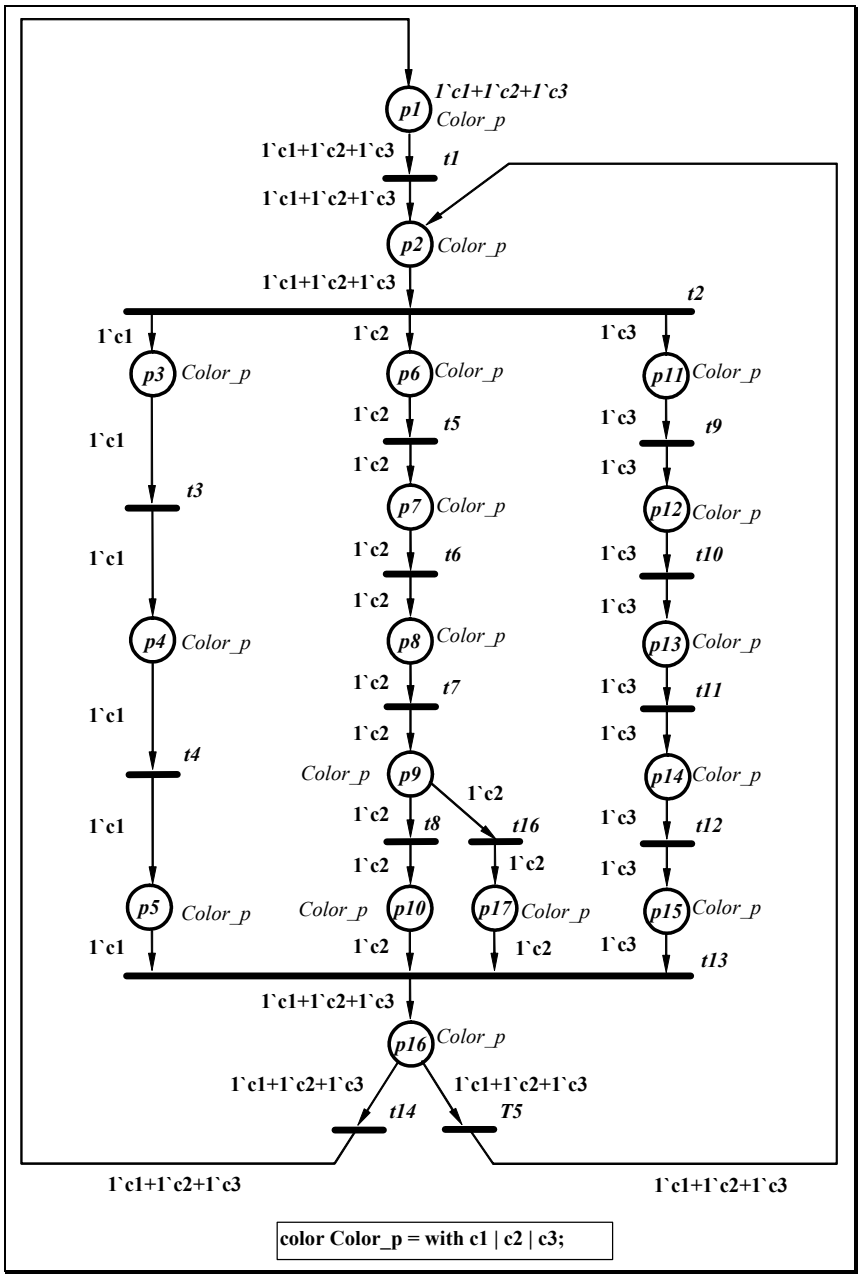
Ostatni prezentowany sposób modelowania wykorzystuje zmienne do reprezentacji kolorów, jakie mogą przechodzić przez dany łuk, np. zmienna *cc* (Rys. 84b) `[WĘGRZYN,WĘGRZYN98]`. Kolor miejsca jest równy kolorowi jego łuków wejściowych i wyjściowych. Znakowaniem początkowym jest suma kolorów *1`c1 + 1`c2 + 1`c3*. Po realizacji tranzycji *t2* miejsce *p3* otrzymuje znakowanie *1`c1* natomiast miejsce *p4* znakowanie *1`c2+ 1`c3*. Łuki pomiędzy miejscami *p4* a *p5* opisane są za pomocą

zmiennej *cc*, która przybiera jedną z wartości *1`c2* albo *1`c3*. Aby zrealizować tranzycję *t3* znaczniki *1`c1* oraz *1`c2 + 1`c3* muszą się znajdować odpowiednio w miejscach *p3* i *p5*.

Na Rys. 85 i Rys. 86 pokazano kolorowane sieć Petriego, odpowiednio żywą i nieżywą, z rozdziału 7, w notacji programu Design/CPN. Sieci te zostały zamodelowane zgodnie z pierwszą metodą. Ta metoda modelowania kontrolerów cyfrowych ilustruje ważną cechę kolorowanych sieci Petriego – zwięzłość. Jeżeli kilka procesów ma taką samą reprezentację w sieci Petriego, czyli przedstawiane są tą samą podsiecią, to zamiast kilkukrotnego występowania takich samych gałęzi w sieci, można wykorzystać tylko jedną, z tyloma kolorami, ile procesów ma ona reprezentować [WĘGRZYN,WĘGRZYN98], [WĘGRZYN98].



Rys. 85. Żywa kolorowana sieć Petriego

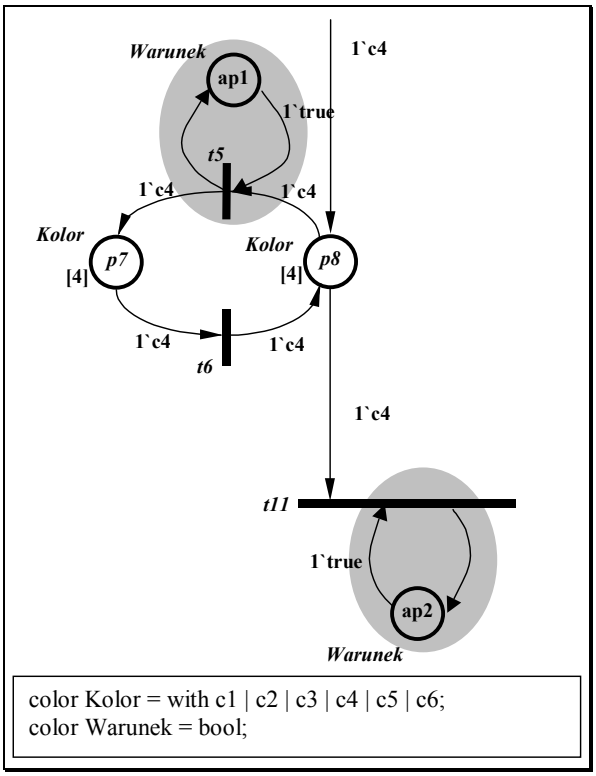


Rys. 86. Nieżywa kolorowana sieć Petriego

Konflikt

Sterowniki cyfrowe muszą być układami deterministycznymi, czyli powinny być jednoznacznie określone nie tylko wszystkie planowane zmiany stanów w układzie, ale również ich uwarunkowania. Zmiana stanów nie może przebiegać w sposób losowy. Z tego względu wykorzystuje się interpretowane sieci Petriego, w których przy tranzycji wypisany jest warunek przejścia z jednego stanu lokalnego do drugiego. W miejscu *p8* (Rys. 87) występuje konflikt (tranzycje *t5* i *t11*). W rzeczywistym układzie sterownika konflikt jest rozwiązany poprzez dodanie warunków przejść przy odpowiednich tranzycjach (interpretacja w sieci Petriego). Dlatego wykrywanie potencjalnych konfliktów ma duże znaczenie praktyczne.

W systemie Design/CPN nie ma możliwości wprowadzenia dodatkowych warunków przy tranzycjach w formie etykiet (predykatów) w zależności od czynników zewnętrznych. Istnieje tutaj możliwość wprowadzenia warunków jedynie w zależności od kolorów. Proponowanym sposobem rozwiązania tego konfliktu jest więc wprowadzenie dodatkowych miejsc rozwiązujących konflikt przy odpowiednich tranzycjach, należących do rozpatrywanego konfliktu. Dodane miejsca *ap1* i *ap2* (Rys. 87) otrzymują sztuczne znakowanie początkowe wówczas, gdy projektant podczas symulacji stwierdzi, że powinna zostać zrealizowana dana tranzycja. Gdy znacznik nie zostanie „wstrzyknięty" do wybranego jednego z tych miejsc to, wówczas układ znajdzie się w stanie zastoju.



Rys. 87. Fragment sieci z Rys. 85 z dodatkowymi miejscami ap1 i ap2

Hierarchia

W Design/CPN są dwie możliwości tworzenia hierarchicznej kolorowanej sieci Petriego: zastępczej tranzycji (ang. *substitution transition*) oraz fuzji miejsc (ang. *fusion place*) [JENSEN92], [JENSEN94], [JENSEN97].

Fuzja miejsc podobna jest do zmiennych globalnych w programie komputerowym. Tak jak w programie mogą występować odwołania do zmiennej globalnej i przypisywanie nowej wartości do zmiennej globalnej z różnych części programu, także zbiór fuzji może być wykorzystywany do zmiany znakowania tego samego miejsca z różnych miejsc sieci. Wszystkie miejsca w zbiorze fuzji są jednym miejscem, a łuki dochodzące lub wychodzące

z niego są sumą mnogościową łuków dochodzących lub wychodzących z miejsc należących do fuzji.

Fuzja miejsc może być więc używana w wielu przypadkach do uproszczenia oraz uogólnienia sieci. Na przykład sieć może reprezentować wiele różnych akcji, które wykorzystują te same zasoby.

Zastępcza tranzycja reprezentuje fragment sieci rozpoczętej i zakończonej tranzycjami. Zastępcza tranzycja podobna jest do podprogramu w programie komputerowym. Efekt nie jest identyczny, gdyż zastępcza tranzycja realizowana jest fizycznie. Nie jest wywoływana funkcja, ale w miejscu zastępczej tranzycji podstawiany jest wycinek sieci, który reprezentowany jest przez daną tranzycję zastępczą. Zastępcza tranzycja jest analogiczna do makrotranzycji.

8.2. Metody analizy sieci Petriego z wykorzystaniem systemu Design/CPN

W systemie Design/CPN są stosowane trzy sposoby analizy kolorowanej sieci Petriego:

- symulacja,
- graf znakowań,
- P-niezmienniki.

Symulacja jest podobna do sprawdzania programu (ang. *debugging*). Podczas symulacji możliwe jest sprawdzenie, czy zamodelowany układ działa poprawnie. Do tego celu może być wykorzystany mechanizm punktów zatrzymania (ang. *breakpoint*). Metoda ta pozwala na wczesne wykrycie i usunięcie błędów w modelu. Podczas symulacji możliwe jest również wygenerowanie raportu z symulacji. W raporcie z symulacji przedstawione są po kolei wszystkie tranzycje, które zostały zrealizowane podczas przeprowadzonej symulacji.

Druga metoda analizy bazuje na grafie znakowań. Graf znakowań tworzony jest automatycznie na podstawie sieci Petriego. Za pomocą tej metody sprawdzane są własności sieci Petriego takie jak:

- żywotność,
- ograniczoność,
- martwe znakowania i tranzycje.

Sieci Petriego z Rys. 85 i Rys. 86 analizowano z wykorzystaniem grafu znakowań. Na podstawie otrzymanego raportu z programu Design/CPN, można stwierdzić, że badana sieci Petriego z Rys. 85 jest żywa, natomiast sieć Petriego z Rys. 86 nie jest żywa, gdyż

znalezione zostały martwe znakowania sieci `{p5,p10,p15}` i `{p5,p15,p17}` oraz martwe tranzycje `t13`, `t14` i `t15`.

System Design/CPN nie jest systemem dedykowanym dla analizy interpretowanych sieci Petriego. Z tego względu nie może on być bezpośrednio wykorzystywany do specyfikacji kontrolerów współbieżnych [ABOUAISSA,ET.AL.95]. W rozdziale 8.1 zaproponowano i opisano kilka metod modelowania kontrolerów współbieżnych w tym systemie.

9. System PeNCAD w implementacji sterowników cyfrowych

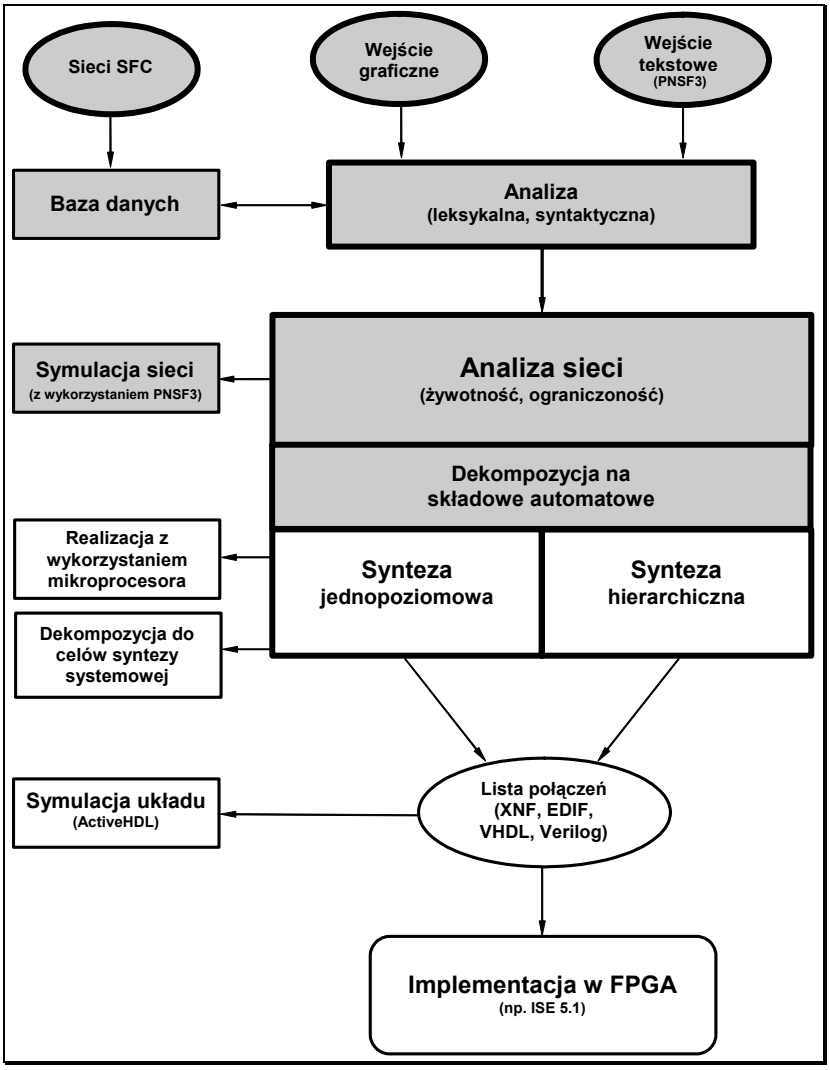
9.1. System PeNCAD

System PeNCAD służy do modelowania, weryfikacji oraz syntezy sterowników cyfrowych opisanych interpretowanymi sieciami Petriego. W systemie tym interpretowana sieć Petriego jest opisywana w formacie tekstowym, w sposób regułowy w języku PNSF3. Możliwe jest również przygotowanie sieci w edytorze graficznym. Istnieje również możliwość wprowadzenia do systemu sieci SFC, która następnie jest transformowana do sieci Petriego. Następnie sieć Petriego poddana jest symulacji oraz analizie bazującej na zaproponowanym w pracy algorytmie. Następnie sieć jest automatycznie syntezowana z wykorzystaniem konkretnej technologii i podana w formacie akceptowanym przez standardowe systemy CAD, tzn. w postaci modelu w języku opisu sprzętu (np. VHDL lub Verilog) lub listy połączeń (np. w formacie XNF). W wykorzystywanej metodzie do celów implementacyjnych stosowane są układy FPGA firmy Xilinx.

Implementacja układów sterowania, opisanych sieciami Petriego, znalazła wiele różnych form realizacji [ANDRZEJEWSKI02], [MACHADO,ET.AL.97], [PARDEY,BOLTON91], [STEWART,ET.AL.91], [WĘGRZYN98]. Jednakże w literaturze przedmiotu specyficzne cechy bazy elementowej traktowane były drugorzędnie, z tego też względu układowe realizacje sieci Petriego dla elementów FPGA były rozpatrywane w oderwaniu od konkretnej realizacji na poziomie wyrażeń logicznych. Ostateczna implementacja układu została pozostawiona narzędziom CAD. W systemie PeNCAD po raz pierwszy wykorzystano metodę bezpośredniego odwzorowania sieci w strukturach FPGA [WĘGRZYN98] bez pośrednictwa złożonych systemów CAD, uniwersyteckich (np. SIS) lub komercyjnych (np. FPGA Express firmy Synopsys) [WOLAŃSKI98]. Ponadto opracowano metodę programowej realizacji z wykorzystaniem systemów mikroprocesorowych [ANDRZEJEWSKI02]. Jednocześnie wprowadzono algorytmy dekompozycji układów cyfrowych na sieci Petriego dla potrzeb syntezy systemowej [SKOWROŃSKI00]. Realizacja algorytmów sterowania z wykorzystaniem układów SPLD i CPLD narzuciła konieczność dekompozycji sieci na składowe automatowe [WĘGRZYN96]. Nowsze metody zostały opracowane w ramach prezentowanej pracy. Dodatkowo, na uwagę zasługuje fakt

hierarchicznej struktury, zarówno specyfikacji, jak i otrzymanej realizacji układowej. Najbardziej zaawansowane systemy CAD, np. Active-HDL firmy ALDEC, pozwalają na zintegrowanie rezultatów pracy z nowoczesną techniką projektowania systemów cyfrowych w środowisku VHDL i Verilog.

Ogólny schemat systemu PeNCAD, z zacieniowanymi elementami systemu PN-XML, które są przedmiotem niniejszej pracy, pokazano na rysunku (Rys. 88).



Rys. 88. Schemat systemu PeNCAD

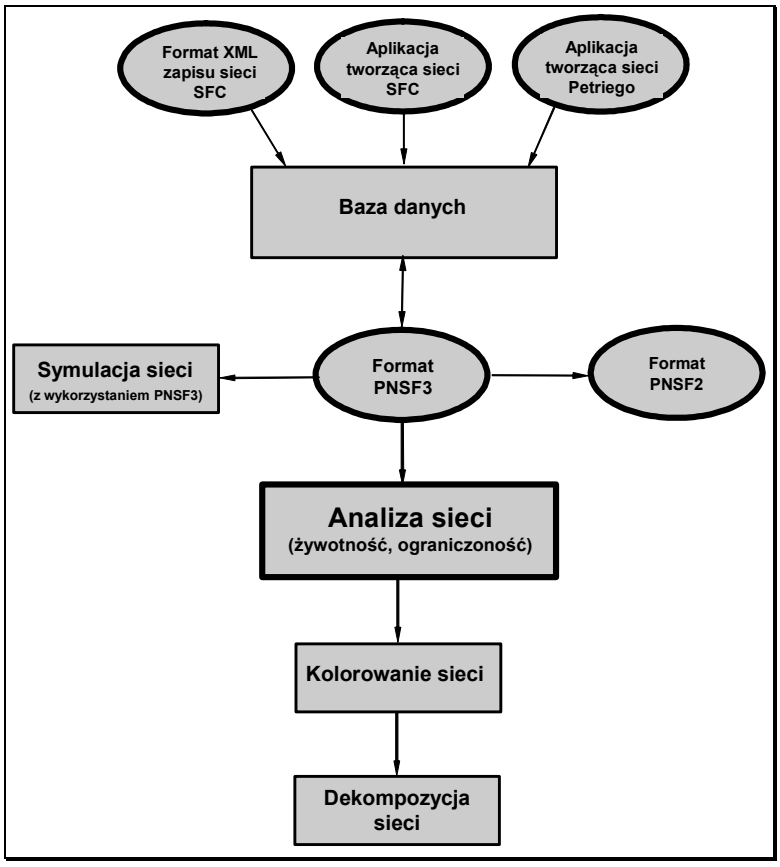
9.2. Opis systemu PN-XML

System *PN-XML* służy docelowo do modelowania, weryfikacji układów sterowania binarnego, opisanego sieciami Petriego oraz sieciami SFC. Dane do systemu mogą być wprowadzane w formie tekstowej, jako format XML dostosowany do poszczególnych metod modelowania lub w formie graficznej. Następnie dane wprowadzane są do bazy danych, skąd można wygenerować poszczególne formaty opisu

sieci Petriego i sieci SFC w języku XML. Istnieje również możliwość wygenerowania formatów opisu sieci Petriego oraz sprecyzowanego podzbioru sieci SFC w języku SVG, w celu symulacji sieci. Kolejną opcją systemu jest bezpośrednia transformacja między poszczególnymi formatami XML opisu sterowników [WĘGRZYN02A]. Ta opcja umożliwia analizę sieci SFC z wykorzystaniem sieci Petriego.

Analiza układów sterowania odbywa się w dwóch kolejnych etapach. Pierwszym krokiem jest symulacja sterownika opisanego sieciami Petriego oraz sieciami SFC. Zasadniczym celem tego etapu jest sprawdzenie zgodności modelu ze specyfikacją (wymaganiami). Dodatkowo wykrywane są niektóre błędy, np. zastoje. Drugą czynnością jest weryfikacja pod względem formalnym, sterownika opisanego sieciami Petriego z wykorzystaniem opracowanej w niniejszej pracy, metody. Rys. 89 przedstawia ogólny schemat systemu. Poszczególne elementy systemu zostały dokładnie opisane w rozdziałach dotyczących analizy sieci Petriego (rozdział 7) i sposobów modelowania sieci w języku XML (rozdział 6).

Dzięki zastosowaniu języka XML w opisie sterowników, istnieje możliwość wymiany danych pomiędzy systemem *PN-XML* a podobnymi systemami uniwersyteckimi, np. *Design/CPN*, czy *CPNTools* z Uniwersytetu w Aarchus (Dania).



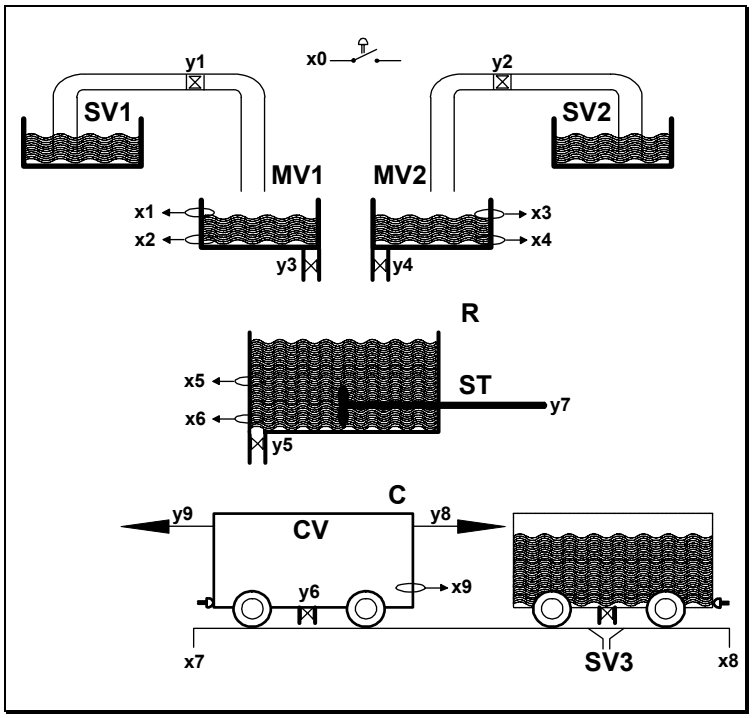
Rys. 89. Schemat przepływu danych w systemie PN-XML

9.3. Analiza specyfikacji sterownika

W rozdziale tym zostaną przedstawione szczegółowo metody modelowania, transformacji oraz weryfikacji sterownika logicznego. Przedstawiony zostanie przykład procesu mieszania i transportu cieczy. Przykład został zaczerpnięty z [ADAMSKI91] i jest często wykorzystywany przez innych autorów. Celem przykładu jest pokazanie proponowanego sposobu transformacji oraz weryfikacji.

9.4. Opis systemu sterowania procesem mieszania i transportu cieczy

System sterowania procesem mieszania i transportu cieczy (Rys. 90) rozpoczyna działanie po naciśnięciu przycisku startowego *x0*. Wówczas zawory *y1* i *y2* zostają otwarte i napełniane są zbiorniki *MV1* i *MV2*, odpowiednio do poziomu *x1* i *x3*. Sygnał *x0* powoduje także, że wózek *C* ze zbiornikiem *CV* zostaje przemieszczony do początkowej pozycji, co jest sygnalizowane sygnałem *x7*. Po równoczesnym zamknięciu zaworów *y1* i *y2*, zostają otwarte zawory *y3* i *y4* i reaktor *R* zostaje napełniony cieczą z wcześniej napełnionych zbiorników *MV1* (sygnał *x2*) i *MV2* (sygnał *x4*). Gdy ciecz podniesie się powyżej poziomu czujnika *x5*, włączane zostaje mieszadło *ST*. Opróżnienie zbiorników *MV1* i *MV2* powoduje zamknięcie odpowiednio zaworów *y3* i *y4*.



Rys. 90. Model rzeczywisty procesu mieszania i transportu cieczy

Dodatkowo, gdy wózek *C* znajdzie się w lewym skrajnym położeniu, otwiera się zawór *y5* i system przechodzi do fazy napełniania zbiornika *CV*. Kiedy reaktor jest pusty (sygnał *x6*), wówczas ciecz jest transportowana do zbiornika *SV3* wózkiem *C*. Kiedy cały cykl jest zakończony wówczas system czeka w stanie początkowym na ponowne aktywowanie sygnału *x0*.

Opis stanów lokalnych sterownika opisano w tabeli (Tabela 11). Natomiast opis wejść i wyjść opisano w tabeli (Tabela 12).

Tabela 11. Opis stanów lokalnych sterownika

Stan lokalny	*Opis*
p1	Stan początkowy
p2	Napełniania zbiornika *SV1*
p3	Napełniania zbiornika *SV2*
p4	Oczekiwanie na pełny zbiornik *SV1*
p5	Oczekiwanie na pełny zbiornik *SV2*
p6	Wózek *C* jedzie w lewo
p7	Mieszadło *ST* zostaje włączone
p8	Nieczynne mieszadło *ST*
p9	Napełnianie zbiornika *R* ze zbiornika *SV1*
p10	Napełnianie zbiornika *R* ze zbiornika *SV2*
p11	Oczekiwanie na opróżnienie zbiornika *SV1*
p12	Oczekiwanie na opróżnienie zbiornika *SV2*
p13	Oczekiwanie na napełnienie zbiornika *CV*
p14	Zbiornik *CV* napełniany jest ze zbiornika *R*
p15	Wózek *C* jedzie w prawo
p16	Zbiornik *CV* jest opróżniany

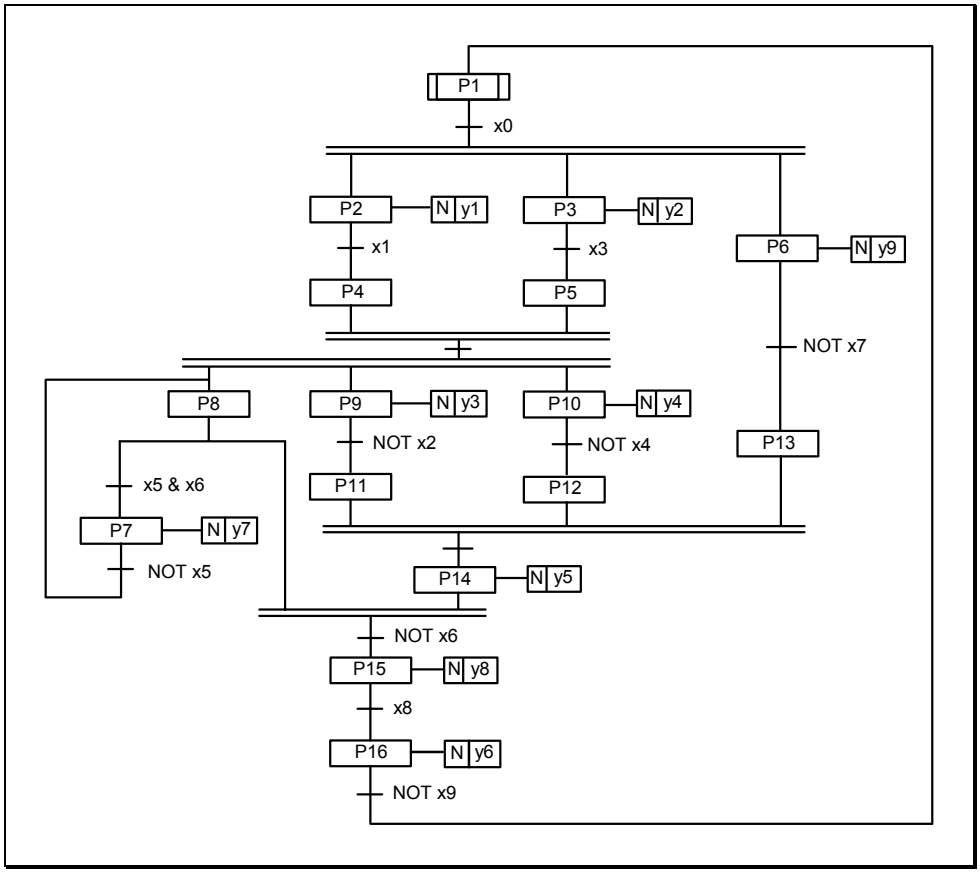
Tabela 12. Opis wejść i wyjść sterownika

We/wy	*Opis*
x0	Stan początkowy (start)
x1,x3	Max poziom płynu w *MV1* i *MV2*
x2,x4	Min poziom płynu w *MV1* i *MV2*
x1	Min poziom dla mieszadła *ST*
x6	Min poziom płynu w reaktorze *R*
x7,x8	Położenie początkowe i końcowe wagonu *C*
x9	Min poziom płynu w zbiorniku *CV*
y1,y2	Zawory *MV1* i *MV2* (napełnianie)
y3,y4	Zawory wyjściowe *MV1* i *MV2* (opróżnianie)
y5	Zawór reaktora *R*
y6	Zawór zbiornika *CV*
y7	Mieszadło *ST*
y8,y9	Ruch wagonu w lewo i w prawo

Sterownik zostanie przedstawiony w postaci modelu sieci SFC i sieci Petriego.

9.4.1. Opis sterownika siecią SFC

Na Rys. 91 został przedstawiony równoważny model w postaci sieci SFC, opisujący proces mieszania i transportu cieczy. Sieć SFC składa się z szesnastu kroków opisanych w tabeli (Tabela 11). Opis warunków, które muszą być spełnione, w celu realizacji tranzycji, oraz opis wyjść zamieszczony jest w tabeli (Tabela 12).



Rys. 91. Sieć SFC modelująca proces mieszania i transportu cieczy

Specyfikacja dokumentu opisującego strukturę sieci SFC opracowana została z wykorzystaniem języka znaczników XML. Nowy format został przygotowany dla hierarchicznej sieci SFC. Zawiera on zbiór reguł opisujących strukturę sieci SFC.

Dokument XML opisujący sieci SFC składa się z czterech podstawowych części:

```
<SFC>
    <INPUTS>        </INPUTS>
    <OUTPUTS>       </OUTPUTS>
    <MACRO_PART>    </MACRO_PART>
    <PART>          </PART>
</SFC>
```

Znacznik `<INPUTS>` opisuje listę sygnałów wejściowych układu sterowania i definiuje nazwy sygnałów. Znacznik `<OUTPUTS>` opisuje listę sygnałów wyjściowych. Znacznik `<MACRO_PART>` jest znacznikiem nadrzędnym i zawiera znaczniki opisujące

kroki, akcje z nimi związane, tranzycje oraz strukturę makrokroku. Część <PART> definiuje strukturę sieci nadrzędnej.

Na Rys. 92 i Rys. 93 zaprezentowano opis w formacie XML, przedstawiający sieć SFC z Rys. 91.

```
<?xml version="1.0" encoding="ISO-8859-2" standalone="no"?>
<!DOCTYPE SFC SYSTEM "SFC.dtd">
<SFC>
<INPUTS> <INPUT ID=”in0”>x0</INPUT>
      <INPUT ID=”in1”>x1</INPUT>
      <INPUT ID=”in2”>x2</INPUT>
      <INPUT ID=”in3”>x3</INPUT>
      <INPUT ID=”in4”>x4</INPUT>
      <INPUT ID=”in5”>x5</INPUT>
      <INPUT ID=”in6”>x6</INPUT>
      <INPUT ID=”in7”>x7</INPUT>
      <INPUT ID=”in8”>x8</INPUT>
      <INPUT ID=”in9”>x9</INPUT> </INPUTS>
<OUTPUTS> <OUTPUT ID=”out1” TYPE=”N”>y1</OUTPUT>
      <OUTPUT ID=”out2” TYPE=”N”>y2</OUTPUT>
      <OUTPUT ID=”out3” TYPE=”N”>y3</OUTPUT>
      <OUTPUT ID=”out4” TYPE=”N”>y4</OUTPUT>
      <OUTPUT ID=”out5” TYPE=”N”>y5</OUTPUT>
      <OUTPUT ID=”out6” TYPE=”N”>y6</OUTPUT>
      <OUTPUT ID=”out7” TYPE=”N”>y7</OUTPUT>
      <OUTPUT ID=”out8” TYPE=”N”>y8</OUTPUT>
      <OUTPUT ID=”out9” TYPE=”N”>y9</OUTPUT> </OUTPUTS>
<ACTIONS></ACTIONS>
<STEPS> <STEP ID="s1" INITIAL_STEP=”YES” NAME=”P1”></STEP>
      <STEP ID="s2" NAME=”P2” OUT=”out1”></STEP>
      <STEP ID="s3" NAME=”P3” OUT=”out2”></STEP>
      <STEP ID="s4" NAME=”P4”></STEP>
      <STEP ID="s5" NAME=”P5”></STEP>
      <STEP ID="s6" NAME=”P6” OUT=”out9”></STEP>
      <STEP ID="s7" NAME=”P7” OUT=”out7”></STEP>
      <STEP ID="s8" NAME=”P8”></STEP>
      <STEP ID="s9" NAME=”P9” OUT=”out3”></STEP>
      <STEP ID="s10" NAME=”P10” OUT=”out4”></STEP>
      <STEP ID="s11" NAME=”P11”></STEP>
      <STEP ID="s12" NAME=”P12”></STEP>
      <STEP ID="s13" NAME=”P13”></STEP>
      <STEP ID="s14" NAME=”P14” OUT=”out5”></STEP>
      <STEP ID="s15" NAME=”P15” OUT=”out8”></STEP>
      <STEP ID="s16" NAME=”P16” OUT=”out6”></STEP> </STEPS>
<TRANSITIONS>
      <TRANSITION ID="t1" NAME=”T1”>
            <OPERAND ID_O =”in0”></OPERAND>
      </TRANSITION>
      <TRANSITION ID="t2" NAME=”T2”>
            <OPERAND ID_O =”in1”></OPERAND>
      </TRANSITION>
      <TRANSITION ID="t3" NAME=”T3”>
            <OPERAND ID_O =”in3”></OPERAND>
      </TRANSITION>
      <TRANSITION ID="t4" NAME=”T4”>
      </TRANSITION>
      <TRANSITION ID="t5" NAME=”T5”>
            <OPERAND ID_O =”in5” NEXT_O=”AND”></OPERAND>
            <OPERAND ID_O =”in6”></OPERAND>
      </TRANSITION>
      <TRANSITION ID="t6" NAME=”T6”>
            <OPERAND ID_O =”in5” NOT=”YES”></OPERAND>
      </TRANSITION>
```

Rys. 92. Format XML opisu sieci SFC

```
      <TRANSITION ID="t7" NAME=”T7”>
            <OPERAND ID_O =”in2” NOT=”YES”></OPERAND>
      </TRANSITION>
      <TRANSITION ID="t8" NAME=”T8”>
            <OPERAND ID_O =”in4” NOT=”YES”></OPERAND>
      </TRANSITION>
      <TRANSITION ID="t9" NAME=”T9”>
            <OPERAND ID_O =”in7” NOT=”YES”></OPERAND>
      </TRANSITION>
      <TRANSITION ID="t10" NAME=”T10”>
      </TRANSITION>
      <TRANSITION ID="t11" NAME=”T11”>
            <OPERAND ID_O =”in6” NOT=”YES”></OPERAND>
      </TRANSITION>
      <TRANSITION ID="t12" NAME=”T12”>
            <OPERAND ID_O =”in8”></OPERAND>
      </TRANSITION>
      <TRANSITION ID="t13" NAME=”T13”>
            <OPERAND ID_O =”in9” NOT=”YES”></OPERAND>
      </TRANSITION>
</TRANSITIONS>
<NET>
      <ARC ID_TRANS="T1" IN_STEPS="s1" OUT_STEPS="s2 s3 s6"></ARC>
      <ARC ID_TRANS="T2" IN_STEPS="s2" OUT_STEPS="s4"></ARC>
      <ARC ID_TRANS="T3" IN_STEPS="s3" OUT_STEPS="s5"></ARC>
      <ARC ID_TRANS="T4" IN_STEPS="s4 s5" OUT_STEPS="s8 s9 s10"></ARC>
      <ARC ID_TRANS="T5" IN_STEPS="s8" OUT_STEPS="s7"></ARC>
      <ARC ID_TRANS="T6" IN_STEPS="s7" OUT_STEPS="s8"></ARC>
      <ARC ID_TRANS="T7" IN_STEPS="s9" OUT_STEPS="s11"></ARC>
      <ARC ID_TRANS="T8" IN_STEPS="s10" OUT_STEPS="s12"></ARC>
      <ARC ID_TRANS="T9" IN_STEPS="s6" OUT_STEPS="s13"></ARC>
      <ARC ID_TRANS="T10" IN_STEPS="s11 s12 s13" OUT_STEPS="s14"></ARC>
      <ARC ID_TRANS="T11" IN_STEPS="s14 s8" OUT_STEPS="s15"></ARC>
      <ARC ID_TRANS="T12" IN_STEPS="s15" OUT_STEPS="s16"></ARC>
      <ARC ID_TRANS="T13" IN_STEPS="s16" OUT_STEPS="s1"></ARC>
</NET>
<SFC>
```

Rys. 93. Format XML opisu sieci SFC (cd.)

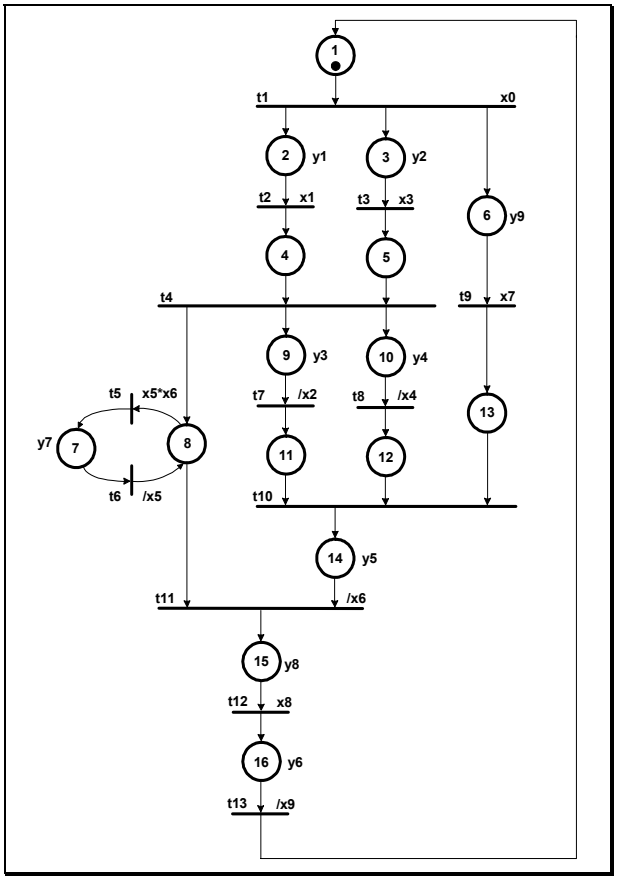
Dokument XML sprawdzany jest za pomocą definicji typu dokumentu (DTD), w którym określona jest dopuszczalna lista elementów, które mogą wystąpić w dokumencie XML, oraz ilość występowania poszczególnych elementów. Dokładny opis dokumentu XML opisującego sieci SFC oraz dokumentu DTD znajduje się w [WĘGRZYN01A].

9.4.2. Moduł transformacji sieci SFC na sieć Petriego

Metoda transformacji sieci SFC na sieć Petriego wykorzystuje bazy danych i język XML. Pierwszym etapem jest przygotowanie modelu sieci SFC w środowisku Oracle Forms oraz zapisanie tego modelu do bazy danych. Dane do bazy mogą być wprowadzone również z dokumentu XML opisującego sieci SFC. Następnie z wykorzystaniem oprogramowania Oracle Reports, na podstawie danych w bazie, generowany jest opis sieci Petriego w formacie PNSF3 po translacji sieci SFC. Istnieje również możliwość wygenerowania formatów XML i SVG opisujących sieci SFC. Format SVG wykorzystywany jest w symulacji sieci SFC.

9.4.3. Transformacja sieci SFC do sieci Petriego

Na Rys. 94 przedstawiono sieć Petriego, odpowiadającą sieci SFC, odpowiednio z Rys. 91. Natomiast Rys. 95 i Rys. 96 prezentuje format PNSF3 dla sieci z Rys. 94.



Rys. 94. Sieć Petriego odpowiadająca sieci SFC z Rys. 91

```
<?xml version="1.0" encoding="ISO-8859-2" standalone="no"?>
<!DOCTYPE PNSF3 SYSTEM "pnsf3.dtd">
<PNSF3>
<CLOCKS>    <CLOCK ID="clk1" > clk </CLOCK>    </CLOCKS>
<INPUTS>       <INPUT ID="i1" > x0 </INPUT>
      <INPUT ID="i2" > x1 </INPUT>
      <INPUT ID="i3" > x2 </INPUT>
      <INPUT ID="i4" > x3 </INPUT>
      <INPUT ID="i5" > x4 </INPUT>
      <INPUT ID="i6" > x5 </INPUT>
      <INPUT ID="i7" > x6 </INPUT>
      <INPUT ID="i8" > x7 </INPUT>
      <INPUT ID="i9" > x8 </INPUT>
      <INPUT ID="i10" > x9 </INPUT>     </INPUTS>
<OUTPUTS>      <OUTPUT ID="o1" > y1 </OUTPUT>
      <OUTPUT ID="o2" > y2 </OUTPUT>
      <OUTPUT ID="o3" > y3 </OUTPUT>
      <OUTPUT ID="o4" > y4 </OUTPUT>
      <OUTPUT ID="o5" > y5 </OUTPUT>
      <OUTPUT ID="o6" > y6 </OUTPUT>
      <OUTPUT ID="o7" > y7 </OUTPUT>
      <OUTPUT ID="o8" > y8 </OUTPUT>
      <OUTPUT ID="o9" > y9 </OUTPUT>   </OUTPUTS>
```

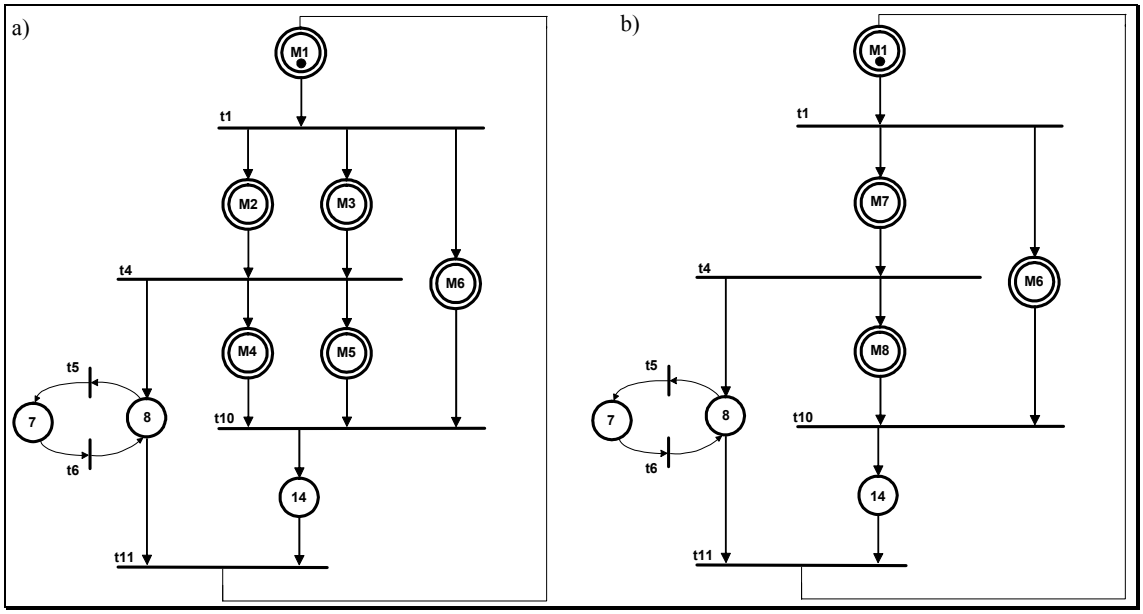
Rys. 95. Format PNSF3 dla sieci z Rys. 94

```
<REG_OUTPUTS>   <REG_OUTPUT ID="ro1" > y1 </REG_OUTPUT>
      <REG_OUTPUT ID="ro2" > y2 </REG_OUTPUT>
      <REG_OUTPUT ID="ro3" > y3 </REG_OUTPUT>
      <REG_OUTPUT ID="ro4" > y4 </REG_OUTPUT>
      <REG_OUTPUT ID="ro5" > y5 </REG_OUTPUT>
      <REG_OUTPUT ID="ro6" > y6 </REG_OUTPUT>
      <REG_OUTPUT ID="ro7" > y7 </REG_OUTPUT>
      <REG_OUTPUT ID="ro8" > y8 </REG_OUTPUT>
      <REG_OUTPUT ID="ro9" > y9 </REG_OUTPUT> </REG_OUTPUTS>
<PLACES>   <PLACE ID="p1" MARKING="yes" > p1 </PLACE>
    <PLACE ID="p2" > p2 </PLACE>
    <PLACE ID="p3" > p3 </PLACE>
    <PLACE ID="p4" > p4 </PLACE>
    <PLACE ID="p5" > p5 </PLACE>
    <PLACE ID="p6" > p6 </PLACE>
    <PLACE ID="p7" > p7 </PLACE>
    <PLACE ID="p8" > p8 </PLACE>
    <PLACE ID="p9" > p9 </PLACE>
    <PLACE ID="p10" > p10 </PLACE>
    <PLACE ID="p11" > p11 </PLACE>
    <PLACE ID="p12" > p12 </PLACE>
    <PLACE ID="p13" > p13 </PLACE>
    <PLACE ID="p14" > p14 </PLACE>
    <PLACE ID="p15" > p15 </PLACE>
    <PLACE ID="p16" > p16 </PLACE>   </PLACES>
<PREDICATES>     <PREDICATE ID="pred1">x5 * x6</PREDICATE>
    <PREDICATE ID="pred2">/x2</PREDICATE>
    <PREDICATE ID="pred3">/x4</PREDICATE>
    <PREDICATE ID="pred4">/x5</PREDICATE>
    <PREDICATE ID="pred5">/x6</PREDICATE>
    <PREDICATE ID="pred6">/x9</PREDICATE> PREDICATES>

<TRANSITIONS>  <TRANSITION ID="t1" ID_INPUTS="i1"> t1 </TRANSITION>
    <TRANSITION ID="t2" ID_INPUTS="i2"> t2 </TRANSITION>
    <TRANSITION ID="t3" ID_INPUTS="i4"> t3 </TRANSITION>
    <TRANSITION ID="t4" > t4 </TRANSITION>
    <TRANSITION ID="t5" ID_INPUTS="pred1"> t5 </TRANSITION>
    <TRANSITION ID="t6" ID_INPUTS="pred4"> t6 </TRANSITION>
    <TRANSITION ID="t7" ID_INPUTS="pred2"> t7 </TRANSITION>
    <TRANSITION ID="t8" ID_INPUTS="pred3"> t8 </TRANSITION>
    <TRANSITION ID="t9" ID_INPUTS="i8"> t9 </TRANSITION>
    <TRANSITION ID="t10" > t10 </TRANSITION>
    <TRANSITION ID="t11" ID_INPUTS="pred5"> t11 </TRANSITION>
    <TRANSITION ID="t12" ID_INPUTS="i9"> t12 </TRANSITION>
    <TRANSITION ID="t13" ID_INPUTS="pred6"> t13 </TRANSITION>  </TRANSITIONS>
<NET> <ARC ID_TRANSITION="t1" ID_IN_PLACES="p1" ID_OUT_PLACES="p2 p3 p6">
   </ARC>
   <ARC ID_TRANSITION="t2" ID_IN_PLACES="p2" ID_OUT_PLACES="p4"> </ARC>
   <ARC ID_TRANSITION="t3" ID_IN_PLACES="p3" ID_OUT_PLACES="p5"> </ARC>
   <ARC ID_TRANSITION="t4" ID_IN_PLACES="p4 p5" ID_OUT_PLACES="p8 p9 p10">
   </ARC>
   <ARC ID_TRANSITION="t5" ID_IN_PLACES="p8" ID_OUT_PLACES="p7"> </ARC>
   <ARC ID_TRANSITION="t6" ID_IN_PLACES="p7" ID_OUT_PLACES="p8"> </ARC>
   <ARC ID_TRANSITION="t7" ID_IN_PLACES="p9" ID_OUT_PLACES="p11"> </ARC>
   <ARC ID_TRANSITION="t8" ID_IN_PLACES="p10" ID_OUT_PLACES="p12"> </ARC>
   <ARC ID_TRANSITION="t9" ID_IN_PLACES="p6" ID_OUT_PLACES="p13"> </ARC>
   <ARC ID_TRANSITION="t10" ID_IN_PLACES="p11 p12 p13" ID_OUT_PLACES="p14">
   </ARC>
   <ARC ID_TRANSITION="t11" ID_IN_PLACES="p14 p8" ID_OUT_PLACES="p15">
   </ARC>
   <ARC ID_TRANSITION="t12" ID_IN_PLACES="p15" ID_OUT_PLACES="p16"> </ARC>
   <ARC ID_TRANSITION="t13" ID_IN_PLACES="p16" ID_OUT_PLACES="p1"> </ARC> </NET>
<MOORE_OUTPUTS> <MOORE_DESC ID_IN_PLACES="p2" ID_OUT_SIGNALS="o1" > </MOORE_DESC>
   <MOORE_DESC ID_IN_PLACES="p3" ID_OUT_SIGNALS="o2" > </MOORE_DESC>
   <MOORE_DESC ID_IN_PLACES="p6" ID_OUT_SIGNALS="o9" > </MOORE_DESC>
   <MOORE_DESC ID_IN_PLACES="p7" ID_OUT_SIGNALS="o7" > </MOORE_DESC>
   <MOORE_DESC ID_IN_PLACES="p9" ID_OUT_SIGNALS="o3" > </MOORE_DESC>
   <MOORE_DESC ID_IN_PLACES="p10" ID_OUT_SIGNALS="o4" > </MOORE_DESC>
   <MOORE_DESC ID_IN_PLACES="p14" ID_OUT_SIGNALS="o5" > </MOORE_DESC>
   <MOORE_DESC ID_IN_PLACES="p15" ID_OUT_SIGNALS="o8" > </MOORE_DESC>
   <MOORE_DESC ID_IN_PLACES="p16" ID_OUT_SIGNALS="o6" > </MOORE_DESC>
</MOORE_OUTPUTS>
</PNSF3>
```

Rys. 96. Format PNSF3 dla sieci z Rys. 94 (cd.)

9.4.4. Analiza sterownika

Pierwszym etapem analizy sterowników zgodnie z zaproponowaną metodą (rozdział 7), jest redukcja sieci. Na Rys. 97a i b przedstawione zostały kolejne etapy redukcji sieci. Miejscom *p1*, *p15* i *p16* odpowiada makromiejsce *M1*; miejscom *p2*, *p4* - makromiejsce *M2*; miejscom *p3* i *p5* - makromiejsce *M3*; miejscom *p9* i *p11* - makromiejsce *M4*; miejscom *p10* i *p12* - makromiejsce *M5*; natomiast miejscom *p6* i *p13* - makromiejsce *M6* (Rys. 97a). Kolejny etap redukcji polega na zastąpieniu w sieci z Rys. 97a, makromiejsc *M2* i *M3* makromiejscem *M7*, natomiast makromiejsc *M4* i *M5* makromiejscem *M8*.



Rys. 97. a) Pierwszy, b) drugi etap redukcji sieci Petriego

Następnie dla zredukowanej sieci (Rys. 97b), przygotowano formuły Horna *HF* i *HF'*, opisujące odpowiednio, blokady i pułapki (Rys. 98).

```
HF = (/M1 + M7) * (/M1 + M6) * (/M7 + M8) * (/M7 + p8) *
     (/p8 + p7) * (/p7 + p8) * (/M6 + /M8 + p14) *
     (/p8 + /p14 + M1)

HF' = (M1 + /M7 + /M6) * (M7 + /M8 + /p8) * (/p7 + p8) *
     (/p8 + p7) * (M8 + /p14) * (M6 + /p14) *
     (p8 + /M1) * (p14 + /M1)
```

Rys. 98. Formuły Horna opisujące blokady i pułapki

Po wygenerowaniu drzew Thelena dla formuł Horna *HF* i *HF'* opisujących blokady i pułapki, uzyskano wyniki jak na Rys. 99. Oznacza to, że sieć zawiera blokady i pułapki.

```
RESULT_HF = (p7 * p8 * /p14 * /M1 * /M6 * M8) +
            (p7 * p8 * /p14 * /M1 * /M7 * /M8) +
            (p7 * p8 * /p14 * /M1 * /M6 * /M7) +
            (/p7 * /p8 * p14 * /M1 * /M7) +
            (/p7 * /p8 * /M1 * /M7 * /M8) +
            (/p7 * /p8 * /M1 * /M6 * /M7) +
            (p7 * p8 * p14 * M1 * M6 * M7 * M8)
RESULT_HF' = (/p7 * /p8 * /p14 * /M1 * /M6) +
             (p7 * p8 * /p14 * /M1 * /M6 * /M8) +
             (p7 * p8 * /p14 * /M1 * /M6 * M7) +
             (p7 * p8 * p14 * M1 * M6 * M7 * M8) +
             (/p7 * /p8 * /M1 * M6 * /M7 * M8) +
             (/p7 * /p8 * /p14 * /M1 * /M7) +
             (p7 * p8 * /p14 * /M1 * /M7 * /M8)
```

Rys. 99. Blokady i pułapki w sieci z Rys. 97b

Na podstawie powyższych rozwiązań wygenerowany został zbiór wszystkich blokad i zbiór wszystkich pułapek, jak w tabeli (Tabela 13). Na szaro zostały oznaczone minimalne blokady i minimalne pułapki.

Tabela 13. Zbiór blokad i pułapek

Blokady	***Pułapki***
{p14, M1, M6}	{p7, p8, p14, M1, M6}
{p14, M1, M6, M7}	{p7, p8, p14, M1, M6, M7}
{p14, M1, M7, M8}	{p7, p8, p14, M1, M6, M7, M8}
{p14, M1, M6, M7, M8}	{p7, p8, p14, M1, M6, M8}
{p7, p8, M1, M7, M8}	{p14, M1, M6}
{p7, p8, p14, M1, M7, M8}	{p14, M1, M6, M8}
{p7, p8, p14, M1, M6, M7, M8}	{p14, M1, M6, M7, M8}
{p7, p8, M1, M6, M7, M8}	{p7, p8, M1, M7}
{p7, p8, M1, M7}	{p7, p8, p14, M1, M7}
{p7, p8, M1, M6, M7}	{p7, p8, p14, M1, M7, M8}
{p7, p8, M1, M7, M8}	{p14, M1, M7, M8}
{p7, p8, M1, M6, M7, M8}	
{p7, p8, p14, M1, M6, M7}	

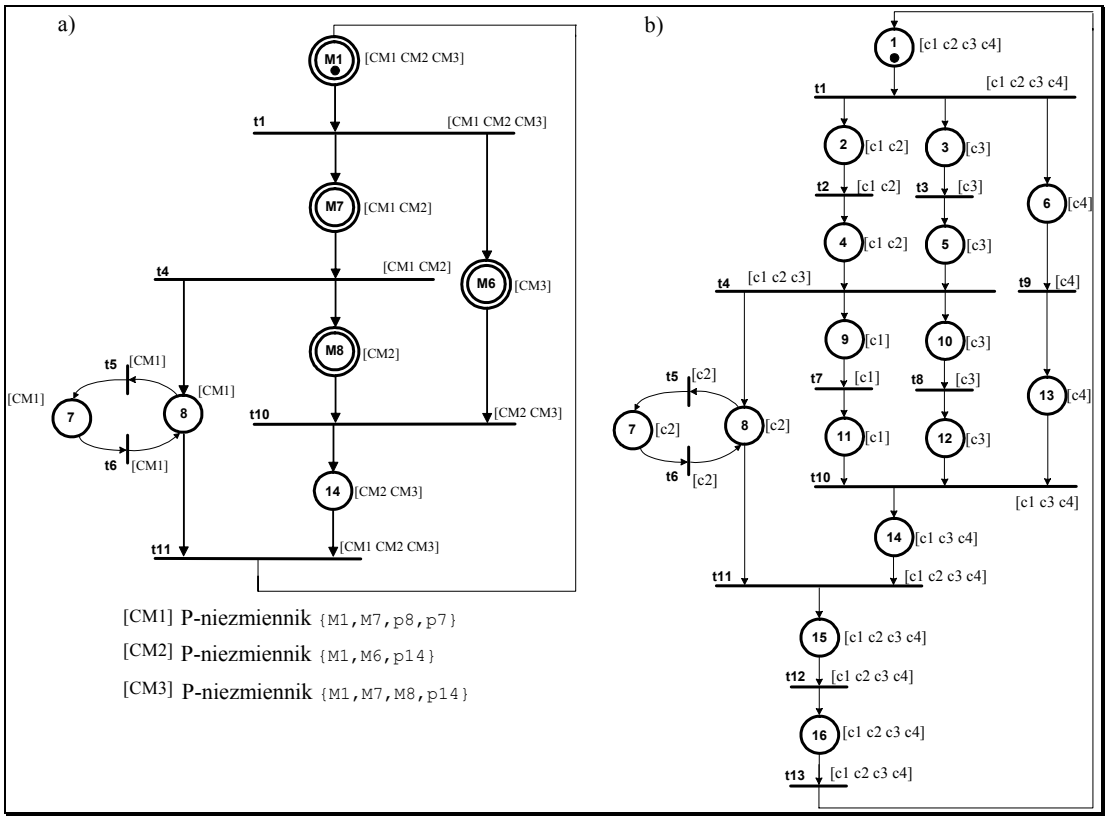
Kolejnym etapem jest sprawdzenie na podstawie uzyskanych wyników, czy badana sieć jest żywa. W pierwszym kroku należy sprawdzić, czy wszystkie minimalne blokady są oznakowane w znakowaniu początkowym. Dla analizowanego przykładu nie znaleziono minimalnej blokady nieoznakowanej w znakowaniu początkowym *M1*. Następnie sprawdzane są minimalne pułapki, czy są oznakowane w znakowaniu początkowym. Wyznaczono trzy minimalne pułapki oznakowane w znakowaniu początkowym, zostały one zaznaczone szarym kolorem w tabeli (Tabela 13). Każdy wygenerowany zbiór miejsc wyznaczający minimalną blokadę, zawiera minimalną pułapkę oznakowaną w znakowaniu początkowym *M1*. Na podstawie otrzymanych rozwiązań można stwierdzić, że badana sieć Petriego jest żywa.

Kolejnym krokiem jest sprawdzenie pokrycia sieci P-niezmiennikami, w celu stwierdzenia, czy dana sieć jest ograniczona i kolorowalna. Wyznaczane są wszystkie podzbiory miejsc odpowiadające blokadzie, które równają się podzbiorom miejsc odpowiadającym pułapkom (`{p14,M1,M6}`, `{p14,M1,M7,M8}`, `{p14,M1,M6,M7,M8}`, `{p7,p8,p14,M1,M7,M8}`, `{p7,p8,p14,M1,M6,M7,M8}`, `{p7,p8,M1,M7}`, `{p7,p8,p14,M1,M6,M7}`). Wyznaczone wektory na podstawie takich zbiorów, wymnażane są przez macierz incydencji (Rys. 100) dla badanej sieci.

	M1	M6	M7	M8	p7	p8	p14
t1	-1	1	1	0	0	0	0
t4	0	0	-1	1	0	1	0
t5	0	0	0	0	1	-1	0
t6	0	0	0	0	-1	1	0
t10	0	-1	0	-1	0	0	1
t11	1	0	0	0	0	-1	-1

Rys. 100. Macierz incydencji

Po wymnożeniu otrzymano trzy P-niezmienniki `{p14,M1,M6}`, `{p7,p8,M1,M7}`, `{p14,M1,M7,M8}`, które pokrywają całą analizowaną sieć (Rys. 101). Liczba P-niezmienników oznacza liczbę kolorów w sieci.



Rys. 101. Pokrycie sieci P-niezmiennikami

Dla analizowanej sieci z Rys. 97b liczba kolorów wynosi 4. W rozpatrywanym przykładzie kolor [CM2] makromiejsca reprezentowany jest przez trzy kolory [C1 C2 C3] w rozwinięciu tego makromiejsca.

9.5. Realizacja sterownika w językach HDL

W literaturze znane są realizacje sterowników cyfrowych opisanych sieciami Petriego w języku VHDL. Opis w językach HDL jest pomostem między opisem działania układu cyfrowego na poziomie RTL z wykorzystaniem sieci Petriego, a systemami komputerowego wspomagania projektowania CAD. Uwiarygodnienie specyfikacji jest procesem, który praktycznie nie daje się sformalizować. Rozbieżność między tym, co opisano w specyfikacji, a tym, co ma w rzeczywistości powstać jest trudna do uchwycenia. Rezultat projektowania jest co najwyżej na tyle poprawny, na ile poprawna jest specyfikacja. Symulacja funkcjonalna działania układu cyfrowego z wykorzystaniem specyfikacji jako języka HDL (np. Verilog) jest bardzo wygodnym i efektywnym sposobem uwiarygodnienia specyfikacji. Porównując wyniki symulacji z oczekiwaniami projektanta i użytkownika można stwierdzić na ile są one zgodne [WOLAŃSKI98].

W kodzie w języku Verilog można wyróżnić bloki: odzwierciedlające pewne funkcje logiczne sieci Petriego oraz odzwierciedlające strukturę rzeczywistą projektowanego układu.

Formalny zapis sieci Petriego w języku Verilog:

- $P_{we}1...P_{we}N$ - miejsca, które muszą być oznaczone tak, aby można było oznaczyć sprawdzany stan.
- $P_{wy}1...P_{wy}N$ - miejsca, które muszą być oznaczone tak, aby można było wyjść ze sprawdzanego stanu aktywną tranzycją wyjściową. Można pominąć aktualnie sprawdzane miejsce.
- T_{we} - tranzycja wejściowa do sprawdzanego miejsca.
- T_{wy} - tranzycja wyjściowa ze sprawdzanego miejsca.
- S - następna wartość stanu.
- S* - następna wartość stanu.
- $S^* = S \,\&\, \sim(T_{wy} \,\&\, P_{wy}1 \,\&\, P_{wy}2 \,..... \,\&\, P_{wy}N) | (T_{we} \,\&\, T_{we} \,\&\, P_{we}1 \,\&\, P_{we}2 \,.. \,\&\, P_{we}N)$

  Na Rys. 102 przedstawiono model w języku Verilog sieci Petriego z Rys. 94.

```
module siec   (
      clk, reset, x0, x1,x2, x3,x4,x5,x6,x7,x8,x9,
      y1, y2, y3,y4, y5, y6, y7,y8,y9 );
      input clk, reset;
      input x0, x1,x2, x3,x4,x5,x6,x7,x8,x9;
      output y1, y2, y3,y4, y5, y6, y7,y8,y9;
      reg p1, p2, p3, p4, p5, p6, p7, p8, p9,
            p10,p11,p12,p13,p14,p15,p16;
      wire  t1, t2, t3, t4,
            t5,t6,t7,t8,t9,t10,t11,t12, t13;
      assign t1 = x0;
      assign t2 = x1;
      assign t3 = x3;
      assign t4 = 1'b1 ;
      assign t5 = x5 & x6;
      assign t6 = ~x5;
      assign t7 = ~x2;
      assign t8 = ~x4;
      assign t9 = x7;
      assign t10 = 1'b1 ;
      assign t11 = ~x6;
      assign t12 = x8;
      assign t13 = ~x9;
      assign y1 = p2;
      assign y2 = p3;
      assign y3 = p9;
      assign y4 = p10;
      assign y5 = p14;
      assign y6 = p16;
      assign y7 = p7;
      assign y8 = p15;
      assign y9 = p6;
// miejsce 1
always @(posedge clk)
begin  if (reset)  p1 <= 1;
      else p1 <= (p1 & ~t1 )| (t13 & p16 );
end
// miejsce 2
always @(posedge clk)
begin  if (reset)  p2 <= 0;
      else p2 <= (p2 &  ~t2 )| (t1 & p1 ) ;
end
// miejsce 3
always @(posedge clk)
begin  if (reset)  p3 <= 0;
      else p3 <= (p3 & ~t3 ) | (t1 & p1) ;
end
// miejsce 4
always @(posedge clk)
begin  if (reset)  p4 <= 0;
      else p4 <= (p4 & ~(t4&p5))|(t2 & p2);
end
// miejsce 5
always @(posedge clk)
begin  if (reset)  p5 <= 0;
      else p5 <= (p5 & ~(t4&p4))|(t3 & p3);
end
```

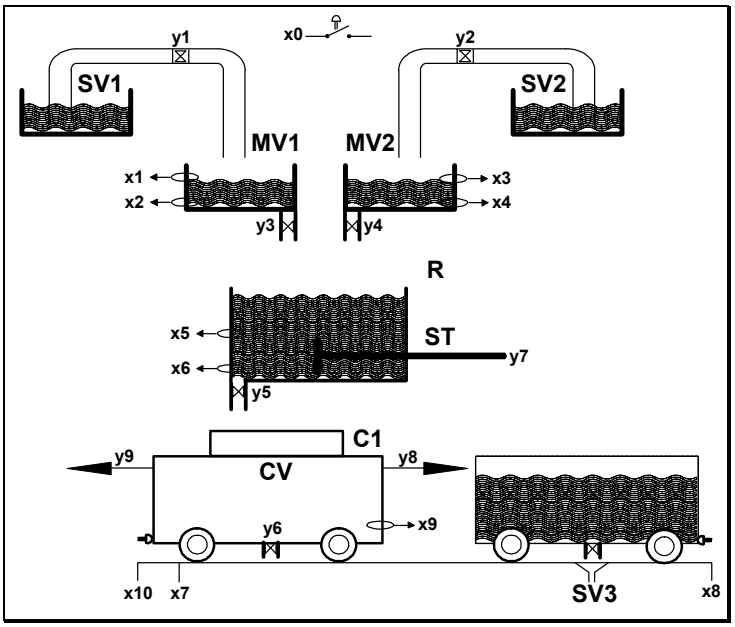
```
// miejsce 6
always @(posedge clk)
begin  if (reset)  p6 <= 0;
      else p6 <= (p6 &  ~t9 ) | (p1 & t1 );
end
// miejsce 7
always @(posedge clk)
begin  if (reset)  p7 <= 0;
      else p7 <= (p7 &  ~t6 ) | (t5 & p8 ) ;
end
// miejsce 8
always @(posedge clk)
begin  if (reset)  p8 <= 0;
      else p8 <= (p8 & ~(t5 | (t11&p14))) |(t4 &
      p4 &p5|p7&t6)
end
// miejsce 9
always @(posedge clk)
begin  if (reset)  p9 <= 0;
      else p9 <= (p9 & ~t7)|(t4 & p4 & p5 );
end
// miejsce 10
always @(posedge clk)
begin  if (reset)  p10 <= 0;
      else p10 <= (p10 & ~t8)|(t4 & p4 & p5);
end
// miejsce 11
always @(posedge clk)
begin  if (reset)  p11 <= 0;
      else p11 <= (p11&~(t10&p12&p13))|(t7&p9);
end
// miejsce 12
always @(posedge clk)
begin  if (reset)  p12 <= 0;
      else p12 <= (p12&~(t10&p11&p13))|(t10&p9);
end
// miejsce 13
always @(posedge clk)
begin  if (reset)  p13 <= 0;
      else p13 <= (p13&~(t10&p11&p12))|(t9&p6);
end
// miejsce 14
always @(posedge clk)
begin  if (reset)  p14 <= 0;
      else
p14<=(p14&~(t11&p8))|(t10&p11&p12&p13);
end
// miejsce 15
always @(posedge clk)
begin  if (reset)  p15 <= 0;
      else p15 <= (p15 & ~t12 )|(t11& p8 & p14);
end
// miejsce 16
always @(posedge clk)
begin  if (reset)  p16 <= 0;
      else p16 <= (p16 &  ~t13 ) | (t12 & p15 );
end
endmodule
```

Rys. 102. Model w języku Verilog sieci Petriego

Na początku kodu standardowo dla języka Verilog deklaruje się interfejs projektowanego układu. Oznacza się ilość, typy oraz rozmiary portów wejścia/wyjścia. Kolejną częścią kodu jest konstrukcja *assign*. Konstrukcja ta decyduje o tym, jakie funkcje logiczne pojawią się na wyjściach układu zależnie od aktywnego stanu sieci. Kolejną częścią opisu jest grupa konstrukcji *always*. Ciało każdej z konstrukcji kontroluje jedno miejsce sieci Petriego. Stan większości miejsc jest zerowany, gdy pojawi się sygnał *reset*. Niektóre miejsca, które są oznaczone podczas inicjalizacji sieci ustawiane są na wartość ‘1’.

9.6. Zmodyfikowany przykład sterownika

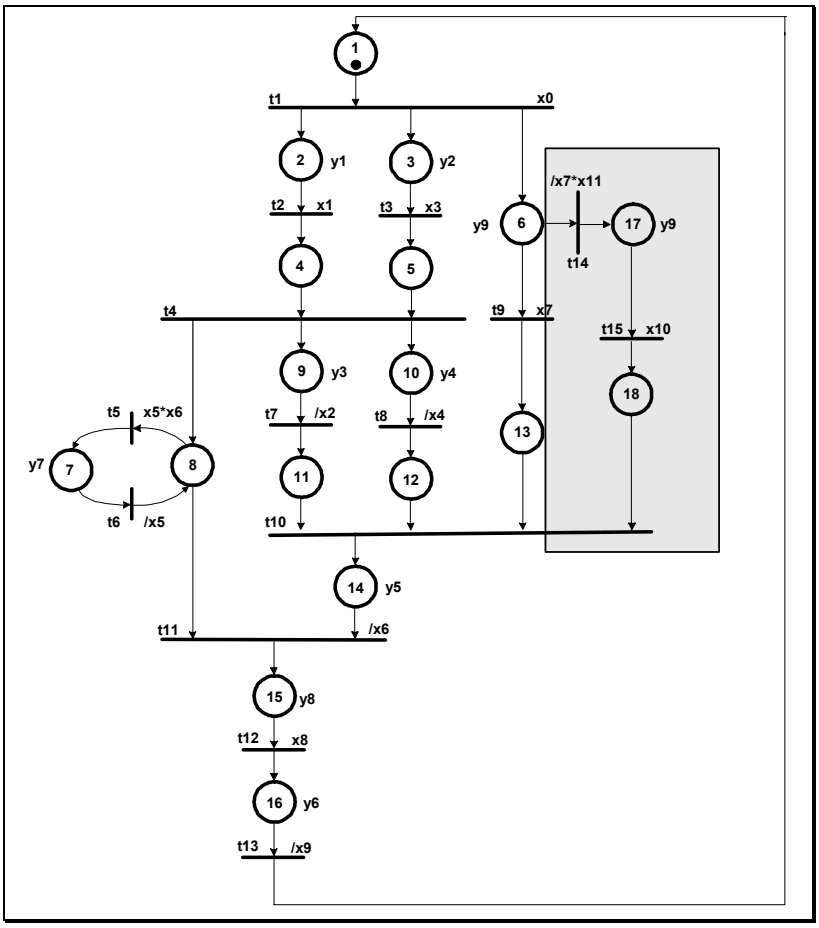
W celu zobrazowania działania algorytmu, dla źle skonstruowanej sieci, zmodyfikowano przykład procesu mieszania i transportu cieczy (Rys. 103).



Rys. 103. Zmodyfikowany model rzeczywisty procesu mieszania i transportu cieczy

System działa podobnie jak w wersji pierwotnej, jednakże dodano kolejny wózek *C1*, który jest dłuższy od wózka *C*, natomiast komora wlewu jest krótsza od długości wózka. W związku z tym, wózek powinien przejechać pozycję *x7* i zatrzymać się w pozycji *x10*, w celu napełnienia go. Rys. 104 pokazuje model sieci Petriego, odpowiadający procesowi mieszania i transportu cieczy. Model został przygotowany w ten sposób, że dodano dodatkowe miejsca *p17* i *p18*, oraz tranzycje *t14* i *t15*, które opisują drogę dla dłuższego wózka *C1*, który musi dojechać do pozycji *x10*. Projektant sporządzając specyfikację nie zdawał sobie sprawy, że wprowadzając modyfikację doprowadził układ do ewentualnego zastoju.

Rys. 105 przedstawia te fragmenty w opisie sieci w formacie PNSF3, które zostały dodane do poprzedniego modelu (Rys. 95 i Rys. 96). Fragmenty zacieniowane zostały dodane do formatu PNSF3, dla sieci z Rys. 94. Widać wyraźnie, że modyfikacja układu ma charakter lokalny i łatwo jest zidentyfikować fragmenty, które zostały tutaj wprowadzone.



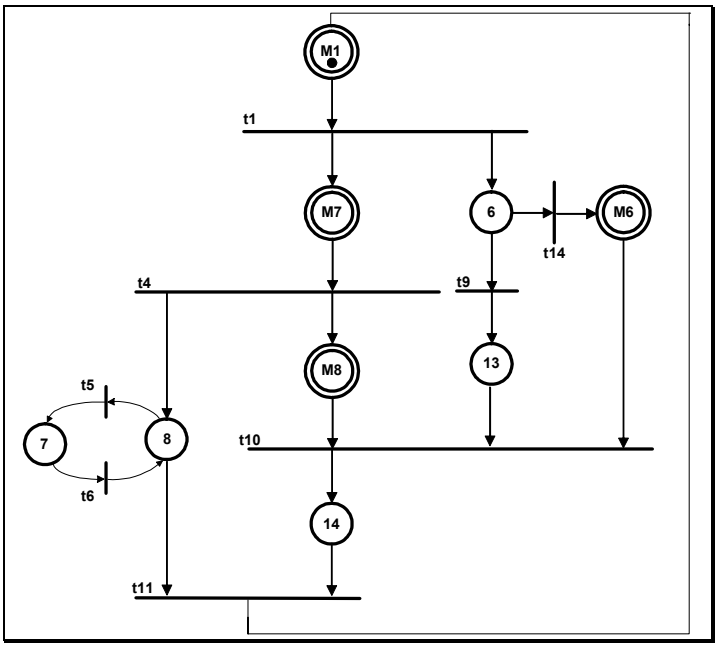
Rys. 104. Zmodyfikowana sieć Petriego

```
<?xml version="1.0" encoding="ISO-8859-2" standalone="no"?>
<!DOCTYPE PNSF3 SYSTEM "pnsf3.dtd">
<PNSF3>
<CLOCKS>    <CLOCK ID="clk1" > clk </CLOCK>    </CLOCKS>
 <INPUTS>       …
      <INPUT ID="i11" > x10 </INPUT>
      <INPUT ID="i12" > x11 </INPUT>
 </INPUTS>
 <OUTPUTS>       …     </OUTPUTS>
 <REG_OUTPUTS>      …  </REG_OUTPUTS>
<PLACES>     …
    <PLACE ID="p17" > p17 </PLACE>
    <PLACE ID="p18" > p18 </PLACE>
  </PLACES>
<PREDICATES>    …
    <PREDICATE ID="pred7">/x7*x11</PREDICATE>
</PREDICATES>
<TRANSITIONS>     …  </TRANSITIONS>
<NET>    …
   <ARC ID_TRANSITION="t14" ID_IN_PLACES="p6" ID_OUT_PLACES="p17"> </ARC>
   <ARC ID_TRANSITION="t15" ID_IN_PLACES="p17" ID_OUT_PLACES="p18"> </ARC>
</NET>
<MOORE_OUTPUTS>    …
   <MOORE_DESC ID_IN_PLACES="p17" ID_OUT_SIGNALS="o9" > </MOORE_DESC>
</MOORE_OUTPUTS>
</PNSF3>
```

Rys. 105. Format PNSF3 dla sieci z Rys. 94

Rys. 106 prezentuje ostatni - trzeci etap redukcji sieci z Rys. 104. Miejscom *p1*, *p15*, *p16* odpowiada makromiejsce *M1*. Miejscom *p2*, *p4*, *p3*, *p5* odpowiada

makromiejsce *M7*, miejscom *p9*, *p11*, *p10*, *p12* odpowiada makromiejsce *M8*, miejscom *p17* i *p18* odpowiada makromiejsce *M6*.



Rys. 106. Uproszczenia sieci Petriego z Rys. 104

Na Rys. 107 przedstawia formuły Horna *HF* i *HF'*, opisujące blokady i pułapki tej sieci. Na podstawie tych równań wygenerowano drzewa Thelena i uzyskano wyniki, jak na Rys. 109.

```
HF = (/M1 + M7) * (/M1 + p6) * (/M7 + p8) * (/M7 + M8) *
     (/p7 + p8) * (/p8 + p7) * (/p6 + p13) * (/p6 + M6) *
     (/M8 + /p13 + /M6 + p14) * (/p8 + /p14 + M1)
HF' = (M1 + /p6 + /M7) * (M7 + /p8 + /M8) * (p8 + /p7)
    * (p7 + /p8) * (p6 + /p13) * (p6 + /M6) * (M8 + /p14)
   * (p13 + /p14) * (M6 + /p14) * (p8 + /M1) * (p14 + /M1)
```

Rys. 107. Formuły Horna opisujące blokady i pułapki

```
HF_RESULT = (/M1 * /M6 * M8 * /p6 * p7 * p8 * /p14) +
            (/M1 * M8 * /p6 * p7 * p8 * /p13 * /p14) +
            (/M1 * M6 * /M7 * /M8 * p7 * p8 * p13 * /p14) +
            (/M1 * /M6 * /M7 * /p6 * p7 * p8 * /p14) +
            (/M1 * /M7 * /p6 * p7 * p8 * /p13 * /p14) +
            (/M1 * /M7 * /M8 * /p6 * p7 * p8 * /p14) +
            (/M1 * M6 * /M7 * /M8 * /p7 * /p8 * p13) +
            (/M1 * M6 * /M7 * /p7 * /p8 * p13 * p14) +
            (/M1 * /M6 * /M7 * /p6 * /p7 * /p8) +
            (/M1 * /M7 * /p6 * /p7 * /p8 * /p13) +
            (/M1 * /M7 * /M8 * /p6 * /p7 * /p8) +
            (/M1 * /M7 * /p6 * /p7 * /p8 * p14)
```

Rys. 108. Blokady w sieci z Rys. 106

```
HF_RESULT' = (/M1 * /M6 * /M8 * /p6 * p7 * p8 * /p13 * /p14) +
             (/M1 * /M6 * /p6 * /p7 * /p8 * /p13 * /p14) +
             (/M1 * /M6 * M7 * /p6 * p7 * p8 * /p13 * /p14) +
             (/M1 * /M6 * /M7 * /M8 * p7 * p8 * /p13 * /p14) +
             (/M1 * /M7 * /M8 * p6 * p7 * p8 * /p14) +
             (/M1 * /M6 * /M7 * /p7 * /p8 * /p13 * /p14) +
             (/M1 * M6 * /M7 * M8 * p6 * /p7 * /p8 * p13) +
             (/M1 * /M7 * p6 * /p7 * /p8 * /p14)
```

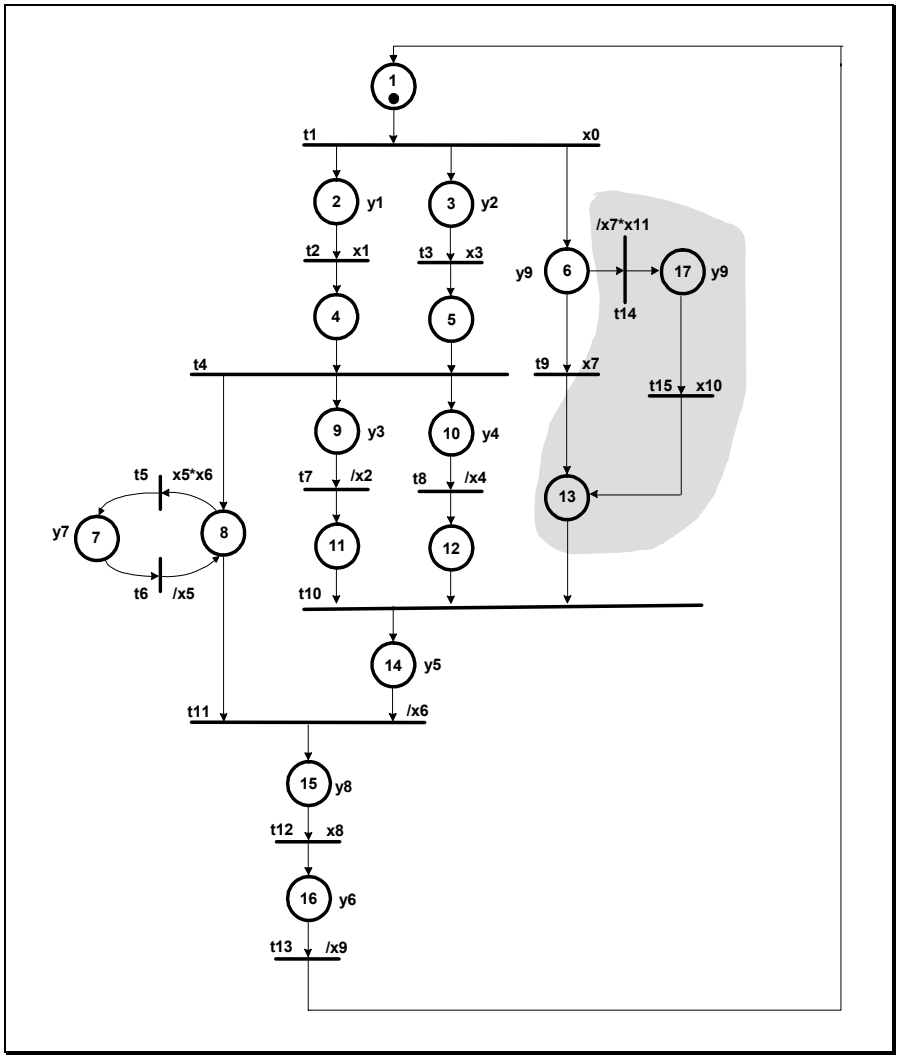
Rys. 109. Pułapki w sieci z Rys. 106

Na podstawie rozwiązań (Rys. 108 i Rys. 109) wygenerowany został zbiór wszystkich blokad i zbiór wszystkich pułapek, jak w tabeli (Tabela 14). Każda minimalna blokada jest oznakowana w znakowaniu początkowym. Wyznaczone minimalne pułapki również oznakowane są w znakowaniu początkowym, czyli należy sprawdzić zawieranie pułapek minimalnych przez minimalne blokady. Analizowana sieć nie jest żywa, gdyż dwa zbiory minimalnych blokad nie zostały pokryte przez minimalne pułapki oznakowane w znakowaniu początkowym: {M1,M6,p6,p14}, {M1,p6,p13,p14}.

Tabela 14. Zbiór blokad i pułapek

Blokady	*Pułapki*
{M1,M6,p6,p14}	{M1,M6,M8,p6,p13,p14}
{M1,M6,M7,p6,p14}	{M1,M6,M7,M8,p6,p13,p14}
{M1,M6,p6,p13,p14}	{M1,M6,p6,p7,p8,p13,p14}
{M1,M6,M7,p6,p13,p14}	{M1,M6,M8,p6,p7,p8,p13,p14}
{M1,p6,p13,p14}	{M1,M6,M7,p6,p7,p8,p13,p14}
{M1,M7,p6,p13,p14}	{M1,M6,M7,M8,p6,p7,p8,p13,p14}
{M1,M7,M8,p14}	{M1,M6,p6,p13,p14}
{M1,M7,M8,p6,p14}	{M1,M6,M7,M8,p13,p14}
{M1,M6,M7,M8,p6,p14}	{M1,M7,M8,p14}
{M1,M6,M7,M8,p6,p13,p14}	{M1,M7,M8,p13,p14}
{M1,M7,M8,p6,p13,p14}	{M1,M6,M7,M8,p14}
{M1,M7,M8,p7,p8}	{M1,M6,M7,M8,p7,p8,p13,p14}
{M1,M7,M8,p6,p7,p8}	{M1,M6,M7,p7,p8,p13,p14}
{M1,M7,M8,p7,p8,p14}	{M1,M7,p7,p8}
{M1,M7,M8,p6,p7,p8,p14}	{M1,M7,p7,p8,p14}
{M1,M7,p7,p8}	{M1,M7,p7,p8,p13,p14}
{M1,M7,p6,p7,p8}	{M1,M7,M8,p7,p8,p14}
{M1,M7,M8,p6,p7,p8}	{M1,M7,M8,p7,p8,p13,p14}
{M1,M6,M7,p6,p7,p8}	{M1,M6,M7,p7,p8,p14}
{M1,M6,M7,p6,p7,p8,p14}	{M1,M6,M7,M8,p7,p8,p14}
{M1,M6,M7,p6,p7,p8,p13}	
{M1,M6,M7,M8,p6,p7,p8}	
{M1,M6,M7,M8,p6,p7,p8,p14}	
{M1,M6,M7,M8,p6,p7,p8,p13}	
{M1,M6,M7,M8,p6,p7,p8,p13,p14}	
{M1,M6,M7,p6,p7,p8,p13,p14}	
{M1,M7,p6,p7,p8,p13}	
{M1,M7,p6,p7,p8,p13,p14}	
{M1,M7,M8,p6,p7,p8,p13}	
{M1,M7,M8,p6,p7,p8,p13,p14}	
{M1,M7,M8,p6,p7,p8}	

Błędnie zaprojektowaną sieć można poprawić jak na Rys. 110. Błąd projektanta polegał na tym, że potraktował on stan lokalny *p17*, oznaczający, że długi wózek jedzie w lewo, jako stan alternatywny do stanów *p6* (krótki wózek jedzie w lewo) i *p13* (oczekiwanie na napełnienie zbiornika *CV*). Stan *p17* może być jedynie stanem alternatywnym do stanu *p6*. Poprawiony fragment sieci został zaznaczony na rysunku kolorem szarym.



Rys. 110. Poprawiona sieć Petriego

10. Podsumowanie

Sieć Petriego może być analizowana i animowana w celu wykrycia podstawowych błędów strukturalnych w modelowanym algorytmie oraz niezgodności ze specyfikacją w języku naturalnym.

W pracy przedstawiono nową metodę badania żywotności i ograniczoności sieci Petriego. Metoda ta łączy ze sobą symboliczne przetwarzaniu danych oraz metodę redukcji sieci. Należy zwrócić szczególną uwagę na fakt, że opracowana metoda, należy do metod dokładnych, czyli wyraźnie wskazywane jest miejsce wystąpienia defektu w sieci, co jest bardzo ważnym atutem w procesie projektowania sterowników logicznych. Dużą zaletą opracowanej metody jest możliwość analizy sieci należących klasy AC i NSC. Istnieją metody analizy sieci Petriego bardziej efektywne, jednakże nie są one metodami tak dokładnymi, gdyż dają jedynie odpowiedź, czy badana sieć spełnia pewne własności. W opracowanym podejściu ukierunkowano się na pokazanie projektantowi tych fragmentów, które są błędne, aby w szybkim czasie można było je usunąć.

Podejście symboliczne pozwoliło na wykorzystanie logiki matematycznej oraz narzędzi z nią związanych. Strukturę topologiczną sieci Petriego przedstawić można w sposób regułowy, co umożliwia łatwe przejście z takiego zapisu do równań opisujących blokady i pułapki. Kolejną zaletą symbolicznego przetwarzania danych jest to, że nie wymaga ona skomplikowanych struktur danych. Pozwala również na zwięzłe reprezentowanie zbiorów blokad i pułapek.

Rozpatrywana klasa sterowników z założenia powinna zostać zaimplementowana w strukturach FPGA. Sterowniki takie nie są na tyle duże, że metody symboliczne nie mogą być wykorzystywane, ze względu na mniejszą efektywność. Nad efektywnością przeważa podejście praktyczne, czyli możliwość zlokalizowania błędów. Liczba rozwiązań w przypadku rzeczywistych układów nie jest duża. Ponadto w reprezentowanym podejściu zajętość pamięci jest liniowa względem liczby zmiennych w równaniu, czyli obszaru pamięci na wykonanie obliczeń nie powinno zabraknąć.

Opracowaną metodę można zastosować do dekompozycji automatu współbieżnego na sekwencyjne automaty składowe. Dekompozycja wykorzystywana jest często w celu ułatwienia procesu syntezy układu cyfrowego. Na podstawie opracowanego algorytmu wyznaczane są P-niezmienniki sieci, które reprezentują składowe automatowe. Na podstawie wygenerowanych P-niezmienników kolorowana jest sieć. Pokolorowana sieć Petriego, oprócz informacji o współbieżności i sekwencyjności zdarzeń, niesie informację

o przynależności miejsc do poszczególnych procesów sekwencyjnych [WĘGRZYN98]. Przedstawione podejście pozwala również na kodowanie miejsc sieci, a tym samym stanów lokalnych automatu współbieżnego [ADAMSKI86], [KOZŁOWSKI,ET.AL.95], [PARDEY,ET.AL.92]. Odpowiednie kodowanie jest zasadniczym środkiem umożliwiającym bezpośrednią implementację sieci Petriego w programowalnych strukturach logicznych.

Do symbolicznego opisu sieci Petriego i sieci SFC można wykorzystywać język regułowy oparty na logice sekwentów Gentzena, z elementami logiki temporalnej [ADAMSKI90]. Stał się on podstawą uproszczonej, tekstowej reprezentacji sieci interpretowanych i został nazwany formatem PNSF [KOZŁOWSKI,ET.AL.95]. Podobną postać regułową wykorzystano w języku CONPAR [FERNANDES,ET.AL.97]. Zalety obydwu reprezentacji [WĘGRZYN98] połączono w uzupełnionym formacie PNSF2. Ze względów teoretycznych i praktycznych, w pracy zaproponowano wykorzystanie technologii XML do przedstawienia interpretowanych, hierarchicznych, kolorowanych sieci Petriego (format PNSF3), które odzwierciedlają algorytm sterowania. Opracowany format został wykorzystany do symulacji sieci, która może być pierwszym krokiem w etapie weryfikacji zamodelowanego układu. Atutem tego formatu jest łatwe przekształcanie informacji zgromadzonej w bazie danych w postaci relacyjnej na język XML i odwrotnie. W związku z czym, opracowano system automatycznej transformacji sieci SFC na sieci Petriego. Format PNSF3 pozwala na łatwe wygenerowanie regułowego opisu topologicznej struktury sieci Petriego w postaci formuł Horna. Z drugiej strony zachowana jest odpowiedniość pomiędzy uzyskanymi w drodze analizy podsieciami a modelem w formacie PNSF3.

Praca doktorska zawiera szereg elementów będących oryginalnym dorobkiem autorki. Do najważniejszych własnych osiągnięć autorka zalicza:

- Opracowanie oryginalnej metody analizy sieci Petriego bazującej na symbolicznym przetwarzaniu danych i redukcji sieci, oraz nowej metody wyznaczania P-niezmienników, w celu dekompozycji i kodowania sieci Petriego.
- Wprowadzenie nowego formatu XML opisu interpretowanej, hierarchicznej i kolorowanej sieci Petriego.
- Opracowanie modelu relacyjnej bazy danych, reprezentującej sieci Petriego i sieci SFC, w celu przechowywania informacji o strukturze modeli oraz transformacji sieci SFC na sieci Petriego z wykorzystaniem formatu XML.
- Opracowanie metody symulacji interpretowanej, hierarchicznej i kolorowanej sieci Petriego z wykorzystaniem opracowanego formatu XML.

W ramach badań powstało szereg programów o charakterze pomocniczym, takich jak: program analizy sieci opierający się na opracowanym algorytmie analizy, program wyznaczania P-niezmienników sieci, oraz dekompozycji sieci na składowe automatowe, program symulacji sieci z wykorzystaniem opracowanego formatu PNSF3, system automatycznej transformacji sieci SFC na sieci Petriego w ramach, którego opracowano moduły tworzenia modeli sieci Petriego, sieci SFC, moduły wprowadzania danych z formatów XML opisu tych modeli, moduł automatycznej transformacji do formatu PNSF3.

W zamierzeniu autorki jest dalsze rozwijanie opracowanego algorytmu. Kolejnym etapem będzie wprowadzenie współbieżności w celu przyspieszenia wykonywania obliczeń. W związku z tym, że algorytm ma zajętość pamięci liniową, współbieżne wykonywanie obliczeń nie powinno zbytnio obciążyć zasobów. Rozpatrywane jest podejście w dwojaki sposób: podział formuły na kilka części i współbieżne tworzenie drzew dla poszczególnych fragmentów formuły, bądź równoległe przeszukiwanie poszczególnych gałęzi w drzewie. Rozważane jest również wykorzystanie, oprócz oprogramowania współbieżnego, rozwiązań sprzętowych.

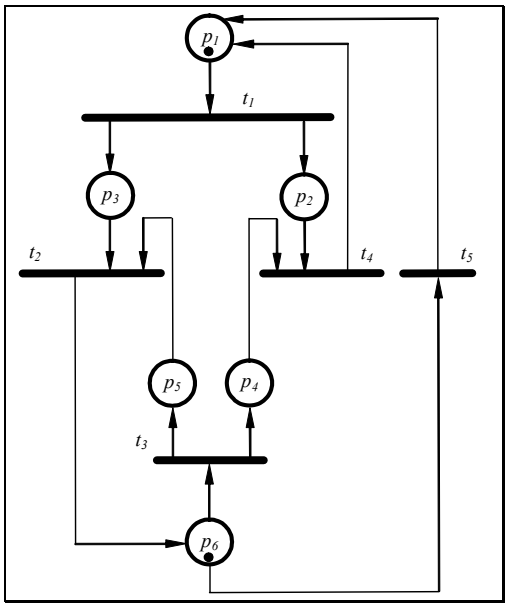
Najważniejsze częściowe wyniki pracy doktorskiej zostały przedstawione w dwóch artykułach, dziesięciu referatach na konferencjach o zasięgu międzynarodowym oraz dwóch na konferencjach krajowych. Znaczna część pracy została wykonywana w ramach grantów KBN, dla młodych badaczy i grantu promotorskiego (8 T11C 018 18 oraz 7 T11C 009 20).

Dodatek. Przykłady analizy wybranych sieci Petriego

W rozdziale tym zostaną przedstawione sieci z różnych klas, pozytywnie zweryfikowane pod kątem badania żywotności i ograniczoności. Przykłady sieci zaczerpnięte zostały z [BARKAOUI,MINOUX92], [GIRAULT,VALK03], [KOTOW84], [MURATA89], [STARKE89], [TOULOTTE,PARSY79], [WĘGRZYN98].

Przykłady sieci z klasy FC

Na Rys. 111 przedstawiono sieć z klasy FC, zaczerpniętą z [KOTOW84]. Natomiast w tabeli (Tabela 15) zamieszczono minimalne blokady i minimalne pułapki. Oznakowane w znakowaniu początkowym są miejsca *p1* i *p6*. Wszystkie minimalne blokady i pułapki są oznakowane w znakowaniu początkowym. Blokada {p5,p6} nie zawiera minimalnych pułapek, czyli sieć nie jest żywa. Sieć jest jednak ograniczona, gdyż wszystkie miejsca zostały pokryte przez wyznaczone P-niezmienniki {p1,p3,p4,p6}, {p1,p2,p5,p6}. Tabela z wyznaczonymi wszystkimi blokadami i pułapkami została zamieszczona w rozdziale 2.3.

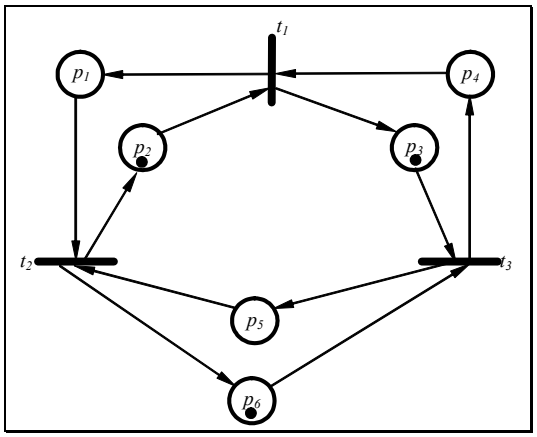


Rys. 111. Przykład 1 sieci z klasy FC

Tabela 15. Minimalne blokady i minimalne pułapki (przykład 1 sieci FC)

Minimalne blokady	*Minimalne pułapki*
{p5,p6}	{p1,p2}
{p1,p2,p3,p6}	{p1,p3,p5,p6}
{p1,p3,p4,p6}	{p1,p3,p4,p6}

Na Rys. 112 zamieszczono kolejny przykład sieci z klasy FC. Został on zaczerpnięty z [STARKE86]. Tabela 16 zawiera minimalne blokady i minimalne pułapki. Oznakowane w znakowaniu początkowym są miejsca *p2*, *p3* i *p6*. Wszystkie minimalne blokady i minimalne pułapki są oznakowane w znakowaniu początkowym. Każda minimalna blokada zawiera minimalną pułapkę, czyli sieć jest żywa. Jest ona również ograniczona, gdyż została pokryta P-niezmiennikami {p2,p3,p5}, {p3,p4}, {p1,p4,p6}, {p5,p6}, {p1,p2}.

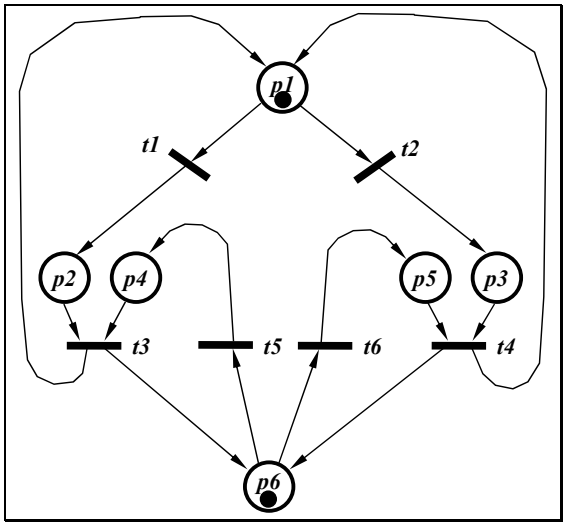


Rys. 112. Przykład 2 sieci z klasy FC

Tabela 16. Minimalne blokady i minimalne pułapki (przykład 2 sieci FC)

Minimalne blokady	*Minimalne pułapki*
{p2,p3,p5}	{p2,p3,p5}
{p3,p4}	{p3,p4}
{p1,p4,p6}	{p1,p4,p6}
{p5,p6}	{p5,p6}
{p1,p2}	{p1,p2}

Rys. 113 przedstawia przykład sieci z klasy FC zaczerpnięty z [WĘGRZYN98]. Tabela 17 zawiera minimalne blokady i minimalne pułapki. Sieć jest oznakowana w znakowaniu początkowym w miejscach *p1* i *p6*. Wszystkie minimalne blokady i pułapki są oznakowane w znakowaniu początkowym. Dwie minimalne blokady nie zawierają pułapek minimalnych: {p1,p2,p5,p6}, {p1,p3,p4,p6}, czyli badana sieć nie jest żywa. Jednakże jest ona ograniczona, gdyż została pokryta przez P-niezmienniki {p1,p2,p3} i {p4,p5,p6}.



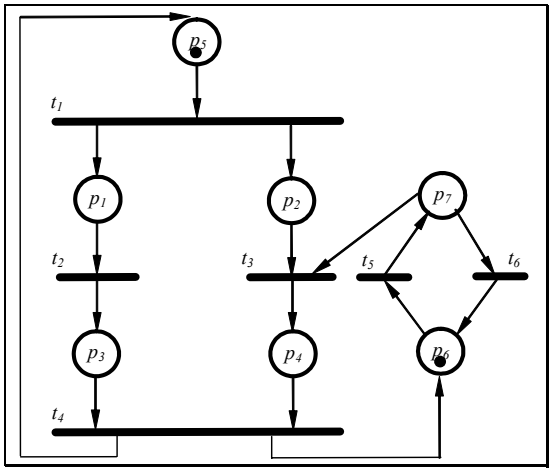
Rys. 113. Przykład 3 sieci z klasy FC

Tabela 17. Minimalne blokady i minimalne pułapki (przykład 3 sieci FC)

Minimalne blokady	*Minimalne pułapki*
{p4,p5,p6}	{p4,p5,p6}
{p1,p2,p3}	{p1,p2,p3}
{p1,p2,p5,p6}	
{p1,p3,p4,p6}	

Przykład sieci z klasy NSC

Na Rys. 114 przedstawiono sieć z klasy NSC zaczerpniętą z [BARKAOUI,MINOUX92]. Natomiast Tabela 18 zawiera minimalne blokady i minimalne pułapki dla badanej sieci. Sieć jest oznakowana w znakowaniu początkowym w miejscach *p5* i *p6*. W celu sprawdzenia żywotności sieci należy sprawdzić, czy wszystkie minimalne blokady równe są pułapkom. W analizowany przykładzie wszystkie minimalne blokady równe są minimalnym pułapkom, czyli sieć jest żywa (twierdzenie 5, rozdział 2.3).



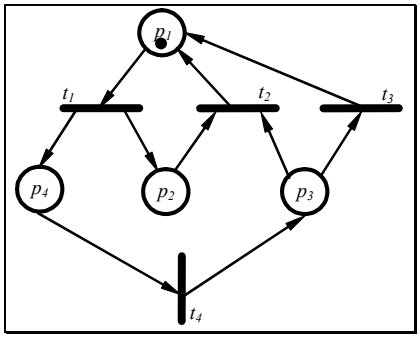
Rys. 114. Przykład sieci z klasy NSC

Tabela 18. Minimalne blokady i minimalne pułapki (przykład sieci NSC)

Minimalne blokady	*Minimalne pułapki*
{p4,p6,p7}	{p4,p6,p7}
{p1,p3,p5}	{p1,p3,p5}
{p2,p4,p5}	{p2,p4,p5}

Przykłady sieci z klasy AC

Rys. 115 przedstawia sieć z klasy AC zaczerpniętą z [MURATA89]. Tabela 19 zawiera minimalne blokady i minimalne pułapki. Minimalna blokada i minimalne pułapki są oznakowane w znakowaniu początkowym *p1*. Wyznaczona minimalna blokada jest pułapką, czyli sieć jest żywa.

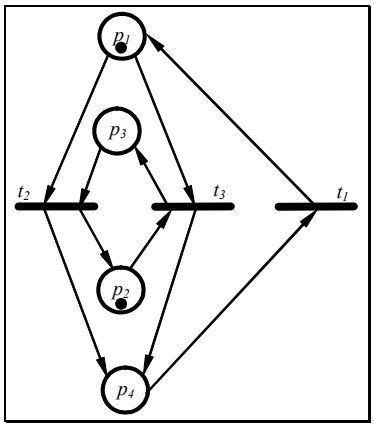


Rys. 115. Przykład 1 sieci z klasy AC

Tabela 19. Minimalne blokady i minimalne pułapki (przykład 1 sieci AC)

Minimalne blokady	*Minimalne pułapki*
{p1,p3,p4}	{p1,p2}
	{p1,p3,p4}

Rys. 116 przedstawia sieć z klasy AC [MURATA89]. W tabeli (Tabela 20) zamieszczono wszystkie minimalne blokady i minimalne pułapki, które są oznakowane w znakowaniu początkowym *p1* i *p2*. Minimalne blokady {p1,p4} i {p2,p3} nie są pułapkami, czyli sieć nie jest żywa.

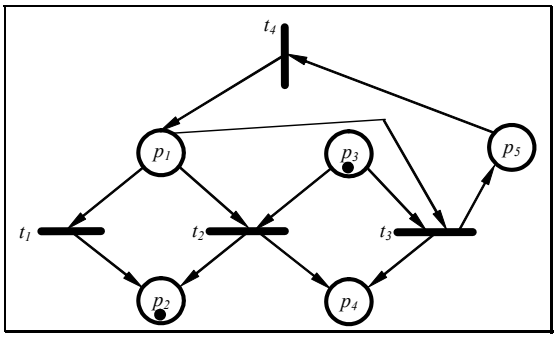


Rys. 116. Przykład 2 sieci z klasy AC

Tabela 20. Minimalne blokady i minimalne pułapki (przykład 2 sieci AC)

Minimalne blokady	*Minimalne pułapki*
{p1,p4}	{p1,p2}
{p2,p3}	{p1,p3,p4}

Na Rys. 117 został przedstawiony kolejny przykład sieci z klasy AC [GIRAULT,VALK03]. Natomiast Tabela 21 zawiera minimalne blokady i pułapki dla badanej sieci. Każda minimalna blokada jest oznakowaną w znakowaniu początkowym *p2* i *p3* pułapką, czyli analizowana sieć jest żywa.



Rys. 117. Przykład 3 sieci z klasy AC

Tabela 21. Minimalne blokady i minimalne pułapki (przykład 3 sieci AC)

Minimalne blokady	*Minimalne pułapki*
{p1,p2,p5}	{p1,p2,p5}
{p3,p4}	{p3,p4}

Bibliografia

[ABOUAISSA,ET.AL.95]
H.Abouaissa, B.Rabenasolo, M.Ferney, P.Vaudrey, "Generic models for representing reactive and complex systems using hierarchical coloured Petri nets", *Petri Net Newsletter*, No. 48, ss.27-48, April 1995

[ADAMSKI86]
M.Adamski, "Heurystyczna metoda strukturalnego kodowania miejsc sieci Petriego", *Zeszyty Naukowe Wyższej Szkoły Inżynierskiej*, Zielona Góra, Vol. 78, No.12, Elektrotechnika, ss.113-125, 1986

[ADAMSKI90]
M.Adamski, *Projektowanie układów cyfrowych systematyczną metodą strukturalną*, Monografia Nr 49, Wydawnictwo Wyższej Szkoły Inżynierskiej w Zielonej Górze, Zielona Góra, 1990

[ADAMSKI91]
M.Adamski, "Parallel Controller Implementation using Standard PLD Software", w: W.R.Moore, W.Luk, (red.), *FPGAs*, Abingdon EE&CS Books, Abingdon, England, ss.296-304, 1991

[ADAMSKI98]
M.Adamski, "Metodologia projektowania reprogramowalnych sterowników logicznych z wykorzystaniem elementów CPLD i FPGA", *Reprogramowalne Układy Cyfrowe - RUC : Materiały I Krajowej Konferencji Naukowej*. Szczecin, Polska, 1998, ss. 15—22

[ADAMSKI98A]
M.Adamski, "SFC, Petri Nets and Application Specific Logic Controllers", *IEEE International Conference on Systems, Man, and Cybernetics - SMC '98*, San Diego, USA, 1998, Vol.1, ss. 728-733

[ADAMSKI,WĘGRZYN94]
M.Adamski, M.Węgrzyn, "Hierarchically Structured Coloured Petri Net Specification and Validation of Concurrent Controllers", *Proceedings of the 39th International Scientific Colloquium, IWK'94*, Ilmenau, Niemcy, 27-30.09.1994, Band 1, ss.517-522, 1994

[ADAMSKI,MONTEIRO94]
M.Adamski, J.L.Monteiro: "Restricted Grafcet and its Rule-based Specification and Implementation", *Proceedings of the 39th International Scientific Colloquium, IWK'94*, Ilmenau, Niemcy, 27-30.09.1994, Band 1, ss.543-548

[ADAMSKI,CHODAŃ00]
M.Adamski, M.Chodań, *Modelowanie układów sterowania dyskretnego z wykorzystaniem sieci SFC*, Wydawnictwo Politechniki Zielonogórskiej, Zielona Góra, 2000

[ANDRZEJEWSKI02]
G.Andrzejewski, *Programowy model interpretowanej sieci Petriego dla potrzeb projektowania mikrosystemów cyfrowych*, Rozprawa doktorska, Politechnika Szczecińska, Szczecin, 2002

[BANASZAK,ET.AL.93]
Z.Banaszak, J.Kuś, M.Adamski, *Sieci Petriego. Modelowanie, Sterowanie i Synteza Systemów Dyskretnych*, Wydawnictwo Wyższej Szkoły Inżynierskiej, Zielona Góra, 1993

[BARKAOUI,MINOUX92]
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

[BARKAOUI95]
K.Barkaoui, J.M.Couvreur, C.Dutheillet, "On Liveness in Extended Non Self-Controlling Nets", *Lecture Notes in Computer Science*, Springer Verlag, Berlin, Vol.995, 1995, pp.25-44

[BELHADJ,ET.AL.93]
H.Belhadj, L.Gerbaux, M.-C.Bertrand, G.Saucier, "Specification and Synthesis of Communicating Finite State Machines", w: G.Saucier, J.Trilhe, (red.), *Synthesis for Control Dominated Circuits*, Elsevier Science Publishers B.V., North-Holland, IFIP, ss.91-102, 1993

[BERNADINELLO,CINDIO92]
L.Bernadinello, F.De Cindio, "A Survey of Basic Net Models and Modular Net Classes", w: G.Rozenberg, (red.), *Lecture Notes in Computer Science*, Vol.609, "Advances in Petri Nets 1992", Springer-Verlag, Berlin, ss.304-351, 1992

[BEST,FERNANDEZ86]
E.Best, C.Fernandez, "Notations and Terminology on Petri Net Theory", *Petri Net Newsletter*, ss.21-46, 23.04.1986

[BEST87]
E.Best "Structural theory of Petri Nets: The free choice hiatus", *Lecture Notes in Computer Science*, Springer-Verlag, ss.168-206, 1987

[BIERE,ET.AL.99]
A.Biere, A.Cimatti, E.M.Clarke, M.Fujita, Y.Zhu, "Symbolic model checking using SAT procedures instead of BDDs", *Proceedings of Design Automation Conference (DAC'99*), New Orleans, USA, 1999

[BILIŃSKI96]
K.Biliński, *Application of Petri Nets in parallel controllers design*, PhD. Thesis, University of Bristol, Electrical and Electronic Engineering Department, Bristol, 1996

[BILIŃSKI,ET.AL.95]
K.Biliński, J.M.Saul, E.L.Dagless, "An Efficient Functional Verification Algorithm for Petri Net Based Parallel Controllers", *IEE Proceedings, Computers and Digital Technique*, Vol.142, No.6, ss.255-262, July 1995

[BOURRET02]
R.Bourret, „XML and Databases" http://www.rpbourret.com

[BOUSSIN79]
J.L.Boussin, "Synthesis and analysis of logic automation systems", *Proceedings of the Seventh Triennial World Congress of the International Federation of Automatic Control*, Helsinki, Finlandia, Vol.2, ss.1527-1535, June 1978

[BUCHHOLZ94]
P.Buchholz, "Hierarchical High Level Petri Nets for Complex System Analysis", w R.Valette, (red.), *Lecture Notes in Computer Science*, , Vol., "Proceedings of the 15th International Conference on Application and Theory of Petri Nets 1994", Zaragoza, Spain, ss.119-138, Springer-Verlag, 1994

[COMMONER72]
F.Commoner, *Deadlocks in Petri nets*, Applied Data Res. Inc, Wakefield, 1972

[CORTADELLA,ET.AL.02]
J.Cortadella, A.Yakovlev, G.Rozenberg, *Concurrency and hardware design*, Lecture Notes in Computer Science, Springer-Verlag, Berlin, Heidelberg, 2002

[DATTA,GHOSH84]
A.Datta, S.Ghosh, "Synthesis of a Class of Deadlock-Free Petri Nets", *Journal of the Association for Computing Machinery*, Vol.31, No.3, ss.486-506, July 1984

[DATTA,GHOSH86]
A.K.Datta, S.Ghosh, "Modular Synthesis of Deadlock-Free Control Structures", w: K.V.Nari, (red.), *Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science*, Vol.241, ss.288-318, Springer-Verlag, 1986

[DAVID,ALLA92]
R.David, H.Alla, *Petri Nets & Grafcet. Tools for modelling discrete event systems*, Prentice Hall, New York, 1992

[DOWLING, GALLIER76]
W.F.Dowling, J.H.Gallier: "Linear time algorithms for testing the satisfiability of propositional Horn formulae", *Journal of Logic Programming*, No.3, 1976, pp.691-703

[ESPERZA,SILVA91A]
J.Esperza, M.Silva, "Top-down Synthesis of Live and Bounded Free Choice Nets", *Lecture Notes in Computer Science*, Vol.524, ss.118-139, Springer Verlag, 1991

[ESPERZA,SILVA91B]
J.Esperza, M.Silva, "On the Analysis and Synthesis of Free Choice Systems", *Lecture Notes in Computer Science*, Vol.483, ss.243-286, Springer Verlag, 1991

[EZPELETA,ET.AL.93]
J.Ezpeleta, J.M.Couvreur, M.Silva, "A new technique for finding a generating family of siphon, traps and st-components. Application to colored Petri nets", *Lecture Notes in Computer Science*, Vol.691, ss.126-147, Springer-Verlag, 1993

[FEHLING94]
R.Fehling, "A concept of Hierarchical Petri Nets with Building Blocks", w: M.A. Marsan, (red.), *Lecture Notes in Computer Science*, "Proceedings of 14th International Conference on Application and Theory of Petri Nets 1994", Chicago, Illinois, USA, Vol.691, ss.148-166, Springer-Verlag, 1994

[FERRARINI92]
L.Ferrarini, "An Incremental Approach to Logic Controller Design with Petri Nets", *IEEE Transactions on System, Man, and Cybernetics,* Vol.22, No.3, ss.461-474, May/June 1992

[FERNANDES,ET.AL.97]
J.M.Fernandes, M.Adamski, A.Proenca, "VHDL generation from Petri net specification", IEE Proceedings, E. Computer and Digital Techniques, Vol.144, No.2, ss.127-137, 1997

[GIRAULT,VALK03]
C.Girault, R.Valk, *Petri nets for systems engineering*, Springer-Verlag Berlin Heilderberg, 2003

[GOLDBERG,NOVIKOV02]
E.Goldberg, Y.Novikov, "BerkMin: a fast and robust Sat-solver", *Proceedings of the Design, Automation and Test in Europe*, Paryż, Francja, 2002, ss.142-149

[HALANG89]
W.A.Halang, "Languages and tools for the graphical and textual system independent programming of programmable logic controllers", *Microprocessing and microprogramming*, North Holland, Vol.27, ss.583-590, 1989

[HAROLD00]
E.R.Harold, *XML. Księga eksperta*, Wydawnictwo Helion, Warszawa, 2000

[IEC92]
International Electrotechnical Commission, *International standard IEC 1131-3, Programmable Controllers, Part 3: Programming Languages*, Geneva, 1992

[JANICKI84]
R.Janicki, "Nets, Sequential Components and Concurrency Relations", *Theoretical Computer Science*, Vol. 29, Elsevier Science Publishers B.V., North Holland, ss.87-121, 1984

[JIANG,HOLDING96]
J.Jiang, D.J.Holding, "The Formalisation and Analysis of Sequential Function Charts using a Petri Net Approach", *Proceedings of the 13th World Congress of IFAC*, San Francisco, CA, USA, Vol.J - Discrete Event Systems, ss.513-518, 1996

[JENSEN92]
K.Jensen, *Coloured Petri Nets. Basic Concept, Analysis Methods and Practical Use*, *Volume 1, Basic Concepts*, Springer-Verlag, Berlin, 1992

[JENSEN94]
K.Jensen, *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, *Volume 2, Analysis Methods*, Monographs in Theoretical Computer Science, Springer-Verlag, Berlin, 1994

[JENSEN97]
K.Jensen, *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, *Volume 3, Practical Use*, Monographs in Theoretical Computer Science, Springer-Verlag, Berlin, 1997

[JENSEN,ROZENBERG91]
K.Jensen, G.Rozenberg, (red.), *High-level Petri Nets. Theory and Application*, Springer-Verlag, Berlin, 1991

[KARATKEVICH,ET.AL.00]
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

[KALINOWSKI84]
J.Kalinowski, *Wykorzystanie sieci Petriego do projektowania systemów cyfrowych*, Rozprawa doktorska, Politechnika Warszawska, Warszawa, 1984

[KOWALSKI89]
R.Kowalski, *Logika w rozwiązywaniu zadań*, Wydawnictwa Naukowo-Techniczne, Warszawa, 1989

[KOTOW84]
Е.Котов, *Сети Петри*, М.: Наука, Главная редакция физико-математической литературы, 1984

[KOZŁOWSKI95]
T.Kozłowski, "Petri Nets for State Machines", w: J.V.Oldfield, R.C.Dorf, *Field-Programmable Gate Arrays: Reconfigurable Logic for Rapid Prototyping and Implementation of Digital Systems*, ss.36-44; Wiley-Interscience, USA, (ISBN 0-471-55665-3), 1995

[KOZŁOWSKI,ET.AL.95]
T.Kozłowski, E.L.Dagless, J.M.Saul, M.Adamski, J.Szajna, "Parallel controller synthesis using Petri nets", *IEE Proceedings-E, Computers and Digital Techniques*, Vol.142, No.4, ss.263-271, July 1995

[LEE,FAVREL85]
K.H.Lee, J.Favrel, "Hierarchical Reduction Method for Analysis and Decomposition of Petri Nets", *IEEE Transactions on Systems, Man and Cybernetics*, Vol.SMC-15, No.2, ss.272-280, March-April 1985

[LEWIS95]
R.W.Lewis, *Programming industrial control systems using IEC 1131-3,* The Institution of Electrical Engineers, London, 1995

[LI,WOODSIDE95]
Y.Li, C.M.Woodside, "Complete Decomposition of Stochastic Petri Nets Representing Generalized Service Networks", *IEEE Transactions on Computers*, Vol.44, No.4, ss.577-592, April 1995

[LYNGSO,MAILUND98]
R.B.Lyngso, T.Mailund, "Textual Interchange Format for High-Level Petri Nets", *Workshop on Practical Use of Coloured Petri Nets and Design*, June 1998, Aarhus University, ss.47-64

[ŁABIAK01A]
G.Łabiak, "Statecharts diagram as linked hierarchical sequential automata", *Computer - Aided Design of Discrete Devices - CAD DD 2001, Proceedings of the Fourth International Conference*, Mińsk, Białoruś, 2001, ss.38-43

[ŁABIAK01B]
G.Łabiak, "Symbolic state exploration of controllers specified by means of Statecharts", *Discrete - Event System Design - DESDes '01, Proceedings of the International Workshop*, Przytok k/Zielonej Góry, Polska, 2001, ss.209-214

[ŁUBA,ZBIERZCHOWSKI02]
T.Łuba, B.Zbierzchowski, Układy logiczne, *Wyższa Szkoła Informatyki Stosowanej i Zarządzania*, Warszawa 2002

[MACHADO,ET.AL.97]
R.J.Machado, J.M.Fernandes, A.J.Proença, "SOPHIA: A CAD Environment to Design Digital Control Systems", *XIII IFIP Conference on Computer Hardware Description Languages and Their Applications CHDL'97*, Toledo, Spain, 20-25.04.1997

[MAJEWSKI,ŁUBA86]
Praca zbiorowa po kierunkiem W.Majewskiego i T.Łuby: *Cyfrowe układy telekomunikacyjne - Podstawy teoretyczne i zasady syntezy*, Wydawnictwo Komunikacji i Łączności, Warszawa 1986

[MAJEWSKI98]
W.Majewski, *Układy logiczne. Wybrane Zagadnienia*, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 1998

[MATHONY89]
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

[MINOUX88]
M.Minoux, "LTUR: a simplified linear-time unit resolution algorithm for Horn formulae and computer implementation", *Information Processing Letters*, Elsevier Science Publishers (North Holland), Vol.29, No.1, 1988, pp.1-12

[MINOUX, BARKAOUI90]
M.Minoux, K.Barkaoui, "Deadlocks and traps in Petri nets as a Horn-satisfability solutions and some related polynomially solvable problems", *Discrete Applied Mathematics*, Elsevier Science Publishers (North Holland), Vol.29, 1990, pp.195-210

[MISIUREWICZ78]
P.Misiurewicz, "Sieci Petriego i ich zastosowanie do projektowania dyskretnych układów sterowania", *Zeszyty Naukowe Politechniki Śląskiej*, Automatyka, z.43, Gliwice, 1978

[MURATA89]
T.Murata, "Petri Nets: Properties, Analysis and Applications", *Proceedings of the IEEE*, Vol.77, No.4, ss.541-580, April 1989

[NOTOMI,MURATA94]
M.Notomi, T.Murata, "Hierarchical Reachability Graph of Bounded Petri Nets for Concurrent-Software Analysis", *IEEE Transactions on Software Engineering*, Vol.20, No.5, ss.325-336, May 1994

[NOVIKOV,GOLDBERG01]
Y.Novikov, E.Goldberg, "An efficient learning procedure for multiple implication checks", *Proceedings of the Design, Automation and Test in Europe*, Monachium, Niemcy, 2001, ss.127-133

[PAPADIMITRIOU02]
Ch.H.Papadimitriou, *Złożoność obliczeniowa*, Wydawnictwa Naukowo-Techniczne, Warszawa, 2002

[PARDEY,BOLTON91]
J.Pardey, M.Bolton, "Logic Synthesis of Synchronous Parallel Controllers", *Proceedings of the IEEE International Conference on Computer Design*, ss.454-457, IEEE Computer Society Press, 1991

[PARDEY,ET.AL.92]
J.Pardey, T.Kozlowski, J.Saul, M.Bolton, "State Assignment Algorithms for Parallel Controllers Synthesis", *Proceedings of the IEEE International Conference on Computer Design*, IEEE Computer Society Press, ss.316-319, 1992

[PASTOR,ET.AL.94]
E.Pastor, O.Roig, J.Cortadella, R.Badia, "Petri Net Analysis Using Boolean Manipulation", w: R.Valette, *Lecture Notes in Computer Science*, Vol.815, "Proceedings of the 15th International Conference on Application and Theory of Petri Nets", ss.416-435, Springer-Verlag, 1994

[PENCZEK,ET.AL.02]
W.Penczek, B.Woźna, A.Zbrzeżny, "*Branching Time Bounded Model Checking for Elementary Net Systems*", Raport IPI PAN 940, Warszawa, styczeń 2002

[PETERSON81]
J.L.Peterson, *Petri Net Theory and The Modeling of Systems*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1981

[PETRI62]
C.A.Petri, *Kommunikation mit automaten*, Institut fur Instrumentelle Mathematik, Bonn, 1962

[REISIG88]
W.Reisig, *Sieci Petriego. Wprowadzenie*, Wydawnictwa Naukowo-Techniczne, Warszawa, 1988

[SACHA]
http://www.ia.pw.edu.pl/~sacha/petri.html

[SKOWROŃSKI00]
Z.Skowroński, *Translacja specyfikacji funkcjonalnej układów cyfrowych na sieć Petriego dla potrzeb syntezy systemowej*, Rozprawa doktorska, Politechnika Szczecińska, Szczecin, 2000

[STARKE87]
P.H.Starke, *Sieci Petri - podstawy, zastosowania, teoria*, PWN, Warszawa, 1987

[STEWART,ET.AL.91]
J.Stewart, E.Dagless, D.Milford, O.Miles, "A Petri Net Based Framestore", *Proceedings of the International Workshop on Field Programmable Logic and Applications*, ss.332-342, 1991

[SURAJ,KOMAREK94]
Z.Suraj, B.Komarek*, GRAF. System graficznej konstrukcji i analizy sieci Petriego*, Akademicka Oficyna Wydawnicza PLJ, Warszawa, 1994

[SURAJ,SZPYRKA99]
Z.Suraj, M.Szpyrka*, Sieci Petriego i PN-Tools*, Wydawnictwo Wyższej Szkoły Pedagogicznej, Rzeszów, 1999

[TARJAN72]
R.Tarjan, "Depth-first search and linear graph algorithms", *SIAM. Jour. Comp.*, Vol.2, 1972

[THELEN88]
B.Thelen, *Investigations of algorithms for computer-aided logic design of digital circuits*, (język niemiecki), PhD dissertation, ITIV, Univ. of Karlsruhe, 1988

[TOULOTTE,PARSY79]
J.M.Toulotte, J.P.Parsy, "A method for decomposing interpreted Petri nets and its utilization", *Digital Processes*, No.5, ss.223-234, 1979

[TRACZYK01]
T.Traczyk, "Czy już warto używać XML", *I Seminarium PLOUG*, Warszawa, marzec 2001

[VALETTE79]
R.Valette, "Analysis of Petri nets by stepwise refinements", *Journal of Computer and System Science*, Vol.18, ss.35-36, 1979

[VENKATESH,ET.AL.94]
K.Venkatesh, M.Zhou, R.J.Caudill, "Comparing ladder logic diagrams and Petri nets for sequence controller design through a discrete manufacturing system", *IEEE Transactions on Industrial Electronics*, Vol.41, No.6, ss.611-619, December 1994

[WAGNER,RAKOWSKI81]
F.Wagner, M.Rakowski, "Zastosowanie sieci Petri do projektowania mikroprogramowanych układów sekwencyjnych", *Archiwum Automatyki i Telemechaniki*, Tom XXVI, Zeszyt 3, ss. 385-395, 1981

[WEBER,KINDLER02]
M.Weber, E.Kindler, "The Petri Net Markup Language", w: H.Ehrig, W.Reisig, G.Rozenberg, H.Weber (eds.), *Petri Net Technology for Communication Based Systems*, Lecture Notes in Computer Science, Springer 2002

[WĘGRZYN96]
M.Węgrzyn, "Implementacja współbieżnych kontrolerów cyfrowych z wykorzystaniem PLD", *Kwartalnik Elektroniki i Telekomunikacji PAN*, Tom 42, Zeszyt 2, czerwiec 1996, ss.235-251

[WĘGRZYN98]
M.Węgrzyn, *Hierarchiczna implementacja współbieżnych kontrolerów cyfrowych z wykorzystaniem FPGA*, Rozprawa doktorska, Politechnika Warszawska, Warszawa, 1998

[WĘGRZYN01]
A.Węgrzyn, "Symbolic verification of concurrent logic controllers by means of Petri nets", *International PHD Students' Workshop Control & Information Technology, IWCIT'01*, Ostrava, Czechy, 19–20.09.2001, ss.198-203

[WĘGRZYN01A]
A.Węgrzyn, "Zastosowanie języka XML do regułowego opisu sieci Petriego", *Raport Nr UZ/ZIK-01/2001*, Uniwersytet Zielonogórski, 2001

[WĘGRZYN02]
A.Węgrzyn, "Concurrent controller design by means of high-level Petri nets", *Radioelektronika i Informatyka*, Charków, Ukraina, 2002, ss.71-76

[WĘGRZYN02A]
A.Węgrzyn, "New approach for logic controller modelling"*, Proceedings of the 5th Portuguese Conference on Automatic Control*, Aveiro, Portugalia, 05-07.09.2002, ss. 219-224

[WĘGRZYN,ADAMSKI99]
M.Wegrzyn, M.Adamski, "Hierarchical Approach for Design of Application Specific Logic Controller", *IEEE International Symposium on Industrial Electronics ISIE'99*, Bled, Slovenia, 12-16.07.1999, Vol.3, pp.1389-1394, 1999

[WĘGRZYN,BUBACZ01]
A.Węgrzyn, P.Bubacz, "XML application for modelling and simulation of concurrent controllers", *The International Workshop on Discrete-Event System Design, DESDes'01*, Przytok, 27-29.06.2001 ss.215-221

[WĘGRZYN,BUBACZ01A]
A.Węgrzyn, P.Bubacz, "XML Format for High-Level Petri Net", *Advanced Computer Systems, ACS'01*, Mielno, 17-19.10.2001, ss.269-278

[WĘGRZYN,CHODAŃ00]
A.Wegrzyn, M.Chodan, "Formal analysis of SFC nets", *Advanced Computer Systems, ACS'2000*, Szczecin, 23-25.10.2000

[WĘGRZYN,KARATKIEVICH00]
A.Węgrzyn, A.Karatkievich, "Использование Раскрашенных Сетей Петри Для Моделирования И Верификации Параллельных Устройств Логического Управления", *Рабиозлектроника и информатика*, Харьков, Украина, 2000, ss.70-74

[WĘGRZYN,WĘGRZYN98]
A.Węgrzyn, M.Węgrzyn, "Modelling, Simulation and Analysis of Concurrent Controllers by Means of Coloured Petri Nets and Design/CPN System", *Design and Diagnostics of Electronic Circuits and Systems Workshop*, Szczyrk, 02-04.09.1998, Szczyrk,

[WĘGRZYN,WĘGRZYN99]
A.Wegrzyn, M.Wegrzyn, "Symbolic verification of concurrent logic controllers by means of Petri nets", *The Third International Conference on Computer-Aided Design of Discrete Devices - CAD DD'99*, 10-12.11.1999, Minsk, Belarus, Vol.1, pp.45-50

[WĘGRZYN,WĘGRZYN00]
A.Węgrzyn, M.Węgrzyn, "Petri net-based Specification, Analysis and Synthesis of Logic Controllers", *IEEE International Symposium on Industrial Electronics ISIE'2000*, Puebla, Mexico, 04-08.12.2000, ss.20-26

[WOLAŃSKI98]
P.Wolański, *Modelowanie układów cyfrowych na poziomie RTL z wykorzystaniem sieci Petriego i podzbioru języka VHDL*, Rozprawa doktorska, Politechnika Warszawska, Warszawa, 1998

[W3.ORG]
http://www.w3.org

[ZAKREVSKIJ85]
А.Д.Закревкий, "К проверке живости ордиарных сетей Петри", *Доклады Академии Наук БССР*, Том XXIX, No.11, ss.1006-1009, 1985

[ZAKREVSKIJ87]
A.D.Zakrevskij, "The analysis of concurrent logic control algorithms", *Lecture Notes in Computer Science*, vol.278, Springer Verlag, 1987, ss.497-500

[ZAKREVSKIJ99]
A.D.Zakrevskij The analysis of concurrent logical control algorithms.- LNCS, 278. Fundamentals of Computation Theory.- Springer Verlag, 1987.- P. 497-500.A.Zakrevskij, "High-level design of logical control devices", *Proceedings of the 3rd International Conference CAD DD'99*, Minsk, 1999, ss.13-18

[ŻURAWSKI,ZHOU94]
R.Zurawski, M.C.Zhou, "Petri Nets and Industrial Applications: A Tutorial", *IEEE Transactions on Industrial Electronics*, Vol.41, No.6, Dec.1994, ss.567-58