*Marcin Szpyrka**

FORMALNA WERYFIKACJA WYMAGAŃ SYSTEMÓW CZASU RZECZYWISTEGO

1. Wprowadzenie

Analiza wymagań jest pierwszym etapem w tworzeniu systemów czasu rzeczywistego (zob. [2], [3], [5], [6], [7], [8], [9], [10]). Jej wynikiem powinien być ścisły, zrozumiały i poprawny model rozważanego systemu. Model wynikający z analizy służy kilku celom: klaruje wymagania dotyczące systemu, dostarcza podstaw do porozumienia pomiędzy osobą zgłaszającą potrzebę utworzenia systemu oraz osobą, która ten system będzie rozwijać i stanowi podstawę dla dalszego projektowania i implementacji systemu.

Analiza rozpoczyna się od ogólnego sformułowania problemu przez klienta. Wymagania dotyczące omawianego systemu mogą być niekompletne i przedstawione w sposób nieformalny. Dotyczą one w dużej mierze tej części zachowania się systemu, która jest obserwowana z zewnątrz. Analiza wymagań ma za zadanie uczynić je bardziej dokładnymi oraz wskazać wszelkie niejasności i niekonsekwencje. Aby to zrobić konieczne jest przetestowanie wymagań, analiza wniosków i ponowne ich sformułowanie.

Zazwyczaj osoba zgłaszająca potrzebę utworzenia konkretnego systemu nie jest specjalistą z zakresu inżynierii oprogramowania i przedstawiane przez siebie wymagania formułuje w języku naturalnym, który jest z natury dwuznaczny. Zapis specyfikacji wymagań z użyciem tylko i wyłącznie języka naturalnego prowadzi z reguły do opisu, który z jednej strony może być łatwy do zrozumienia, zaś z drugiej jest mało przydatny do analizy jego własności, takich jak zupełność czy też niesprzeczność takiego opisu.

Analiza nie jest procesem mechanicznym. W większości przypadków konieczne jest komunikowanie się analityka z osobą zgłaszającą potrzebę zbudowania systemu, w celu usunięcia wszelkich niejasności i braków. Potrzebny jest więc taki sposób zapisu wymagań stawianych przez klienta, by z jednej strony były one przydatne przy analizie ich własności, zaś z drugiej strony zrozumiałe dla osoby je formułującej. Zdaniem autora wśród wielu istniejących metod formalnego zapisu wymagań (zob. [4]), najlepiej do tego celu nadają się

* Katedra Automatyki, Akademia Górniczo-Hutnicza, Kraków

tablice decyzyjne. Niepodważalną zaletą tej formy zapisu jest możliwość zrozumienia go przez osoby całkowicie niezwiązane z zagadnieniami analizy i projektowania systemów czasu rzeczywistego. Dla analityka natomiast mogą one stanowić podstawę do analizy poprawności wstępnie sformułowanych wymagań.

Przez poprawnie utworzoną specyfikację wymagań dotyczących zachowania się systemu będziemy rozumieć formalny kompletny opis pozbawiony niejasności i dwuznaczności. Oznacza to, że system powinien „potrafić" jednoznacznie zareagować w każdej możliwej sytuacji wywołanej przez czynniki zewnętrzne i jednocześnie opis taki powinien być możliwie zwięzły, tzn. pozbawiony zbędnych elementów.

Problemem pozostaje wybór narzędzi, jakimi posłużymy się przy analizie specyfikacji sformułowanej w postaci tablicy decyzyjnej. W poniższym artykule do tego celu wykorzystano kolorowane sieci Petriego. Wybór ten można uzasadnić faktem, iż przedstawienie wymagań w postaci sieci Petriego pozwala na łatwą analizę wspomnianych wcześniej ich własności. Poza tym sieci są formą zapisu przydatną nie tylko w fazie analizy wymagań, ale również w fazie projektowania systemu, natomiast dostępne narzędzia komputerowe pozwalają w łatwy sposób konstruować i analizować stworzone sieci, w pełni automatyzując etap ich analizy. W dalszej części pracy można znaleźć algorytm przekształcający tablicę decyzyjną w kolorowaną sieć Petriego oraz zbiór twierdzeń, które pozwalają łączyć własności tak zbudowanej sieci ze wspomnianymi wcześniej własnościami dotyczącymi specyfikacji wymagań.

Należy jeszcze podkreślić fakt, że budowa modelu, a później systemu czasu rzeczywistego w oparciu o specyfikację zawierającą błędy powoduje, że ich usuwanie na dalszych etapach tworzenia systemu, powoduje znaczny wzrost kosztów. Zatem wyeliminowanie wszelkich niejasności przed rozpoczęciem konstruowania modelu może przyspieszyć jego powstanie i zmniejszyć koszty jego budowy.

2. Tworzenie specyfikacji wymagań

2.1. Opis przykładu

Przy przedstawianiu metody tworzenia specyfikacji zachowania się systemu wykorzystany zostanie przykład uproszczonego sterownika windy (zob. [8]), obsługującej budynek o wysokości n pięter, (bez utraty ogólności rozważań można przyjąć, że w rozpatrywanym przypadku $n = 5$). Zatem poszczególne poziomy mogą być oznaczone liczbami całkowitymi od 0 do 5, gdzie 0 oznacza parter.

Dla uproszczenia przyjmiemy, że fakt zgłoszenia chęci jazdy w danym kierunku (góra, dół) oraz moment wyboru docelowego poziomu odbywają się jednocześnie, np. ktoś na drugim piętrze zgłasza potrzebę jazdy na piąte piętro (zadanie takie będzie oznaczane za pomocą pary liczb (2, 5)). Przyjmiemy założenie, że punktem startowym windy jest poziom 0. Jeżeli sterownik otrzyma jakieś zgłoszenia, to winda startuje w górę i obsługuje wszystkie zadania związane z tym kierunkiem jazdy, aż dotrze do ostatniego piętra, np. sterownik windy otrzymał zadania (1, 4), (2, 4); wówczas winda porusza się w górę, zatrzymuje się na piętrze pierwszym, następnie na drugim i czwartym, na zakończenie przejeżdża do punktu końcowego (piąte piętro) i rozpoczyna się obsługa zgłoszeń związanych z jazdą

w dół. Jeżeli w czasie jazdy w górę pojawia się zgłoszenie potrzeby jazdy w kierunku przeciwnym, to jest ono tymczasowo ignorowane.

Z przedstawionego opisu widać, że taki sposób pracy windy nie jest „doskonałym". Może się bowiem zdarzyć, że sterownik otrzyma tylko jedno zgłoszenie (0, 1), a winda i tak zostanie w efekcie przemieszczona na ostatnie piętro i z powrotem. Uproszczenia te przyjęte zostały na potrzeby poniższej pracy, gdyż dokładne rozpatrzenie wszystkich aspektów rozważanego systemu spowodowałoby, że stałby się on bardziej skomplikowany.

2.2. Opis zachowania się systemu

Zasady pracy sterownika windy można opisać w sposób werbalny, używając potocznego języka. Opis taki będzie wówczas zrozumiały dla dowolnej osoby, zarówno zgłaszającej potrzebę utworzenia odpowiedniego systemu, jak i dla analityka czy projektanta. Jednak w takiej sytuacji trudno będzie sprawdzić, czy przedstawiona specyfikacja zachowania się systemu jest kompletna i czy nie zawiera ona błędów. Postać taka może być jednak przydatna, choć nie jest konieczna, do tworzenia bardziej sformalizowanego opisu.

Poniżej przedstawiona zostanie próba opisania w ten sposób pracy rozważanego sterownika windy. Opis taki będzie składał się z ciągu zdań mających postać implikacji.

- Jeżeli winda znajduje się w pozycji wyjściowej i otrzymała jakiekolwiek zgłoszenie, to rozpoczyna ona jazdę w górę.
- Jeżeli winda jadąc w górę zbliża się do piętra k i otrzymała wcześniej zgłoszenie związane z tym piętrem, zgodne z kierunkiem jazdy windy (ktoś chce na tym piętrze wysiąść z windy lub do niej wsiąść, by jechać w górę), to powinna się na tym piętrze zatrzymać.
- Jeżeli winda, jadąc w górę, zbliża się do piętra k, nie jest to ostatnie piętro i nie otrzymała do tej pory żadnego zgłoszenia związanego z tym piętrem zgodnego z obecnym kierunkiem jazdy, to powinna kontynuować jazdę w górę bez zatrzymywania się na tym piętrze.
- Jeżeli winda, jadąc w dół, zbliża się do piętra k i otrzymała wcześniej zgłoszenie związane z tym piętrem, zgodne z kierunkiem jazdy windy, to powinna się na tym piętrze zatrzymać.
- Jeżeli winda, jadąc w dół, zbliża się do piętra k, nie jest to parter i nie otrzymała do tej pory żadnego zgłoszenia związanego z tym piętrem zgodnego z obecnym kierunkiem jazdy, to powinna kontynuować jazdę w dół bez zatrzymywania się na tym piętrze.
- Jeżeli winda w czasie obsługi zadań związanych z ruchem w górę stoi na piętrze k i nie jest to ostatnie piętro, to powinna ona następnie kontynuować jazdę w górę.
- Jeżeli winda w czasie obsługi zadań związanych z ruchem w dół stoi na piętrze k i nie jest to parter, to powinna ona następnie kontynuować jazdę w dół.
- Jeżeli winda stoi na ostatnim piętrze, to następnie rozpoczyna obsługę zadań związanych z jazdą w dół.

Otrzymaną w ten sposób listę zdań można traktować jako pewien opis zewnętrznego zachowania się rozważanego systemu. Tymczasowo nie będą rozważane kwestie związane z zupełnością czy też poprawnością takiego opisu. Powyższy opis zostanie natomiast przedstawiony w postaci tablicy decyzyjnej. Będzie ona podstawą do skonstruowania kolorowanej

sieci Petriego, której analiza pozwoli na udzielenie odpowiedzi na pytania dotyczące własności przedstawionego opisu.

W oparciu o przedstawioną specyfikację zachowania się systemu, można wyróżnić kilka atrybutów, których wartości wpływają na podejmowanie odpowiednich decyzji. Będą to tzw. **atrybuty warunkowe**.

- Zachowanie się windy zależy od aktualnego jej stanu, tzn. czy porusza się ona w górę, w dół, czy też stoi. Atrybutowi temu nadamy nazwę Stan. Zbiór wartości atrybutu Stan jest trzyelementowy V_{Stan} = {up, down, stop}.
- Drugim atrybutem warunkowym może być aktualne położenie windy. Atrybutowi temu nadamy nazwę Poziom. W tym przypadku V_{Poziom} = {0, 1, ..., 5} jest zbiorem $n + 1$ elementowym. Wartości tego atrybutu mogą być interpretowane w sposób następujący. Jeżeli Poziom = 3 i Stan = stop, to oznacza, że winda stoi na piętrze trzecim. Jeżeli Poziom = 3 i Stan = up, oznacza to, że winda, jadąc w górę, zbliża się do piętra trzeciego i rozpatrywana decyzja będzie dotyczyć ewentualnego zatrzymania się na tym piętrze. Analogicznie interpretuje się sytuację w przypadku ruchu w dół.
- Łatwo zauważyć, że przy podejmowaniu decyzji rozpatrywany jest stan sensora na jednym tylko piętrze, np. przy zbliżaniu się do pierwszego piętra istotne są tylko zgłoszenia postaci (1, i), gdzie i = 0, 1, ..., 5. Dokładniej, ważny jest tylko fakt, czy na tym piętrze wystąpiło co najmniej jedno zgłoszenie zgodne z kierunkiem jazdy windy lub czy w związku ze zgłoszeniem na innym poziomie, na tym piętrze należy się zatrzymać. Ponieważ, w danym momencie, istotny będzie tylko stan sensora na jednym z pięter, to możemy pierwszy element pary pominąć. Rozpatrywanemu atrybutowi nadamy nazwę Sensor, zaś zbiór wartości tego atrybutu jest postaci V_{Sensor} = {0, 1, ..., 5, 6}. Należy jeszcze wyjaśnić, że np. wartość Sensor = 2 na piętrze drugim oznacza po prostu, że na tym piętrze należy się zatrzymać w związku ze zgłoszeniem na innym poziomie, natomiast wartość 6 jest wartością dodatkową i oznacza, że nie ma żadnego zgłoszenia związanego z tym piętrem.
- Kolejny z atrybutów nazwiemy Kierunek. Będzie on oznaczał, czy w danym momencie winda obsługuje zadania związane z jazdą w górę, czy też z jazdą w dół. Stąd też dwuelementowy zbiór $V_{Kierunek}$ = {dup, ddown}.
- Ostatni z atrybutów warunkowych nazwiemy Zadanie. W tym przypadku zbiór wartości $V_{Zadanie}$ = {yes, no}. Wartość Zadanie = yes oznaczać będzie, że wystąpiło co najmniej jedno zgłoszenie. Fakt ten będzie sygnałem do rozpoczęcia cyklu obsługi zgłoszeń w sytuacji, gdy winda będzie znajdować się w pozycji wyjściowej. Druga z możliwych wartości Zadanie = no odpowiada sytuacji, w której nie ma żadnych zgłoszeń. Łatwo zauważyć, że wartości tego atrybutu nie mają wpływu na podejmowane decyzje w sytuacji, gdy winda jest w trakcie obsługi zgłoszeń.

Oprócz atrybutów warunkowych, w rozważanym systemie można wyróżnić również **atrybuty decyzyjne**, których wartości będą odzwierciedlać podejmowane decyzje.

- Atrybut NStan oznaczać będzie nowy stan windy, tzn. czy winda ma się zatrzymać, jechać w górę, czy też jechać w dół. Zbiór wartości tego atrybutu pokrywa się ze zbiorem wartości atrybutu Stan, tzn. $V_{NStan} = V_{Stan}$.
- Atrybut NKierunek oznaczać będzie nowy kierunek jazdy windy. Zbiór wartości tego atrybutu pokrywa się za zbiorem wartości atrybutu Kierunek, tzn. $V_{NKierunek} = V_{Kierunek}$.

Po wyróżnieniu wszystkich atrybutów można przystąpić do utworzenia **tablicy decyzyjnej** (tab. 1). Tablica ta będzie miała liczbę wierszy i kolumn zależną od liczby wszystkich atrybutów i liczby reguł. Każdemu atrybutowi będzie odpowiadać jedna kolumna, zaś regule jeden wiersz. Przecięcie kolumny i wiersza będzie zawierało: wartość danego atrybutu w odpowiedniej regule, jeżeli jest to wartość unikatowa; zmienną o zakresie wartości równym zbiorowi dopuszczalnych wartości danego atrybutu, jeżeli wartość danego atrybutu będzie nieistotna w rozpatrywanej regule; wyrażenie logiczne ograniczające zakres zmiennej, jeżeli dopuszczalnych jest kilka wartości, lecz nie wszystkie. Dla większej przejrzystości zapisu zmienne będą pisane kursywą.

Tabela 1

	Stan	Poziom	Sensor	Kierunek	Zadanie	NStan	NKierunek
r_1	stop	0	s	dup	yes	up	dup
r_2	up	p	$(s \geq p) \wedge (s < 6)$	dup	z	stop	dup
r_3	up	$p < 5$	$(s < p) \vee (s = 6)$	dup	z	up	dup
r_4	down	p	$(s \leq p)$	ddown	z	stop	ddown
r_5	down	$p > 0$	$(s > p)$	ddown	z	down	ddown
r_6	stop	$(p < 5) \wedge (p > 0)$	s	dup	z	up	dup
r_7	stop	$p > 0$	s	ddown	z	down	ddown
r_8	stop	5	s	ddown	z	down	ddown

Można zauważyć, że w wielu regułach niektóre z atrybutów nie odgrywają żadnej roli, np. atrybut K wpływa na wartość decyzji tylko w regułach r_6 i r_7. W pozostałych jednak przypadkach, mimo że nie ma on znaczącej roli w podejmowaniu decyzji, wartość takiego atrybutu jest ściśle określona. Dlatego też wartości te będą brane pod uwagę w konstruowaniu sieci Petriego. W przypadku natomiast pojawienia się sprzecznego układu wartości atrybutów, może to być traktowane jako awaria systemu. W dalszej części pokazany zostanie sposób na uniknięcie takich sytuacji.

W dalszych rozważaniach używane będą następujące oznaczenia związane z zapisem wymagań zachowania się systemu przy użyciu tablicy decyzyjnej:

A_w oznaczać będzie zbiór atrybutów warunkowych,

A_d oznaczać będzie zbiór atrybutów decyzyjnych,

R oznaczać będzie zbiór reguł decyzyjnych.

Niech $u \in A_w \cup A_d$. Symbol V_u oznaczać będzie zbiór dopuszczalnych wartości atrybutu u, zaś symbol W_u zbiór zmiennych o dopuszczalnym zbiorze wartości V_u.

Symbol L_u oznaczać będzie zbiór wszystkich wyrażeń logicznych powstałych z elementów zbiorów V_u i W_u, przy użyciu symboli relacyjnych $\{=, \geq, >, \leq, <, \neq\}$ oraz symboli logicznych $\{\neg, \wedge, \vee\}$.

Niech $u \in A_w \cup A_d$ i $r_i \in R$. Symbol $Tb(u, r_i)$ oznaczać będzie zawartość tej komórki tablicy decyzyjnej, która dotyczy atrybutu u i reguły r_i. W odniesieniu do tablic decyzyjnych przyjmiemy, że $\bigwedge_{u \in A_w \cup A_d} \bigwedge_{r \in R} Tb(u,r) \in V_u \cup W_u \cup L_u$.

W odniesieniu do przedstawionego przykładu poszczególne elementy są postaci:

- A_w = {Stan, Poziom, Sensor, Kierunek, Zadanie};
- A_d = {NStan, NKierunek};
- $R = \{r_1, r_2, ..., r_8\}$;

np.: $Tb(\text{Poziom}, r_1) = 0 \in V_{Stan}$,

$Tb(\text{Sensor}, r_6) = s \in W_{Sensor}$,

$Tb(\text{Poziom}, r_3) = p < 5 \in L_{Stan}$.

2.3. Konstrukcja sieci Petriego

W oparciu o przedstawioną wcześniej tabelę można w prosty sposób zbudować kolorowaną niehierarchiczną sieć Petriego (zob. [5]), służącą to podejmowania decyzji w określonych sytuacjach. Sieć taka będzie nazywana **kolorowaną siecią decyzyjną (D-siecią)** rozważanego systemu.

Definicja 1

Kolorowaną siecią decyzyjną (D-siecią) opisującą reguły zewnętrznego zachowania się systemu, nazywamy niehierarchiczną kolorowaną sieć Petriego, zbudowaną na podstawie tablicy decyzyjnej według następującego algorytmu.

1) W sieci umieszczamy dokładnie tyle miejsc ile atrybutów wyróżniono w tablicy decyzyjnej. Miejsca odpowiadające atrybutom warunkowym będziemy nazywać **miejscami warunkowymi**, zaś miejsca odpowiadające atrybutom decyzyjnym **miejscami decyzyjnymi**.
2) Miejscom D-sieci nadajemy nazwy zgodne z nazwami odpowiadających im atrybutów.
3) Przyjmujemy kolory poszczególnych miejsc równe zbiorom wartości odpowiadających im atrybutów.
4) Znakowanie początkowe D-sieci określamy następująco: w miejscach warunkowych umieszczamy po jednym znaczniku odpowiedniego koloru, zaś miejsca decyzyjne pozostawiamy puste. Należy zwrócić uwagę, by rozkład znaczników w miejscach warunkowych był dopuszczalny (np. nie może być Stan = up i Kierunek = ddown).
5) W sieci umieszczamy dokładnie tyle tranzycji, ile reguł decyzyjnych wyróżniono w tabeli decyzyjnej. Tranzycje takie nazywamy **tranzycjami decyzyjnymi**. Nadajemy im nazwy np.: $r_1, r_2, ..., r_n$.
6) Dla każdej z tranzycji r_i prowadzimy łuki wejściowe od każdego z miejsc warunkowych do tej tranzycji oraz łuki wyjściowe do każdego z miejsc decyzyjnych.
7) Wagi łuków prowadzących od miejsc warunkowych do tranzycji decyzyjnych ustalamy w sposób następujący: jeżeli w odpowiedniej komórce tabeli znajduje się stała wartość, np. Poziom = 0, to wartość ta będzie wagą danego łuku; jeżeli w odpowiedniej komórce tabeli znajduje się zmienna, np. Poziom = p, to zmienna ta będzie wagą dane-

go łuku; jeżeli w odpowiedniej komórce tabeli znajduje się warunek logiczny, np. $p \neq 5$, to wagą łuku będzie zmienna odpowiedniego koloru występująca w tym warunku, zaś warunek umieszczamy w zastrzeżeniu odpowiedniej tranzycji.

8) Wagi łuków prowadzących od tranzycji decyzyjnych do miejsc decyzyjnych ustalamy w sposób analogiczny, przy czym w tej sytuacji mogą zajść tylko dwa pierwsze przypadki.
9) W sieci umieszczamy dodatkową tranzycję zwaną **tranzycją odnawiającą**.
10) Prowadzimy łuki od miejsc decyzyjnych do tranzycji odnawiającej i od tej tranzycji do miejsc warunkowych.
11) Jako wagi wszystkich łuków wejściowych i wyjściowych tranzycji odnawiającej przyjmujemy zmienne (struktury zmiennych) odpowiednich typów.
12) Przy tranzycji odnawiającej umieszczamy zastrzeżenie tak, by w miejscach warunkowych nie wystąpił niedopuszczalny rozkład znaczników.

Otrzymujemy w ten sposób kolorowaną sieć Petriego, której poszczególne elementy są określone w sposób następujący (zob. [5]):

1) $\sum = \bigcup_{u \in A_w \cup A_d} V_u$
2) $P = A_w \cup A_d$
3) $T = R \cup \{\text{TOdn}\}$
4) A jest zbiorem $|A_w \cup A_d| * (|R|+1)$ łuków takim, że:

   $N(A) = (A_w \times R) \cup (R \times A_d) \cup (A_d \times \{\text{Todn}\}) \cup (\{\text{Todn}\} \times A_w)$
5) $\bigwedge_{p \in P} C(p) = V_p$.
6) $$\bigwedge_{r \in R} G(r) = \begin{cases} \prod_{\substack{u \in A_w \\ Tb(u,r) \in L_u}} Tb(u, r) & : \quad \bigvee_{u \in A_w} Tb(u, r) \in L_u \\ \text{True} & : \quad \bigwedge_{u \in A_w} Tb(u, r) \in V_u \cup W_u \end{cases}$$

   G(TOdn) zostanie omówione w dalszej części.
7) $$\bigwedge_{u \in A_w} \bigwedge_{r \in R} E((u, r)) = \begin{cases} Tb(u, r) & : \quad Tb(u, r) \in W_u \cup W_u \\ x, \ x \in \mathrm{W_u} & : \quad Tb(u, r) \in L_u \end{cases}$$

   $$\bigwedge_{u \in A_d} \bigwedge_{r \in R} E((r, u)) = Tb(u, r)$$

   $$\bigwedge_{u \in A_d} E((u, TOdn)) = x, x \in W_u$$

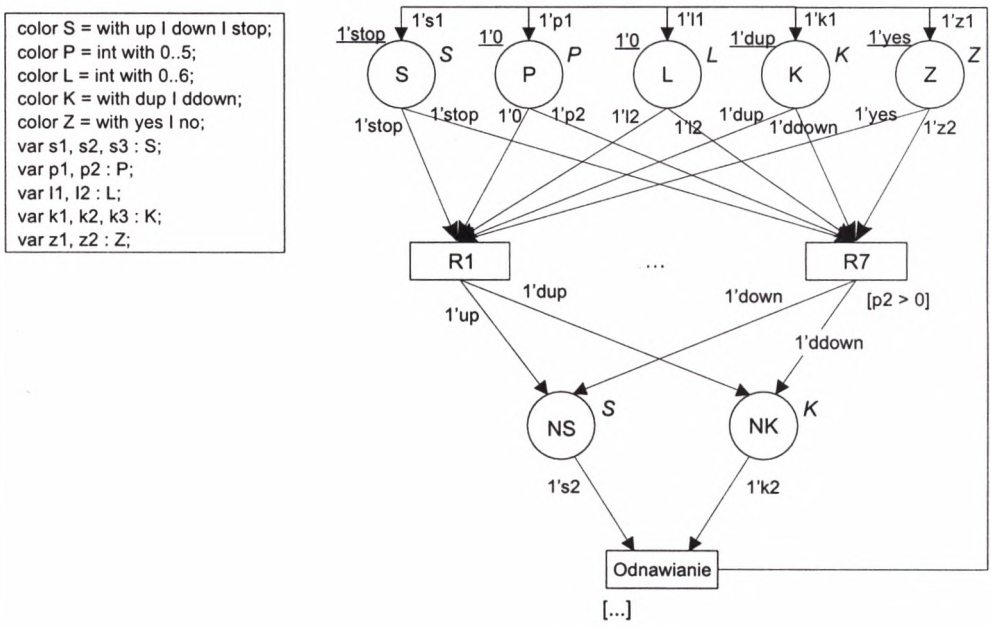
   $$\bigwedge_{u \in A_w} E((TOdn, u)) = x, x \in W_u .$$
8) $$\bigwedge_{u \in A_w} I(u) = x, x \in V_u$$

   $$\bigwedge_{u \in A_d} I(u) = \varnothing.$$

Kilka z powyższych punktów wymagać może małego komentarza. Przyjmujemy, że zbiór miejsc D-sieci, P, jest sumą zbiorów A_w i A_d. Co prawda zbiory te są zbiorami atrybutów,

ale atrybuty te są wzajemnie jednoznacznie powiązane z miejscami sieci, stąd zastosowany został taki uproszczony zapis. Podobnie, ponieważ każdemu łukowi wzajemnie jednoznacznie odpowiada pewna para wierzchołków, stąd też nieco uproszczony zapis w punkcie czwartym. W punkcie szóstym nie zostało opisane zastrzeżenie dla tranzycji odnawiającej (*TOdn*), ale zostanie to uczynione w dalszej części tej pracy. Dla pozostałych tranzycji zastrzeżenie tworzymy jako iloczyn logiczny (koniunkcję) warunków logicznych, które znajdują się w komórkach tablicy decyzyjnej, związanych z tą tranzycją. W przypadku, gdy w żadnej z komórek nie ma warunku logicznego, przyjmujemy, że wartość zastrzeżenia jest równa True.

Na rysunku 1 przedstawiony jest przykład niekompletnej D-sieci dla rozważanego przykładu sterownika windy. Łatwo zauważyć, że przy n atrybutach i m regułach w tak konstruowanej D-sieci wystąpi, aż $n*(m + 1)$ łuków, co powoduje, że rysunek staje się mało czytelny, a także bardzo czasochłonny w konstrukcji. W sieci uwzględniono więc tylko tranzycje odpowiadające regułom r_1 i r_7, a także pominięte zostało zastrzeżenie przy tranzycji odnawiającej.



Rys. 1. Fragment D-sieci zbudowanej dla rozważanego sterownika windy

Przyczyną takiego postępowania jest fakt, że prezentowana postać sieci nie jest jeszcze postacią docelową. W dalszej części algorytm konstrukcji D-sieci zostanie nieco zmodyfikowany. Ten etap przejściowy jest jednak konieczny do wyjaśnienia pewnych własności D-sieci, które pozwalają na zmniejszenie liczby jej elementów. Przedstawienie na rysunku sieci tylko dwóch tranzycji jest wystarczające dla zaprezentowania wyniku zastosowania podanego wcześniej algorytmu oraz pokazania niektórych własności sieci.

Wracając do zastrzeżenia przy tranzycji odnawiającej omówimy teraz dokładny sposób jej utworzenia. Łatwo zauważyć, że niektóre układy wartości atrybutów warunkowych są sprzeczne. Tabela 2 przedstawia wszystkie niepożądane sytuacje, które nie powinny pojawić się jako znakowanie sieci. Wartości atrybutów, które w danym momencie są nieistotne, zostały pominięte.

Jako zastrzeżenie dla tranzycji odnawiającej należy przyjąć koniunkcję zaprzeczeń powyższych sytuacji, tzn.:

$\neg$(Stan = up $\wedge$ Kierunek = ddown) $\wedge \neg$ (Stan = down $\wedge$ Kierunek = dup) $\wedge \ldots$.

Tabela 2

Stan	Poziom	Sensor	Kierunek	Zadanie
up			ddown	
down			dup	
stop	5		up	
stop	0		down	
		mniejszy od 6		no
up	0			
down	5			

Korzystając z faktu, że zaprzeczenie koniunkcji jest alternatywą zaprzeczeń oraz wykorzystując notację zaczerpniętą z sieci kolorowanych, otrzymujemy zastrzeżenie dla tranzycji odnawiającej o następującej postaci (w poniższym zapisie zamiast zmiennych występują nazwy atrybutów, co jest niedopuszczalne przy konstrukcji sieci, jednak w tym przypadku podnosi to czytelność zapisu):

[((Stan up) orelse (Kierunek = dup)),
((Stan down) orelse (Kierunek = ddown)),
((Stan stop) orelse (Poziom 5) orelse (Kierunek = ddown)),
((Stan stop) orelse (Poziom 0) orelse (Kierunek = dup)),
((Sensor = 6) orelse (Zadanie = yes)),
((Stan up) orelse (Poziom 0)),
((Stan down) orelse (Poziom 5))].

2.4. Optymalizacja D-grafu

W oparciu o rysunek 1 możemy zauważyć, że D-sieć posiada następujące własności (zob. [5]).

Wniosek 1

Wszystkie łuki D-sieci są uniformami o wielokrotności 1, tzn. $\bigwedge_{a \in A} |E(a)| = 1$.

Wniosek 2

Sieć jest uniformem.

Wniosek 3

Sieć jest bezpieczna, tzn. $\bigwedge_{M \in [M_0>} \bigwedge_{p \in P} |M(p)| \leq 1$.

Wniosek 4

$\bigwedge_{M \in [M_0>} \bigwedge_{p,p' \in A_w} |M(p)| = |M(p')|$.

Wniosek 5

$\bigwedge_{M \in [M_0>} \bigwedge_{p,p' \in A_d} |M(p)| = |M(p')|$.

Dowód

Wniosek 1 oznacza, że przy odpalaniu dowolnej tranzycji, ilość znaczników przenoszona przez każdy łuk sieci wynosi jeden. Własność ta wynika w oczywisty sposób z postaci D-sieci.

Wniosek 2 jest bezpośrednią konsekwencją wniosku 1 i oznacza, że wszystkie łuki otaczające dowolną tranzycję są uniformami.

Wniosek 3 oznacza, że przy dowolnym znakowaniu sieci liczba znaczników w każdym z miejsc jest nie większa niż 1. Łatwo zauważyć, że warunek ten jest spełniony dla znakowania początkowego określonego zgodnie z przedstawionym wcześniej algorytmem. Odpalenie dowolnej tranzycji ze zbioru R powoduje usunięcie wszystkich znaczników z miejsc warunkowych i umieszczenie po jednym znaczniku w każdym z miejsc decyzyjnych. Rozważany warunek jest w tym przypadku nadal spełniony. Odpalenie tranzycji odnawiającej powoduje zmiany analogiczne, z tym, że opróżniane są miejsca decyzyjne, zaś wysyłane są znaczniki do miejsc warunkowych. Otrzymujemy w ten sposób sytuacje podobne do wyjściowej. Zatem widać, że przy każdym możliwym znakowaniu D-sieci warunek z wniosku 3 jest spełniony.

Wniosek 4 oznacza, że liczba znaczników w miejscach warunkowych jest taka sama przy dowolnym znakowaniu sieci. W oparciu o wcześniejsze wnioski widać, że miejsca te pozostają puste bądź zawierają po jednym znaczniku.

Wniosek 5 jest analogiczny z wnioskiem 4, lecz dotyczy miejsc decyzyjnych.

W oparciu o powyższe własności możemy zmodyfikować nieco postać sieci. Zbiór miejsc warunkowych zastąpimy jednym miejscem warunkowym, którego kolorem będzie iloczyn kartezjański kolorów dotychczasowych miejsc warunkowych. Analogicznie przekształcimy zbiór miejsc decyzyjnych.

Definicja 2

Kolorowaną siecią decyzyjną, opisującą reguły zewnętrznego zachowania się systemu, będziemy nazywać niehierarchiczną kolorowaną sieć Petriego, zbudowaną na podstawie tablicy decyzyjnej według następującego algorytmu.

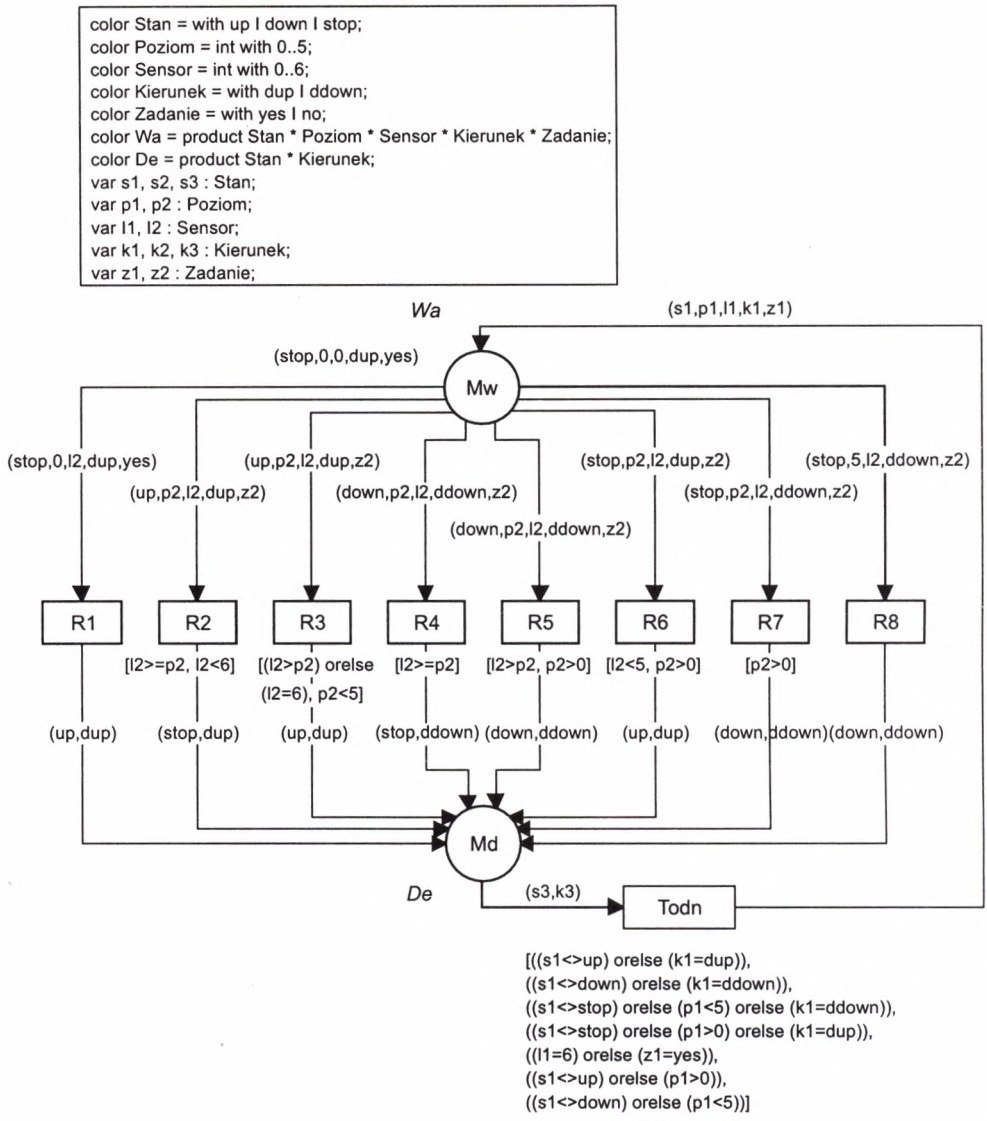
1) W sieci umieszczamy dwa miejsca, jedno odpowiadające atrybutom warunkowym, tzw. **miejsce warunkowe** (*Mw*), drugie zaś odpowiadające atrybutom decyzyjnym, tzw. **miejsce decyzyjne** (*Md*).

2) Jako kolor miejsca warunkowego przyjmujemy iloczyn kartezjański zbiorów wartości atrybutów warunkowych.
3) Jako kolor miejsca decyzyjnego przyjmujemy iloczyn kartezjański zbiorów wartości atrybutów decyzyjnych.
4) Znakowanie początkowe D-sieci określamy następująco: w miejscu warunkowym umieszczamy jeden znacznik odpowiedniego koloru, zaś miejsce decyzyjne pozostawiamy puste. Należy zwrócić uwagę, by wartości współrzędnych znacznika w miejscu warunkowym były dopuszczalne (np. nie może być Stan = up i Kierunek = ddown).
5) W sieci umieszczamy dokładnie tyle tranzycji, ile reguł decyzyjnych wyróżniono w tabeli. Tranzycje takie nazywamy **tranzycjami decyzyjnymi**. Nadajemy im nazwy np.: $r_1, r_2, \ldots, r_n$.
6) Dla każdej z tranzycji r_i prowadzimy łuk wejściowy od miejsca warunkowego do tej tranzycji oraz łuk wyjściowy do miejsca decyzyjnego.
7) Jako wagi łuków przyjmujemy struktury odpowiedniego koloru. Wagę łuku prowadzącego od miejsca warunkowego do tranzycji decyzyjnej ustalamy w sposób następujący: jeżeli w odpowiedniej komórce tabeli znajduje się stała wartość np. Poziom = 0, to wartość ta będzie odpowiednią współrzędną w wyrażeniu będącym wagą danego łuku; jeżeli w odpowiedniej komórce tabeli znajduje się zmienna np. Poziom = p, to zmienna ta będzie odpowiednią współrzędną w wyrażeniu będącym wagą danego łuku; jeżeli w odpowiedniej komórce tabeli znajduje się warunek logiczny np. $p \neq 5$, to odpowiednią współrzędną w wyrażeniu będącym wagą łuku będzie zmienna, zaś warunek umieszczamy w zastrzeżeniu tranzycji.
8) Wagę łuku prowadzącego od tranzycji decyzyjnej do miejsca decyzyjnego ustalamy w sposób analogiczny, przy czym w tej sytuacji mogą zajść tylko dwa pierwsze przypadki.
9) W sieci umieszczamy dodatkową tranzycję zwaną **tranzycją odnawiającą**.
10) Prowadzimy łuki od miejsca decyzyjnego do tranzycji odnawiającej i od tej tranzycji do miejsca warunkowego.
11) Jako wagi łuku wejściowego i wyjściowego tranzycji odnawiającej przyjmujemy struktury złożone ze zmiennych odpowiednich typów.
12) Przy tranzycji odnawiającej umieszczamy zastrzeżenie tak, by w miejscu warunkowym nie wystąpił niedopuszczalny rozkład wartości współrzędnych w znaczniku.

Rysunek 2 przedstawia D-sieć zbudowaną w oparciu o zmodyfikowany algorytm jej tworzenia (definicja 2). Od tej pory, mówiąc o D-sieci, będziemy mieli na myśli sieć zbudowaną według drugiego algorytmu. Algorytm ten pozwala znacznie szybciej tworzyć D-sieci, gdyż zawierają one znacznie mniej elementów (liczba łuków sieci jest funkcją liniową zależną od liczby tranzycji, nie zaś kwadratową).

W oparciu o tak tworzoną D-sieć będziemy badali własności utworzonej wcześniej specyfikacji wymagań zewnętrznego zachowania systemu i poprawiali ewentualne błędy. Proces ten będzie przebiegał w trzech etapach:

1) sprawdzanie, czy system potrafi zareagować w każdej możliwej sytuacji (własność zupełności zbioru reguł);
2) usuwanie sprzeczności w podejmowaniu decyzji (własność zgodności zbioru reguł);
3) usuwanie zależnych reguł (własność optymalności zbioru reguł).

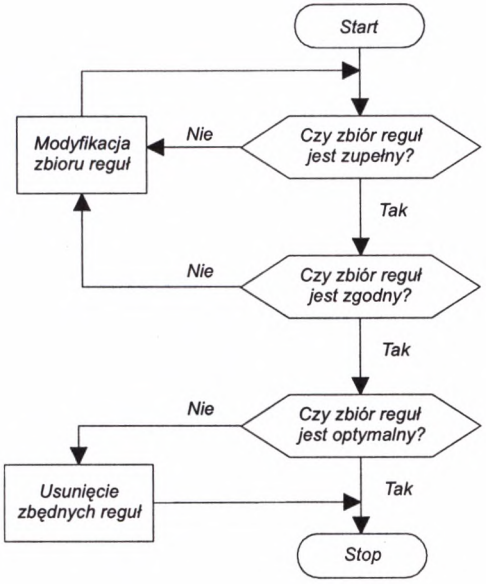


Rys. 2. Kompletna zmodyfikowana D-sieć dla rozważanego przykładu sterownika windy

Poszczególne etapy mogą występować więcej niż jeden raz. Na przykład w przypadku wystąpienia sprzeczności w drugim etapie, konieczne jest powtórzenie pierwszego etapu, po usunięciu reguł, które te sprzeczności powodowały, bądź po ich modyfikacji. Algorytm przedstawiający kolejność sprawdzania poszczególnych własności znajduje się na rysunku 3.

Ponieważ zbiór znakowań osiągalnych D-sieci może być bardzo duży, wszystkie wymienione własności będą badane bez odnoszenia się do pełnego grafu osiągalności sieci,

(mimo że można w prosty sposób udowodnić, że jest on skończony, jeżeli zbiory wartości wszystkich atrybutów są skończone). Wykorzystany zostanie natomiast graf silnie spójnych składowych oraz niezmienniki tranzycji D-sieci.



Rys. 3. Schemat blokowy procedury testowania własności specyfikacji wymagań

3. Wykorzystanie własności D-sieci przy tworzeniu pełnej specyfikacji wymagań zachowania się systemu

3.1. Badanie zupełności zbioru reguł

Analizując postać D-sieci łatwo można zauważyć, że zbiór wszystkich osiągalnych znakowań $[M_0>$ sieci można przedstawić jako sumę dwóch rozłącznych podzbiorów, tzn.

$[M_0>= W_M \cup D_M$ i $W_M \cap D_M = \varnothing$.

Pierwszy ze zbiorów W_M zawiera te wszystkie znakowania sieci, przy których w miejscu warunkowym *Mw* znajduje się znacznik, zaś miejsce decyzyjne *Md* pozostaje puste. Znakowania takie będziemy nazywać **znakowaniami warunkowymi**. Odzwierciedlają one te wszystkie sytuacje, w jakich może znaleźć się projektowany system.

Definicja 3

Zbiór reguł *R*, opisujący zewnętrzne zachowanie się systemu, nazywamy **zupełnym** wtedy i tylko wtedy, gdy system potrafi podjąć decyzję w każdej dopuszczalnej sytuacji.

Należy zwrócić uwagę, że w przyjętej definicji nie wymaga się, by podejmowana decyzja była jednoznaczna, tzn. w pierwszym etapie nie będziemy rozpatrywali problemu sprzeczności decyzji.

Przenosząc powyższą definicję na grunt D-sieci, otrzymujemy następujące twierdzenie.

Twierdzenie 1

Zbiór reguł R opisujący zewnętrzne zachowanie się systemu jest zupełny wtedy i tylko wtedy, gdy zbiór znakowań warunkowych W_M nie zawiera znakowania martwego.

Dowód powyższego twierdzenia jest bardzo prosty. Jeżeli zbiór R jest zupełny, to oznacza to, że system potrafi podjąć decyzję w każdej sytuacji. Zatem przy dowolnym znakowaniu warunkowym aktywna jest co najmniej jedna tranzycja decyzyjna. Oznacza to, że żadne ze znakowań warunkowych nie może być znakowaniem martwym.

Z drugiej strony, gdyby istniało znakowanie martwe D-sieci należące do zbioru W_M, to oznaczałoby to, że istnieje sytuacja, w której system nie potrafi podjąć decyzji. Przeczyłoby to założeniu, że R jest zupełny.

Wróćmy do drugiego z wyróżnionych zbiorów znakowań. Zbiór D_M zawiera te wszystkie znakowania sieci, przy których znacznik znajduje się w miejscu decyzyjnym, zaś miejsce warunkowe pozostaje puste. Znakowania takie będziemy nazywać **znakowaniami decyzyjnymi**. Miejsce decyzyjne posiada tylko jedną tranzycję wyjściową, *TOdn*. Tranzycja ta jest aktywna, jeżeli w miejscu decyzyjnym znajduje się co najmniej jeden znacznik, czyli przy dowolnym znakowaniu decyzyjnym.

Wniosek 6

Zbiór znakowań decyzyjnych D_M D-sieci nie zawiera znakowania martwego.

Biorąc pod uwagę powyższy wniosek, wcześniejsze twierdzenie możemy zapisać w następującej postaci.

Twierdzenie 2

Zbiór reguł R opisujący zewnętrzne zachowanie się systemu jest zupełny wtedy i tylko wtedy, gdy zbiór wszystkich znakowań osiągalnych D-sieci nie zawiera znakowania martwego.

Przejdziemy teraz do omówienia sposobu szybkiego wyszukiwania martwych znakowań decyzyjnych. Łatwo zauważyć, że w przypadku, gdy dla wszystkich atrybutów zbiory ich dopuszczalnych wartości są skończone, zbiór wszystkich osiągalnych znakowań D-sieci jest skończony, tzn.

$$\bigwedge_{u \in A_w \cup A_d} |V_u| < \infty \quad \Rightarrow \quad |[M_0>| < \infty.$$

Jeżeli przyjmiemy, że liczby naturalne n_u są mocami zbiorów V_u, to liczba elementów w zbiorze $[M_0>$ jest nie większa od iloczynu tych liczb. Jednak mimo wszystko analizowanie pełnego grafu osiągalności D-sieci byłoby bardzo czasochłonne, nawet z wykorzystaniem dostępnych narzędzi komputerowych.

Niech M_1 i $M_2 \in W_M$ będą dwoma dowolnymi niemartwymi znakowaniami warunkowymi D-sieci. Jeżeli M_1 nie jest znakowaniem martwym, to istnieje tranzycja $r \in R$ i znakowanie $M \in D_W$ takie, że $M_1[r > M$. Przy znakowaniu M aktywna jest tranzycja odnawiająca *TOdn*. Ze względu na fakt, że wagą łuku wyjściowego tranzycji odnawiającej jest układ (wektor) zmiennych, *TOdn* można odpalić przy takim wiązaniu, by uzyskać, jako kolejne znakowanie, M_2. Zatem $M_2 \in [M_1>$. Podobnie można pokazać, że $M_1 \in [M_2>$.

Ponieważ znakowania M_1 i M_2 były dowolnymi niemartwymi znakowaniami warunkowymi D-sieci, otrzymujemy następujący wniosek.

Wniosek 7

Wszystkie niemartwe znakowania warunkowe D-sieci należą do jednej silnie spójnej składowej pełnego grafu osiągalności tej D-sieci.

Łatwo pokazać, że do tej składowej należą również wszystkie znakowania decyzyjne. Jako wynik powyższych rozważań możemy sformułować następujące twierdzenie.

Twierdzenie 3

Zbiór reguł R opisujący zewnętrzne zachowanie się systemu jest zupełny wtedy i tylko wtedy, gdy graf silnie spójnych składowych jest trywialny, tzn. składa się z tylko jednego wierzchołka.

Dowód powyższego twierdzenia jest natychmiastowy. Jeżeli R jest zupełny, to zbiór W_M nie zawiera znakowania martwego, więc wszystkie znakowania D-sieci należą do jednej silnie spójnej składowej grafu osiągalności tej sieci, (graf taki będziemy nazywali SSS-grafem D-sieci). Z drugiej strony, jeżeli SSS-graf jest trywialny, to w zbiorze $[M_0>$ nie ma znakowania martwego, więc R jest zupełny.

Kolejne twierdzenie daje odpowiedź na pytanie, w jaki sposób można szukać martwych znakowań warunkowych.

Twierdzenie 4

Jeżeli SSS-graf sieci nie jest trywialny, to zbiór wierzchołków terminalnych tego grafu, (tzn. takich, które nie mają łuków wyjściowych), składa się z tych wszystkich znakowań warunkowych, przy których system nie potrafi podjąć decyzji.

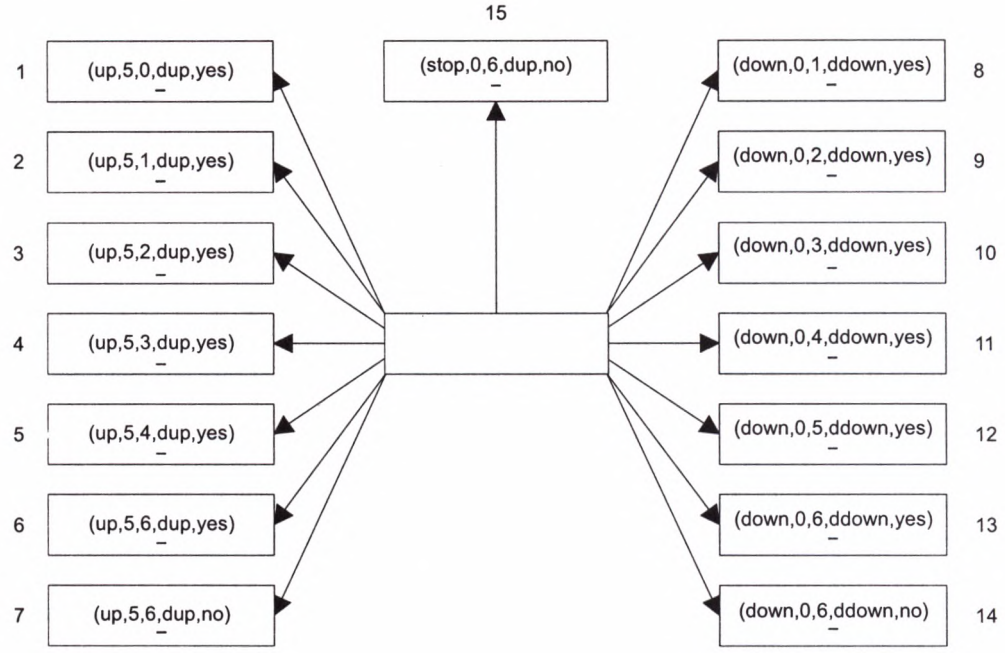
Wracając do rozważanego wcześniej przykładu ze sterownikiem windy, dla D-sieci z rysunku 2 otrzymujemy SSS-graf przedstawiony na rysunku 4. SSS-graf tej sieci zawiera 15 wierzchołków terminalnych.

Wierzchołki terminalowe SSS-grafu z rysunku 4 można podzielić na trzy grupy:

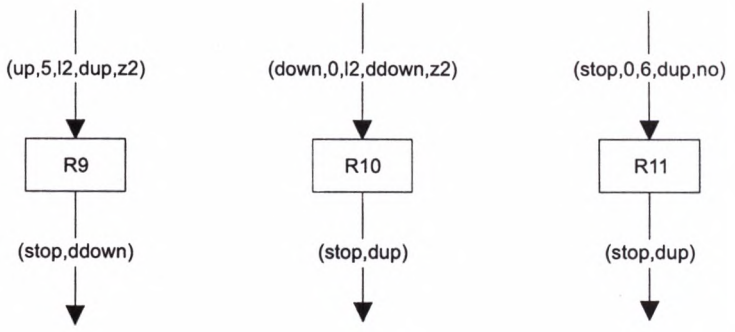
1) Wierzchołki 1–7 odnoszą się do sytuacji, gdy winda zbliża się do 5 piętra. Bez względu na stan sensora na tym piętrze winda powinna się zatrzymać, a następnie rozpocząć obsługę zgłoszeń związanych z ruchem w dół.
2) Wierzchołki 8–14 odnoszą się do sytuacji, gdy winda zbliża się do parteru. Bez względu na stan sensora na parterze winda powinna się zatrzymać, a następnie rozpocząć obsługę zgłoszeń związanych z ruchem w górę, jeśli takie zgłoszenia są.
3) Wierzchołek 15 odnosi się do sytuacji, gdy winda znajduje się w pozycji wyjściowej i nie ma żadnych zgłoszeń. W tej sytuacji postój na parterze powinien być kontynuowany aż do nadejścia dowolnego zgłoszenia.

Na rysunku 5 są przedstawione dodatkowe tranzycje decyzyjne, w jakie powinna zostać wzbogacona rozważana D-sieć. Reguły te odpowiadają omówionym powyżej trzem przypadkom, w których system nie był w stanie podjąć żadnej decyzji.

Analizując tak wzbogaconą D-sieć, otrzymujemy trywialny SSS-graf. Zatem zbiór reguł $R = \{r_1, r_2, ..., r_{11}\}$ jest zupełny.



Rys. 4. SSS-graf zbudowany dla D-sieci z rysunku 2



Rys. 5. Dodatkowe tranzycje do umieszczenia w D-sieci z rysunku 2 w celu uzyskania zupełnego zbioru reguł

3.2. Badanie niesprzeczności zbioru reguł

W sytuacji, gdy zbiór reguł *R* jest zupełny, każdemu znakowaniu warunkowemu odpowiada co najmniej jedno znakowanie decyzyjne. Może się zdarzyć, że przy danym znakowaniu warunkowym aktywnych jest więcej niż jedna tranzycja decyzyjna.

Definicja 4

Niech $r, r' \in R$ będą dwiema różnymi tranzycjami decyzyjnymi, które są aktywne przy znakowaniu warunkowym $M \in W_M$.

Jeżeli $M[r > M_1 \wedge M[r' > M_2 \wedge M_1 \neq M_2$, to mówimy, że tranzycje r i r' są **sprzeczne** przy warunku M.

Jeżeli $M[r > M_1 \wedge M[r' > M_2 \wedge M_1 = M_2$, to mówimy, że tranzycje r i r' są **zgodne** przy warunku M.

Zbiór reguł R opisujących zewnętrzne zachowanie się systemu nazywamy **zgodnym**, jeżeli

$\bigwedge_{r,r' \in R} \bigwedge_{M \in W_M}$ tranzycje r i r' nie są sprzeczne.

Zgodność zbioru reguł oznacza, że w każdej sytuacji system potrafi podjąć decyzję w sposób jednoznaczny. W dalszej części zostanie pokazane, w jaki sposób można sprawdzić, czy dany zbiór reguł jest zgodny.

Przyjmijmy, że zgodnie z definicją D-sieci jej znakowaniem początkowym jest znakowanie warunkowe. Łatwo zauważyć, że każda para tranzycji (r, *TOdn*) złożona z tranzycji decyzyjnej i tranzycji odnawiającej jest niezmiennikiem tranzycji (T-niezmiennikiem) D-sieci. We wcześniejszym paragrafie omówione zostało, w jaki sposób można przejść od dowolnego znakowania M_1 do znakowania M_2 odpalając jedynie dwie tranzycje, pod warunkiem, że M_1 i M_2 nie są martwe. Jeżeli zbiór R jest zupełny, to nie ma w nim już znakowań martwych. Jeżeli przyjmiemy, że $M_1 = M_2$, to odpalając rozważaną parę tranzycji, wracamy ponownie do tego samego znakowania.

Oczywiście przy danym znakowaniu początkowym nie wszystkie T-niezmienniki są realizowalne. Z drugiej strony dla każdego z takich T-niezmienników istnieje znakowanie początkowe, przy którym jest on realizowalny, (bo R jest zupełny).

Z realizowalnością każdego z T-niezmienników związany jest pewien ciąg kolejnych znakowań (pewna droga w pełnym grafie osiągalności), np. dla T-niezmiennika (r, *TOdn*) mamy: $M_0[r > M[TOdn > M_0$.

Drogę taką nazywać będziemy **cyklem decyzyjnym**. Zbiór wszystkich cykli decyzyjnych oznaczymy symbolem DC. W zbiorze tym wprowadzimy relację ~ określoną w sposób następujący:

$$\bigwedge_{d_m, d_n \in DC} (d_m = M_m[r_{m1} > M'_m\,[TOdn > M_m \wedge d_n = M_n[r_{n1} > M'_n[TOdn > M_n) \Rightarrow$$

$$(d_m \sim d_n \Leftrightarrow M_m = M_n$$

Łatwo udowodnić, że tak wprowadzona relacja jest relacją równoważności. W zbiorze $DC/\sim$ interesować nas będą te klasy abstrakcji, które mają więcej niż jeden element.

Rozważmy klasę abstrakcji cyklu decyzyjnego $d \in DC$, takiego, że $|[d]| > 1$. Musi zatem istnieć cykl decyzyjny $d^* \in DC$ taki, że $d^* \in [d]$ i $d^* \neq d$. Niech $d = M[r > M'[TOdn > M$ oraz $d^* = M[r^* > M^{*\prime}[TOdn > M$.

Jeżeli cykle d i d^* są różne, to może zajść jedna z dwóch sytuacji:

1) $r \neq r^* \wedge M' \neq M^{*\prime}$,
2) $r \neq r^* \wedge M' = M^{*\prime}$.

W pierwszym przypadku mamy sprzeczność tranzycji decyzyjnych, zaś w drugim zgodność tych tranzycji.

Jeżeli rozpatrujemy dowolną z klas abstrakcji, to, zgodnie z definicją, wszystkie należące do niej cykle decyzyjne mają wspólne znakowanie warunkowe, natomiast mogą różnić się znakowaniem decyzyjnym. Jeżeli $x \in DC/\sim$, to o klasie abstrakcji x będziemy mówić, że posiada znakowanie decyzyjne M, jeżeli M należy do co najmniej jednego cyklu zawartego w tej klasie. W oparciu o powyższe rozważania można sformułować poniższe twierdzenie.

Twierdzenie 5

Zbiór reguł R jest zgodny wtedy i tylko wtedy, gdy każda klasa abstrakcji relacji $\sim$ posiada dokładnie jedno znakowanie decyzyjne.

Jeżeli wyznaczymy wszystkie klasy dla D-sieci z rysunku 2 to okaże się, że dwie z nich posiadają po dwa znakowania decyzyjne. Klasy te zostały przedstawione w tabeli 3.

Tabela 3

x_1	(up,5,5,dup,yes)	r_2	(stop,dup)	*TOdn*	(up,5,5,dup,yes)
	(up,5,5,dup,yes)	r_9	(stop,ddown)	*TOdn*	(up,5,5,dup,yes)
x_2	(down,0,0,ddown,yes)	r_4	(stop,ddown)	*TOdn*	(down,0,0,ddown,yes)
	(down,0,0,ddown,yes)	r_{10}	(stop,dup)	*TOdn*	(down,0,0,ddown,yes)

Łatwo można zauważyć, że w przypadku klasy abstrakcji x_1 decyzja podejmowana przez tranzycję r_2 jest niepoprawna. By wyeliminować sprzeczność, można zmodyfikować zastrzeżenie tej tranzycji, dodając do koniunkcji jeszcze jeden człon postaci $p \neq 5$. Podobnie w przypadku klasy x_2, do zastrzeżenia tranzycji r_4 dołączamy warunek $p \neq 0$.

Po wprowadzeniu powyższych modyfikacji zbiór reguł R jest zgodny.

3.2. Badanie optymalności zbioru reguł

W sytuacji gdy zbiór reguł R jest zupełny i zgodny, każdemu znakowaniu warunkowemu odpowiada dokładnie jedno znakowanie decyzyjne. Można jednak w tym miejscu postawić pytanie, czy skonstruowanego do tej pory zbioru reguł nie dałoby się uszczuplić, zmniejszając ilość tranzycji decyzyjnych.

Definicja 5

Regułę $r \in R$, gdzie R jest zupełnym i zgodnym zbiorem reguł, nazywamy **niezależną**, jeżeli zbiór reguł $R - \{r\}$ nie jest zbiorem zupełnym. Regułę $r \in R$, gdzie R jest zupełnym i zgodnym zbiorem reguł, nazywamy zależną, jeżeli nie jest ona niezależna.

Zbiór reguł R nazywamy **optymalnym**, jeżeli wszystkie reguły z tego zbioru są regułami niezależnymi.

Jeżeli $x \in DC/\sim$, to o klasie abstrakcji x będziemy mówić, że posiada regułę decyzyjną r, jeżeli r należy do co najmniej jednego cyklu zawartego w tej klasie.

Twierdzenie 6

Reguła $r \in R$, gdzie R jest zupełnym i zgodnym zbiorem reguł, jest regułą niezależną wtedy i tylko wtedy, gdy istnieje jednoelementowa klasa abstrakcji relacji $\sim$, posiadająca regułę r.

Dowód

Jeżeli $r \in R$ jest regułą niezależną, to zgodnie z definicją 5 zbiór reguł $R - \{r\}$ nie zbiorem zupełnym. Oznacza to, że istnieje znakowanie warunkowe M, przy którym system nie potrafi podjąć decyzji. Biorąc pod uwagę również założenie, że R jest zbiorem zupełnym otrzymujemy wniosek, że jedyną tranzycją decyzyjną, która jest aktywna przy znakowaniu M, jest tranzycja r. Zatem klasa abstrakcji posiadająca tranzycję r zawiera tylko cykl decyzyjny, do którego ta tranzycja należy.

Z drugiej strony, jeżeli istnieje klasa abstrakcji posiadająca tranzycję r, która jest jednoelementowa, to musi istnieć znakowanie warunkowe M', przy którym aktywna jest tylko tranzycja r. Zatem usunięcie ze zbioru R tranzycji decyzyjnej r spowoduje, że przy tak zmniejszonym zbiorze reguł system nie będzie potrafił zareagować przy znakowaniu warunkowym M'.

Opierając się na twierdzeniu 6, pokażemy, które ze zbioru reguł $R = \{r_1, r_2, ..., r_{11}\}$ są niezależne. Tabela 4 zawiera przykłady jednoelementowych klas abstrakcji relacji $\sim$. Każda z przedstawionych klas abstrakcji posiada inną tranzycję decyzyjną.

Tabela 4

x_1	(stop,0,0,dup,yes)	r_1	(up,dup)	*TOdn*	(stop,0,0,dup,yes)
x_2	(up,3,3,dup,yes)	r_2	(stop,dup)	*TOdn*	(up,3,3,dup,yes)
x_3	(up,3,1,dup,yes)	r_3	(up,dup)	*TOdn*	(up,3,1,dup,yes)
x_4	(down,2,0,ddown,yes)	r_4	(stop,ddown)	*TOdn*	(down,2,0,ddown,yes)
x_5	(down,1,2,ddown,yes)	r_5	(down,ddown)	*TOdn*	(down,1,2,ddown,yes)
x_6	(stop,1,6,dup,yes)	r_6	(up,dup)	*TOdn*	(stop,1,6,dup,yes)
x_7	(stop,3,1,ddown,yes)	r_7	(down,ddown)	*TOdn*	(stop,3,1,ddown,yes)
x_9	(up,5,2,dup,yes)	r_9	(stop,ddown)	*TOdn*	(up,5,2,dup,yes)
x_{10}	(down,0,1,ddown,yes)	r_{10}	(stop,dup)	*TOdn*	(down,0,1,ddown,yes)
x_{11}	(stop,0,6,dup,no)	r_{11}	(stop,dup)	*TOdn*	(stop,0,6,dup,no)

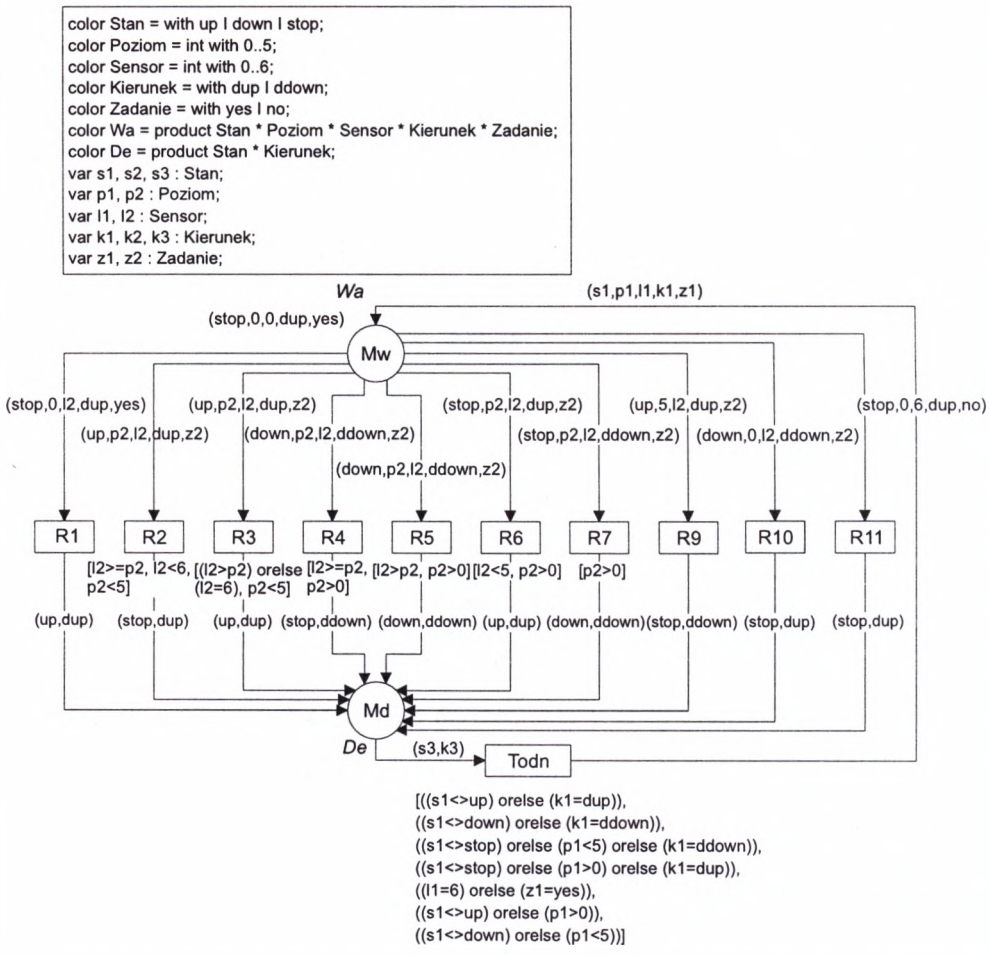
Jedyną tranzycją decyzyjną, która jest zależna, jest tranzycja r_8. Wszystkie klasy abstrakcji relacji $\sim$, które posiadają tranzycję r_8, posiadają również tranzycję r_7.

Twierdzenie 7

Jeżeli zbiór reguł R jest zbiorem zupełnym i zgodnym, to zbiór reguł R', powstały ze zbioru reguł R przez usunięcie reguły zależnej, również jest zbiorem zupełnym i zgodnym.

Dowód powyższego twierdzenia jest bardzo prosty. Skoro usuwamy ze zbioru zupełnego R regułę zależną, to z definicji 5 wynika, że zbiór R' jest zupełny. Natomiast usunięcie ze zbioru zgodnego dowolnej reguły nie może spowodować pojawienia się sprzeczności (sprzeczność może się pojawić jeżeli dodajemy nowe bądź modyfikujemy istniejące reguły). Zatem R' musi być również zgodny.

Po usunięciu reguły r_8, otrzymujemy zbiór reguł R, będący zupełnym, zgodnym i optymalnym zbiorem reguł opisującym zewnętrzne zachowanie się rozważanego systemu sterownika windy. Rysunek 6 przedstawia kompletną D-sieć dla tego systemu.



Rys. 6. Końcowa postać D-sieci dla rozważanego przykładu sterownika windy

4. Podsumowanie

Przedstawiona w powyższym artykule metoda tworzenia formalnego i poprawnego opisu wymagań dotyczących zewnętrznego zachowania się systemu jest metodą opierającą się głównie na zagadnieniach związanych z sieciami Petriego wysokiego poziomu. Wymaga ona zatem od osoby, która chciałaby z niej skorzystać, co najmniej elementarnych wiadomości z tego zakresu. Ze względu jednak na fakt istnienia narzędzi komputerowych umożliwiających w pełni zautomatyzowaną analizę sieci Petriego oraz wykorzystywanie sieci na różnych etapach tworzenia modeli systemów czasu rzeczywistego, metoda ta może się okazać bardzo pomocna.

Literatura

[1] Alford M.: *SREM at the Age of Eight; The Distributed Computing Design System*. Computer, Vol. 18, 1985, No 4, 36–46

[2] Braek R., Haugen O.: *Engineering Real-Time Systems*. Prentice Hall 1993

[3] Coad P., Yourdon E.: *Analiza obiektowa*. Warszawa, Oficyna Wydawnicza Read Me 1994

[4] Davis A.M.: *A Comparison of Techniques For the Specification of External Systems Behaviour*. Communication of the ACM Vol. 31, No 9, 1988, 1098–1115

[5] Jensen K.: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Vol.1, 2 & 3. New York, Springer Verlag 1996

[6] Macaulay L.A.: *Requirements Engineering*. Springer Verlag 1996

[7] Sacha K.: *Projektowanie oprogramowania systemów wbudowanych*. Prace Naukowe Politechniki Warszawskiej z. 115, Warszawa, Oficyna Wydawnicza Politechniki Warszawskiej 1996

[8] Szmuc T.: *Problemy tworzenia oprogramowania systemów czasu rzeczywistego*. Raport z Pierwszej Krajowej Konferencji: Systemy i metody komputerowe w badaniach naukowych i projektowaniu inżynierskim, rozdział 5, s.1–20, 1997

[9] Ward P.T., Mellor S.J.: *Structured Development for Real-Time Systems*. Prentice Hall, Yourdon Press 1985

[10] Yourdon E.: *Modern Structured Analysis*. Tłum. na jęz. polski: *Współczesna Analiza Strukturalna*. Prentice Hall 1988, Warszawa, WNT 1996

Recenzent
prof. dr hab. inż. Tomasz Szmuc