Metoda Huffmana – komentarze do zadań

Zadanie 1. Zaprojektować układ o wejściach a, b i wyjściu w działający wg podanego wykresu czasowego.

W pierwszym etapie tworzymy pierwotną siatkę programu. Nie musi ona (ale może) zawierać wszystkich kolumn, a tylko te, które odpowiadają kombinacjom argumentów wejściowych rzeczywiście występującym na wykresie. Należy uważać na prawidłowe rozróżnienie stanów stabilnych i niestabilnych, jak również na to, by z ostatniego stanu był powrót na początek siatki. Warto również zauważyć, że pierwotna siatka programu zawsze odpowiada automatowi Moore'a.

Mając gotową pierwotną siatkę programu, możemy przystąpić do I etapu redukcji. Jest to redukcja stanów równoważnych i pseudorównoważnych. Analizę możliwości redukcji przeprowadzamy w kolumnach siatki. Wypisujemy numery stanów z każdej kolumny, a następnie sprawdzamy metodą porównania „każdy z każdym", czy redukcja dwóch porównywanych stanów jest możliwa. Sytuacja taka występuje, gdy stany wyjść są identyczne, a przejścia są niesprzeczne.

W pierwszej kolumnie mamy stany 1, 3 i 5. 1 i 3 można zredukować, natomiast żaden z nich nie może być zredukowany ze stanem 5, ponieważ ma on inny stan wyjścia. W kolumnie 2 występuje tylko stan 4 – nie ma czego redukować. Kolumna 3 jest pusta. W kolumnie 4 mamy stany 2 i 6, które można zredukować, ale tylko pod warunkiem, że wcześniej zredukujemy stany 1 i 3. Warunek ten zapisujemy przy kresce łączącej dwa warunkowo redukowane stany.

Uwaga! Warunek redukcji x=y NIE oznacza, że x można zredukować z y. Oznacza on, że x JUŻ został zredukowany z y i jest to ten sam stan. Gdyby bowiem stany x i y nie zostały zredukowane w jeden stan, występowałaby sprzeczność przejść i redukcja byłaby nieprawidłowa.

Po redukcji tworzymy zredukowaną siatkę programu. W naszym przykładzie uzyskaliśmy redukcję z 6 stanów do 4.

Kolejny II etap redukcji to redukcja wierszy. Przebiega ona w różny sposób zależnie od tego, czy redukujemy dla automatu Moore'a, czy Mealy'ego. Zazwyczaj w procesie analizujemy możliwość redukcji dla obu automatów. Dwa wiersze można zredukować, jeśli przejścia nie są sprzeczne. Dodatkowo dla automatu Moore'a musi wystąpić zgodność stanów wyjściowych (warunek ten nie jest konieczny dla automatu Mealy'ego, co zazwyczaj, ale nie zawsze, pozwala osiągnąć mniejszą liczbę elementów pamięci w automacie Mealy'ego).

Oznaczamy poszczególne wiersze np. greckimi literami (lub dowolnymi innymi oznaczeniami, byle uniknąć niejednoznaczności w ramach zadania – np. a i b są argumentami wejściowymi i nie powinniśmy używać a i b do oznaczeń wierszy). Wiersze α i β można zredukować bezwarunkowo dla obu rodzajów automatów, ponieważ przejścia nie są sprzeczne, a stany wyjść są identyczne. Oznaczamy to linią ciągłą. Wiersze γ i δ można zredukować tylko dla automatu Mealy'ego, ponieważ mają różne stany wyjść. Oznaczamy to linią przerywaną.

Od tej chwili dalsze rozwiązanie jest inne dla struktury Moore'a, a inne dla Mealy'ego.

W strukturze Moore'a uzyskujemy siatkę przejść/wyjść, która ma 3 wiersze, a zatem potrzebujemy 2 elementów pamięci. Należy teraz zbadać, czy w siatce występuje wyścig, a jeśli tak, to jakiego jest on rodzaju. W rozpatrywanej siatce, w kolumnie 4 występuje wyścig, ponieważ stanu niestabilny i odpowiadający mu stan stabilny nie są sąsiednie logicznie. Jest to wyścig niekrytyczny, ponieważ w kolumnie mamy tylko jeden stan stabilny. Usunięcie wyścigu jest tu stosunkowo proste – można pokierować przejście z wiersza 11 do 00 przez wiersz 01 lub 10. Uzyskujemy zatem, zamiast

niesąsiedniego logicznie przejścia 11→00, sąsiednie logicznie przejścia 11→01→00 lub 11→10→00. Dzięki temu wyścig zostaje usunięty.

Kolejny etap to kodowanie stanów. W zakodowanej siatce widać jedno z możliwych pokierować wyścigu. Dla jasności rozpisujemy siatkę na osobne siatki dla Q_1 i Q_0. Dalszy ciąg zależy od tego, czy realizacja ma być na bramkach, czy na przerzutnikach – podobnie jak dla TKŁ.

Z siatek widać, że wyjście możemy określić jako $w=q_1$.

W strukturze Mealy'ego uzyskujemy siatkę, która ma tylko 2 wiersze, a zatem potrzebny jest tylko jeden element pamięci. W siatce o 2 wierszach wszystkie przejścia są sąsiednie logicznie, więc nie występuje tu wyścig. Stany wyjść mogą jednak zmieniać się w obrębie wiersza, dlatego zapisujemy je w każdym polu tabeli. Dla stanów stabilnych przyjmujemy taki stan wyjść, jaki jest w odpowiadającym mu wierszu pierwotnej siatki przejść. Dla stanów niestabilnych przyjmujemy zasadę, że jeśli jest to przejście między dwoma stanami stabilnymi, które mają taki sam stan wyjść, to stan powinien być określony także dla stanu niestabilnego. W ten sposób unikniemy hazardu na wyjściach. Jeśli dla rozważanych stanów układu są określone różne stany wyjść, to w stanach niestabilnych możemy pozostawić stany wyjść nieokreślone.

Zadanie 2. Zaprojektować układ o wejściach a, b i wyjściu w działający wg podanego wykresu czasowego.

Tworzymy pierwotną siatkę programu. Następnie przechodzimy do I etapu redukcji.

W kolumnie 1 mamy stany 1, 3, 5, 9, 13, 15. Stany te dzielą się na dwie grupy – stany 3 i 15 mają wyjście w=1, pozostałe w=0. Stany 3 i 15 można zredukować bezwarunkowo. Stan 1 można zredukować ze stanami 9 i 13 bezwarunkowo, a ze stanem 5 pod warunkiem, że stany 2 i 6 zostały wcześniej zredukowane. Stan 5 można także zredukować bezwarunkowo z 9 i 13. Z kolei stan 9 można zredukować z 13 pod warunkiem, że stany 10 i 14 zostały wcześniej zredukowane. Aby stany 1, 5, 9 i 13 można było zredukować w jeden stan, wszystkie redukcje muszą być parami możliwe. Wymaga to, aby stany 2 i 6 oraz 10 i 14 były wcześniej zredukowane. Jeszcze nie wiemy, czy da się to zrobić, ale za chwilę się okaże.

W kolumnie 2 mamy stany 4, 8, 10 i 14. Stan 4 można zredukować bezwarunkowo z 10, z 8 pod warunkiem, że 5 będzie zredukowany z 9 (co, jak już wiemy, jest możliwe) oraz z 14 pod warunkiem, że zredukowany zostanie stan 5 i 15 (co, jak już wiemy, nie jest możliwe).

W kolumnie 3 mamy stany 7 i 11, które można zredukować bezwarunkowo.

W kolumnie 4 mamy stany 2, 6, 12 i 16. Stan 2 można zredukować bezwarunkowo z 6, z 12 pod warunkiem, że 3 i 13 zostały zredukowane (co nie jest możliwe), i z 16 pod warunkiem, że zredukowano 1 i 3 (co także nie jest możliwe). Z kolei 6 można zredukować bezwarunkowo z 12 i 16. Wreszcie stany 12 i 16 można zredukować, o ile zredukowano stany 1 i 13 (co jest możliwe).

Jak widać, w przypadku licznych redukcji warunkowych prawidłowe poprowadzenie redukcji daje duże możliwości optymalizacyjne. Gdybyśmy przykładowo zredukowali 6 i 12, a nie 6 i 2, to nie dałoby się uzyskać redukcji stanów 1 i 5, a więc w kolumnie 1 mielibyśmy po I etapie redukcji nie 2 stany, a 3. Podobna sytuacja występuje w kolumnie 2 – proszę sprawdzić, co się stanie, jeśli zredukujemy stany 10 z 14, a co, kiedy 10 z 8.

W rezultacie I etapu redukcji uzyskujemy stany (pogrubieniem oznaczono numer stanu, który będzie reprezentował wszystkie zredukowane stany):

1. w kolumnie 1:

    a. **1**, 5, 9, 13
    b. **3**, 15
2. w kolumnie 2:
    a. **4**, 8
    b. **10**, 14
3. w kolumnie 3:
    a. **7**, 11
4. w kolumnie 4:
    a. **2**, 6
    b. **12**, 16

Uzyskujemy zredukowaną siatkę programu, która po I etapie liczy 7 stanów. Następnie redukujemy wiersze. Wiersza α nie da się zredukować z żadnym innym. Wiersz β można zredukować z ε i η, natomiast γ – tylko z ε. Wiersz δ można zredukować z ε i ω. Wszystkie redukcje możliwe są dla obu typów automatów, tzn. nie ma takiej redukcji, która przy wyborze struktury Mealy'ego przyniosłaby korzyści w stosunku do struktury Moore'a.

Redukcja wierszy daje nam dwa rozwiązania:

1. α; β, η; γ, ε; δ, ω
2. α; β, η; γ; δ, ε, ω

Rozwiązania są identyczne z punktu widzenia liczby wierszy i wynikającej z tego liczby elementów pamięci, ale nie są identyczne z punktu widzenia wyścigu.

Siatka programu po II etapie dla rozwiązania 1 jest bowiem wolna od hazardu (przejścia między stanami niestabilnymi i odpowiadającymi im stanami stabilnymi są sąsiednie logicznie), natomiast rozwiązanie 2 zawiera w kolumnie 3 wyścig niekrytyczny. Usunięcie tego wyścigu nie wymaga zwiększania liczby wierszy, a jedynie pokierowania przejścia: 01→11→10 lub 01→00→10.

W następnym etapie kodujemy siatki przejść. Dalsza procedura jest taka, jak w poprzednim przykładzie – rozbijamy siatki na osobne dla poszczególnych elementów pamięci i postępujemy zależnie od tego, czy układ należy zrealizować na bramkach, czy na przerzutnikach.