Automaty z czasem

Paulina Kania, Łukasz Osipiuk

17 lutego 2004

1 Protokoły komunikacyjne a automaty z czasem

1.1 Algorytm Petersona

Algorytm ten rozwiązuje problem wzajemnego wykluczania dla n procesów. Zajmijmy się jednak wersją, w której występują tylko dwa procesy.

proces 1	proces2
req1 = 1;	req2 = 1;
turn = 2;	turn = 1;
while (turn != 1 && req2 != 0);	while (turn != 2 && req1 != 0);
// sekcja krytyczna	// sekcja krytyczna
req1 = 0;	req2 = 0;

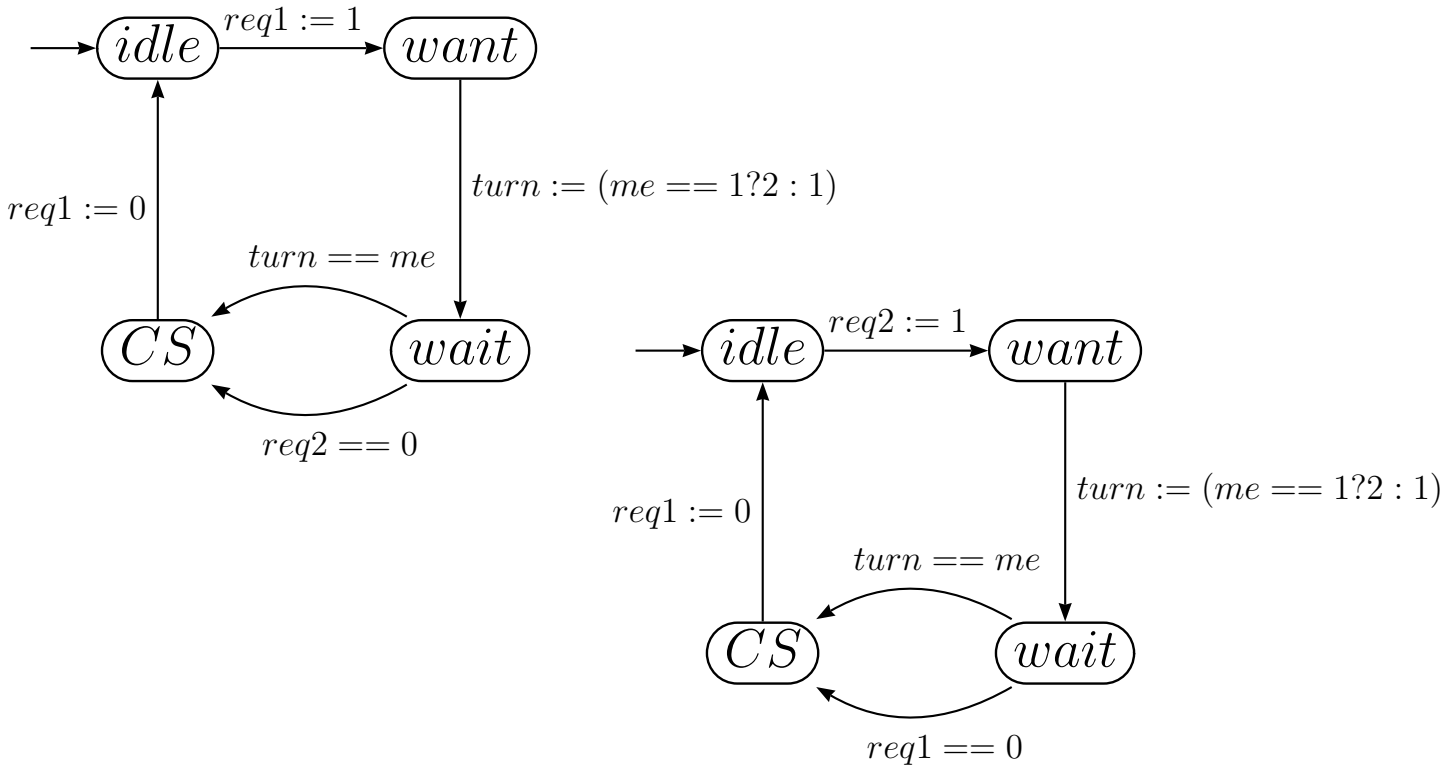
Tablica 1: Algorytm Petersona dla 2 procesów

Skonstruujemy teraz automat z czasem odpowiadający protokołowi 1. Najpierw wydzielimy poszczególne stany. Jako, że zachowanie procesów jest symetryczne, rozważymy tylko pierwszy z nich.

stan	akcja
idle	req1 = 1;
want	turn = 2;
wait	while (turn != 1 && req2 != 0);
CS	// sekcja krytyczna
brak	req1 = 0; // return to idle

Tablica 2: Wydzielenie stanów z alg. Petersona

Jak widzimy na rysunku 2, automat będzie się składał z 4 lokacji. Dodatkowo z każdym przejściem będzie związane albo ustawienie jakiejś zmiennej albo sprawdzenie warunku. Gotowe automaty widzimy na rysunku 1.



Rysunek 1: Automaty z czasem reprezentujące protkół Petersona

1.2 Algorytm Fischera

Jest to algorytm wzajemnego wykluczania dla systemów składających się z n procesów. Wykorzystuje on do tego zmienną dzieloną między wszystkimi procesami oraz odpowiednie odstępy czasowe pomiędzy poszczególnymi akcjami.

Główna idea polega na takim ustawieniu ograniczeń czasowych, żeby tylko jeden proces mógł zmienić zmienną globalną na swój numer, później ją przeczytać i jeśli nadal jest ona ustawiona na jego numer, wejść do sekcji krytycznej.

Zakładamy, że każdy z procesów ma lokalny zegar. Będziemy rozpatrywać tylko uproszczoną wersję protokołu, która pozwala jednemu z procesów wejść do sekcji krytycznej, jednak nie pozwala już jej jemu opuścić.

Krótki opis protokołu widocznego na rysunku 2:

1. Początkowo procesy są w stanie A, a zmienna dzielona v jest ustawiona na 0.

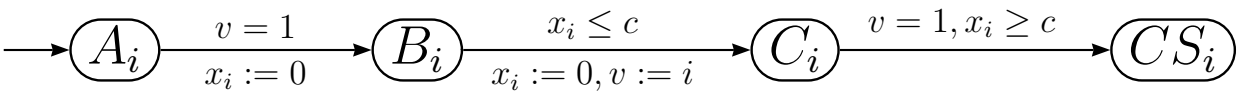
2. Proces P_i, który chce wejść do sekcji krytycznej, zmiena swój stan z A_i na B_i, o ile $v = 0$. Przy przejściu resetuje swój zegarek x_i.

3. W stanie B_i przebywa co najwyżej c jednostek czasu. Wykonuje tranzycję do stanu C_i, ustawiając v na swój numer i i ponownie resetując zegarek x_i.

4. Do stanu CS_i może przejść tylko wtedy, jeśli $v = i$ i jeśli przebywał w stanie C_i co najmniej c jednostek czasu.

Intuicyjnie:

- Proces i musi przejść z A_i do B_i wcześniej niż którykolwiek z innych procesów osiągnie stan C.
- Każdy proces, który dotrze do stanu C musi poczekać, aż wszystki procesy będące w stanie B osiągną stan C.
- Ostatni proces, który dotrze do stanu C i ustawi v na swój numer zyskuje prawo wejścia do sekcji krytycznej.



Rysunek 2: Automat z czasem reprezentujący protkół Fischera

1.3 Prosty system zarządzania koleją

Rozważmy prosty system zarządzania pociągami, które mają przekroczyć jakiś strategiczny punkt, np.: most. Chcielibyśmy wykorzystać automatycznego strażnika, który będzie w stanie bezpiecznie kierować ruchem pociągów jadących (z różnych kierunków) przez jeden most, zamiast np.: budować kilkanaście nowych mostów. Oczywistym celem, który chcemy uzyskać jest to, aby zawsze na moście przebywał co najwyżej jeden pociąg.

Do komunikacji między procesami kontrolera i pociągów wykorzystamy kanały rozgłoszeniowe:

$near_i$ oznaczający, że pociąg już jest blisko

$stop_i$ oznaczający, że pociąg ma się zatrzymać

$leave_i$ oznaczający, że pociąg opuszcza most

go_i oznaczjący, że pociąg może wjechać na most

Kontroler będzie trzymał listę oczekujących pociągów L.

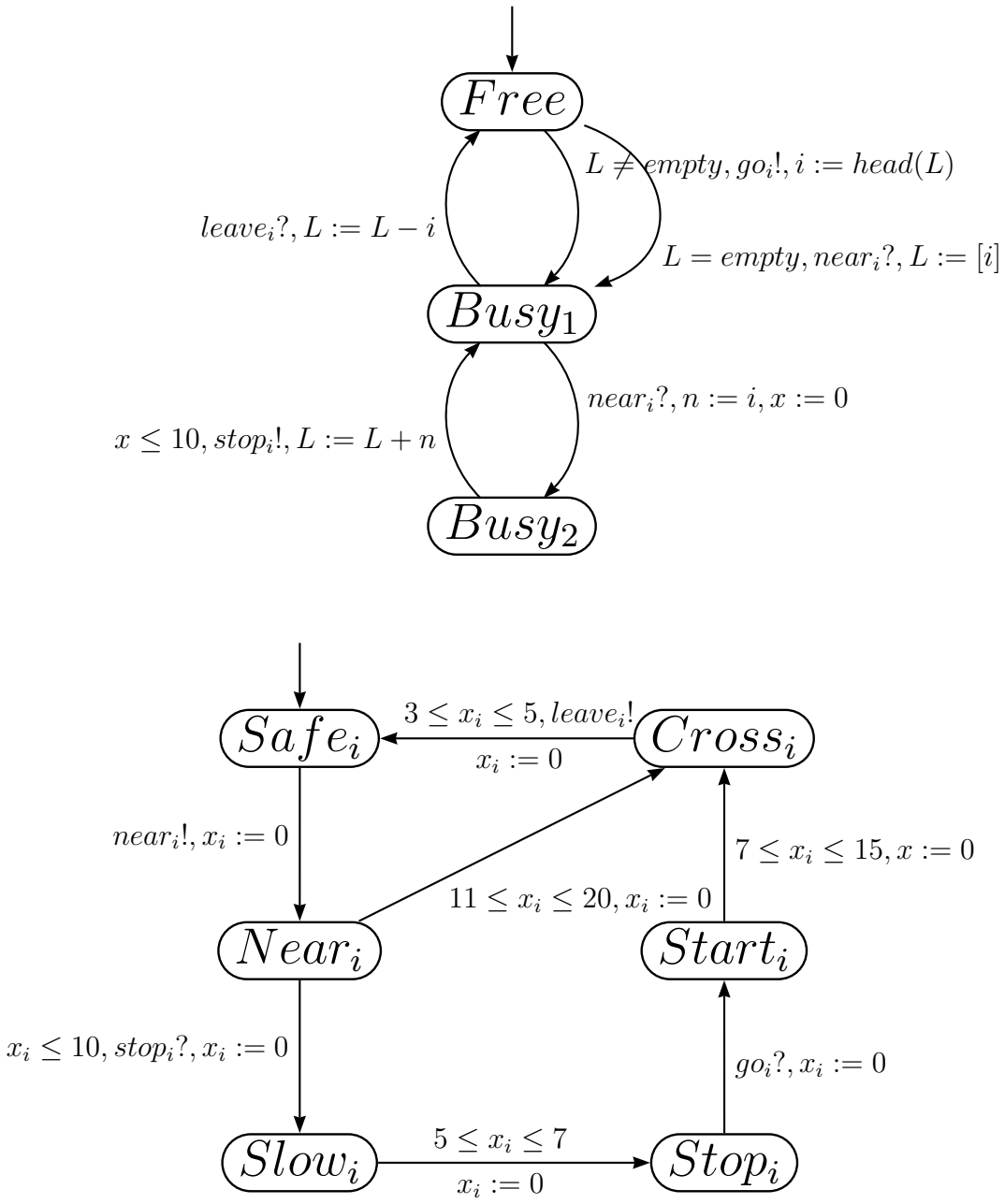
$$
\begin{aligned}
Train_i &\stackrel{def}{=} Safe_i \\
Safe_i &\stackrel{def}{=} (true, near_i!, x_i).Near_i \\
Near_i &\stackrel{def}{=} (x_i \geq 0 \wedge x_i \leq 10, stop_i?, x_i).Slow_i + (x_i \geq 11 \wedge x_i \leq 20, \epsilon, x_i).Cross_i \\
Cross_i &\stackrel{def}{=} (x_i \geq 3 \wedge x_i \leq 5, leave_i!, x_i).Safe_i \\
Slow_i &\stackrel{def}{=} (x_i \geq 5 \wedge x_i \leq 7, \epsilon, x_i).Stop_i \\
Stop_i &\stackrel{def}{=} (true, go_i?, x_i).Start_i \\
Start_i &\stackrel{def}{=} (x_i \geq 7 \wedge x_i \leq 15, \epsilon, x_i).Cross_i
\end{aligned}
$$

$$
\begin{aligned}
Controller &\stackrel{def}{=} Busy_1 \\
Busy_1 &\stackrel{def}{=} (true, leave_i?, L := L - i).Free + (true, near_i?, n := i, x).Busy_2 \\
Busy_2 &\stackrel{def}{=} (x \leq 10, stop_n!, L := L + n).Busy_1 \\
Free &\stackrel{def}{=} (L = empty, near_i?, L := [i]).Busy_1 + (L \neq empty, go_i!, i := head(L)).Busy_1
\end{aligned}
$$

Rysunek 3: Formalny opis protokołu zarządzania koleją

Aby formalnie opisać powyższy protokół (rysunek 3), wykorzystałam algebrę procesów. Wyrażenie $(g, \alpha, \phi).P$ oznacza proces, który może wykonać przejście α, o ile warunek g jest spełniony, przy prejściu resetując zegarki ze zbioru ϕ. P jest kontynuacją, czyli w naszym przypadku stanem po wykonaniu przejścia α.

Po przekształceniu do automatu z czasem wygląda to tak ja rysunku 4.



Rysunek 4: Automaty z czasem reprezentujące protkół zarządzania koleją

2 UPPAAL

UPPAAL jest narzędziem przeznaczonym do walidacji i weryfikacji systemów czasu rzeczywistego.

Jego główną ideą jest modelowanie systemu przy pomocy automatów z czasem, symulowanie jego zachowania oraz weryfikacja właściwości. Symulacja polega na interaktywnym uruchomieniu systemu i sprawdzeniu, czy system zachowuje się jak należy. Na weryfikację właściwości składają się najczęściej:

- sprawdzenie czy dany wierzchołek jest osiągalny
- sprawdzenie czy w sekcji krytycznej zawsze przebywa co najwyżej jeden proces.

Weryfikacja z grubsza polega na przeszukaniu wszystkich możliwych ścieżek zachowania systemu.

System w UPPAALu składa się z wielu współbieżnych procesów modelowanych jako automaty. Do każdego z automatów przyporządkowany jest pewien zbiór lokacji. Przejścia między poszczególnymi lokacjami mogą być kontrolowane poprzez:

- strażników - przejście może mieć miejsce jeśli pewien warunek między zmienną a zegarkiem jest spełniony
- mechanizm synchronizacji - dwa lub więcej procesów może dokonać tranzycji, jeśli jeden z nich ma $a!$, a pozostałe $a?$, gdzie a jest kanałem synchronizacyjnym.

Poza tym każdemu przejściu może towarzyszyć ustawienie zmiennych lub/i resetowanie zagarków.

2.1 Stany

W UPPAALu występują 3 rodzaje stanów:

1. normalne z lub bez ograniczeń czasowych
2. *urgent* - gdy do niego wejdziemy, zatrzymuje się czas w całym systemie; możemy wtedy wykonywać przejścia między stanami, jednak bez upływu czasu.
3. *commit* - stan ten musi być opuszczony natychmiast po wejściu.

Dodatkowo możemy oznaczyć stan jako początkowy, ale nie ma to wpływu na wykonywane przejścia.

2.2 Sprawdzanie własności systemu

W zakładce weryfikatora możemy konstruować różnego typu zapytania i tym samym badać właściwości naszego systemu. Zapytania możemy zapamiętywać, komentować i w razie chęci lub potrzeby zweryfikować. Dopuszczalne są następujące wyrażenia:

- E<> p - istnieje ścieżka taka, że p może zajść
- A[] p - dla wszystkich ścieżek, p zawsze zachodzi
- E[] p - istnieje ścieżka taka, że p zawsze zachodzi
- A<> p - dla wszystkich ścieżek, p może zajść
- p –> q - zawsze kiedy zajdzie p może też zajść q

gdzie p i q są wyrażeniami postaci ($P.stan_i \; and \; x \leq 10$), czyli formalniej:

$$T := Proces.stan | zegarek \preceq stala | zmienna \preceq stala | T_1 \; and \; T_2 | T_1 \; imply \; T_2$$

gdzie $\preceq \in \{<, \leq, >, \geq, =, \neq\}$.

W szczególności przydatne jest wyrażenie $A[] \; not \; deadlock$, które sprawdza czy nie ma zakleszczeń w naszym systemie.

Więcej informacji można znaleźć na stronie domowej UPPAALa i w pozycjach [1], [2], [3] oraz wielu innych.

Literatura

[1] Uppaal2k: Small tutorial.

[2] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 232–243. Springer–Verlag, October 1995.

[3] Wang Yi, Paul Pettersson, and Mats Daniels. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In Dieter Hogrefe and Stefan Leue, editors, *Proc. of the 7*th *Int. Conf. on Formal Description Techniques*, pages 223–238. North–Holland, 1994.