

Zadanie zaliczeniowe z Prologu (10 pkt.)

1 Wprowadzenie

Deterministyczny automat skończony (DFA) definiujemy jako krotkę:

$$\langle Q, \Sigma, \delta, q_0, F \rangle$$

gdzie:

- Q jest skończonym zbiorem stanów
- Σ jest skończonym alfabetem
- $\delta : Q \times \Sigma \rightarrow Q$ jest funkcją przejścia (całkowitą)
- $q_0 \in Q$ jest stanem początkowym
- $F \subseteq Q$ jest zbiorem stanów akceptujących

Powiemy, że automat *akceptuje* słowo $w = a_1 a_2 \dots a_n$, $n \geq 0$ jeśli:

$$\delta(q_0, a_1) = q_1, \delta(q_1, a_2) = q_2, \dots, \delta(q_{n-1}, a_n) = q_n$$

oraz stan q_n jest stanem akceptującym.

Język akceptowany przez automat A definiujemy jako zbiór słów akceptowanych przez ten automat:

$$L(A) = \{w : \text{automat } A \text{ akceptuje słowo } w\}.$$

2 Specyfikacja

Deterministyczny automat skończony reprezentujemy za pomocą termów ustalonych postaci:

`dfa(FunkcjaPrzejścia, StanPoczątkowy, ZbiórStanówAkceptujących)`,

gdzie:

`FunkcjaPrzejścia` jest listą termów postaci `fp(S1, C, S2)` oznaczających,

że: $\delta(S1, C) = S2$,

`StanPoczątkowy` jest początkowym stanem automatu,

`ZbiórStanówAkceptujących` jest listą (bez powtórzeń) wszystkich stanów akceptujących danego automatu.

Zdefiniować następujące predykaty:

1. `correct(+Automat, -Reprezentacja)`

Odnosi sukces wtedy i tylko wtedy, gdy `Automat` jest termem ustalonym reprezentującym deterministyczny automat skończony oraz `Reprezentacja` jest termem reprezentującym wewnętrzną postać automatu (na potrzeby sprawnych obliczeń).

2. `accept(+Automat, ?Słowo)`

Odnosi sukces wtw, gdy podany `Automat` akceptuje słowo `Słowo`.

Parametr `Słowo` może być termem ustalonym (listą elementów), jak również termem nie w pełni ustalonym.

W przypadku gdy `Słowo` nie jest termem ustalonym, ale jest listą zamkniętą, predykat powinien być generatorem słów zgodnych z podanym wzorcem i akceptowanych przez podany automat.

W przypadku gdy `Słowo` jest zmienną predykat powinien być generatorem wszystkich słów należących do języka akceptowanego przez podany automat. W szczególności oznacza to, że każde słowo należące do języka powinno zostać wygenerowane w skończonym czasie.

3. `empty(+Automat)`

Odnosi sukces wtw, gdy język akceptowany przez podany automat jest pusty.

4. `equal(+Automat1, +Automat2)`

Odnosi sukces wtw, gdy $L(\text{Automat1}) = L(\text{Automat2})$ oraz alfabety obu automatów są równe.

5. `subsetEq(+Automat1, +Automat2)`

Odnosi sukces wtw, gdy $L(\text{Automat1}) \subseteq L(\text{Automat2})$ oraz alfabety obu automatów są równe.

3 Orientacyjna punktacja

Ocenie podlega zarówno poprawność rozwiązania, jak i sprawność oraz jakość kodu programu (czyli poprawne, właściwe używanie konstrukcji programistycznych dostępnych w Prologu).

5 pkt. – definicja predykatów `correct/2`, `accept/2`, `empty/1`

5 pkt. – definicja predykatów `equal/2`, `subsetEq/2`

4 Przesłanie rozwiązania

Rozwiązanie zadania powinno składać się z jednego pliku o nazwie `<identyfikator_studenta>.pl` (np. `ab123456.pl`), który należy przesłać przez modle'a.

Pierwszy wiersz pliku powinien zawierać komentarz z imieniem i nazwiskiem autora. W dalszej części komentarza należy umieścić zwięzły opis przyjętej wewnętrznej reprezentacji automatu.

5 Ważne uwagi dodatkowe

1. Programy muszą poprawnie działać pod SICStus Prologiem na komputerze students.

Programy, które nie będą poprawnie kompilowały się (działały) pod SICStus Prologiem nie uzyskają maksymalnej oceny (choćby poprawnie kompilowały się i działały pod inną wersją Prologu).

2. W rozwiązaniu wolno korzystać wyłącznie:
 - z predykatów, konstrukcji przedstawionych na wykładzie
 - z wbudowanych predykatów (np. `member/2`, `append/3`, `length/2`)
 - ze standardowej biblioteki SICStus Prologu o nazwie `lists` (ładowanie: `:- use_module(library(lists)).`)
3. Nie wolno korzystać:
 - z żadnych innych bibliotek (oprócz biblioteki `lists`),
 - z (wbudowanych) predykatów nieprzedstawionych na wykładzie.
4. W programie wolno (poprawnie) używać negacji, odcięcia, konstrukcji `if-then-else`, predykatu `if/3` itp.
5. Program powinien być czytelnie sformatowany, m.in. długość każdego wiersza nie powinna przekraczać 80 znaków. Sposób formatowania programów w Prologu (definicja algorytmu QuickSort):

```
qsort([], []).
qsort([X | L], S) :-          % komentarz niezasłaniający kodu
    partition(L, X, M, W),    % podział listy na podlisty
    qsort(M, SM),             % sortowanie podlist
    qsort(W, SW),
    append(SM, [X|SW], S).    % scalenie wyników
```

6. Program powinien zawierać (krótkie, zwięzłe) **komentarze** opisujące (deklaratywne) znaczenie ważniejszych predykatów pomocniczych oraz przyjęte (podstawowe) rozwiązania.

6 Przykłady

6.1 Przykładowe automaty

```
% example(IdentyfikatorAutomatu, Automat)

example(a11, dfa([fp(1,a,1),fp(1,b,2),fp(2,a,2),fp(2,b,1)], 1, [2,1])).
example(a12, dfa([fp(x,a,y),fp(x,b,x),fp(y,a,x),fp(y,b,x)], x, [x,y])).
example(a2, dfa([fp(1,a,2),fp(2,b,1),fp(1,b,3),fp(2,a,3),
    fp(3,b,3),fp(3,a,3)], 1, [1])).
example(a3, dfa([fp(0,a,1),fp(1,a,0)], 0, [0])).
example(a4, dfa([fp(x,a,y),fp(y,a,z),fp(z,a,x)], x, [x])).
example(a5, dfa([fp(x,a,y),fp(y,a,z),fp(z,a,zz),fp(zz,a,x)], x, [x])).
```

```

example(a6, dfa([fp(1,a,1),fp(1,b,2),fp(2,a,2),fp(2,b,1)], 1, [])).
example(a7, dfa([fp(1,a,1),fp(1,b,2),fp(2,a,2),fp(2,b,1),
                fp(3,b,3),fp(3,a,3)], 1, [3])).

% bad ones
example(b1, dfa([fp(1,a,1),fp(1,a,1)], 1, [])).
example(b2, dfa([fp(1,a,1),fp(1,a,2)], 1, [])).
example(b3, dfa([fp(1,a,2)], 1, [])).
example(b4, dfa([fp(1,a,1)], 2, [])).
example(b4, dfa([fp(1,a,1)], 1, [1,2])).
example(b5, dfa([], [], [])).

```

Automaty **a11** i **a12** akceptują wszystkie słowa postaci $\{a,b\}^*$, czyli wszystkie słowa nad alfabetem $\Sigma = \{a,b\}$.

Automat **a2** akceptuje słowa postaci $(ab)^n, n \geq 0$.

Automat **a3** akceptuje słowa postaci $(aa)^n, n \geq 0$, automat **a4** słowa postaci $(aaa)^n, n \geq 0$, zaś automat **a5** słowa postaci $(aaaa)^n, n \geq 0$.

Języki akceptowane przez automaty **a6** oraz **a7** są puste, gdyż zbiór stanów akceptujących automatu **a6** jest pusty, a jedyny stan akceptujący automatu **a7** nie jest osiągalny ze stanu początkowego.

6.2 Przykładowe testy

Następujące zapytania powinny odnieść sukces:

```

example(a11, A), example(a12, B), equal(A, B).
example(a2, A), example(a1, B), subsetEq(A, B).
example(a5, A), example(a3, B), subsetEq(A, B).
example(a6, A), empty(A).
example(a7, A), empty(A).
example(a2, A), accept(A, []).
example(a2, A), accept(A, [a,b]).
example(a2, A), accept(A, [a,b,a,b]).

```

Następujące zapytania powinny odnieść porażkę:

```

example(b1, A), correct(A, _).    % b2, b3, b4, b5 - też porażki
example(a2, A), empty(A).
example(a3, A), example(a4, B), equal(A, B).
example(a4, A), example(a3, B), subsetEq(A, B).
example(a2, A), accept(A, [a]).

```

Na zapytanie: `example(a1, A), accept(A, [X,Y,Z])` powinno zostać udzielonych 8 odpowiedzi, odpowiadających wszystkim 3-literowym słowom nad dwuelementowym alfabetem.

Na zapytanie: `example(a1, A), accept(A, Var)` odpowiedzi takie jak na przykład `Var = [a,b]`, `Var = [b,a,b]` powinny zostać udzielone w skończonym czasie.