Zadanie zaliczeniowe z Prologu (8 pkt.)

Analiza składniowa metodą LR

Dla podanej gramatyki zbudować, o ile jest to możliwe, automat typu LR(0) oraz sprawdzić, korzystając z utworzonego automatu, czy podane słowo należy do języka generowanego przez tę gramatykę.

W przypadku, gdy dla podanej gramatyki nie można zbudować automatu LR(0) należy podać przyczynę, czyli opisać istniejący konflikt. Wystarczy podać jeden konflikt.

Specyfikacja

Zdefiniować następujące predykaty:

1. `createLR(+Gramatyka, -Automat, -Info)`

   Dla podanej gramatyki tworzy automat LR(0). W przypadku utworzenia automatu parametr `Info` powinien mieć nadaną wartość `yes`, a w przeciwnym przypadku jego wartością powinien być term `konflikt(Opis)`, gdzie `Opis` jest (czytelnym) opisem rozpoznanego konfliktu, a parametr Automat powinien mieć wówczas nadaną wartość `null`.

2. `accept(+Automat, +Słowo)`

   Sukces wtw, gdy podane `Słowo` należy do języka generowanego przez gramatykę, dla której został utworzony podany `Automat`.

Składnia gramatyki

Zbiór produkcji gramatyki reprezentujemy za pomocą listy, której elementy są postaci:

`prod(NazwaNieterminala, ListaPrawychStronProdukcji)`,

gdzie `ListaPrawychStronProdukcji` jest listą wszystkich prawych stron produkcji dla podanego nieterminala. Prawa strona produkcji jest reprezentowana jako lista, której elementami są symbole nieterminalne, reprezentowane jako termy `nt(Nieterminal)`, gdzie `Nieterminal` jest prologową stałą, oraz symbole terminalne (prologowe stałe).

Reprezentacją gramatyki jest term postaci:
`gramatyka(SymbolPoczątkowy, ZbiórProdukcji)`.

Przykład.
Gramatyka:

```
E -> E + T | T
T -> id | ( E )
```

jest reprezentowana za pomocą termu:

```
gramatyka('E', [prod('E', [[nt('E'),+,nt('T')], [nt('T')]]),
                prod('T', [[id], ['(', nt('E'), ')']]) ]).
```

Założenia

Reprezentacja gramatyki jest poprawna.
W gramatyce nie występują:
– symbol nieterminalny o nazwie `Z`,
– symbol terminalny `#`.

Orientacyjna punktacja

6 pkt. - (poprawna, dobra) definicja predykatu `createLR/3`
2 pkt. - (poprawna, dobra) definicja predykatu `accept/2`

Przesłanie rozwiązania

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

Pierwszy wiersz pliku powinien zawierać komentarz z imieniem i nazwiskiem autora. W następnych wierszach należy umieścić komentarz zawierający (zwięzły, pełny) opis przyjętej reprezentacji automatu LR.

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 oraz przyjęte (podstawowe) rozwiązania.

Przykładowe testy

Wyniki przykładowych testów, dla podanego poniżej (prostego) predykatu testującego oraz jednej ze zdefiniowanych poniżej przykładowych gramatyk.

```
?- test(ex1, [[id], ['(',id,')'], [id,'+',ident], [id,'+',id]]).
  slowo: [id] nalezy.
  slowo: [(,id,)] nalezy.
  slowo: [id,+,ident] NIE nalezy.
  slowo: [id,+,id] nalezy.
Koniec testu.
```

```
% test(+NazwaGramatyki, +ListaSlowDoZbadania)
test(NG, ListaSlow) :-
     grammar(NG, G),
     createLR(G, Automat, yes),
     checkWords(ListaSlow, Automat).

checkWords([], _) :-  write('Koniec testu.\n').
```

```
checkWords([S|RS], Automat) :-
      format("  Slowo: ~p ", [S]),
      (accept(Automat, S) -> true; write('NIE ')),
      write('nalezy.\n'),
      checkWords(RS, Automat).
```

Przykładowe gramatyki

```
                                                  % LR(0)
grammar(ex1, gramatyka('E',
               [prod('E', [[nt('E'), '+', nt('T')],  [nt('T')]]),
                prod('T', [[id],  ['(', nt('E'), ')']])   ])).

                                                  % LR(0)
grammar(ex2, gramatyka('A', [prod('A', [[nt('A'), x], [x]])])).

                                                  % SLR(1)
grammar(ex3, gramatyka('A', [prod('A', [[x, nt('A')], [x]])])).

                                                  % nie SLR(1)
grammar(ex4, gramatyka('A',
               [prod('A', [[x, nt('B')], [nt('B'), y], []]),
                prod('B', [[]])])).

                                                  % nie SLR(1)
grammar(ex5, gramatyka('S',
               [prod('S', [[id], [nt('V'), ':=', nt('E')]]),
                prod('V', [[id], [id, '[', nt('E'), ']']]),
                prod('E', [[v]])])).

                                                  % nie SLR(1)
grammar(ex6, gramatyka('A',
                [prod('A', [[x], [nt('B'), nt('B')]]),
                 prod('B', [[x]])])).
```