Zadanie zaliczeniowe z Prologu (10 pkt.)

Analizator prostych programów - wprowadzenie

Analizujemy proste programy sekwencyjne, których kod jest przedstawiony jako (prologowa) lista instrukcji. W systemie może działać dowolna liczba procesów o jednakowym kodzie. Zadanie polega na napisaniu programu w Prologu sprawdzającego spełnienie warunku bezpieczeństwa, czyli sprawdzeniu czy istnieje taki przeplot wykonania procesów, w którym przynajmniej dwa procesy znajdą się równocześnie w sekcji krytycznej.

Specyfikacja

Stałe: liczby całkowite.
Zmienne: proste (typu całkowitoliczbowego) oraz tablice o elementach typu całkowitoliczbowego. Każda tablica jest jednowymiarowa, rozmiaru równego liczbie procesów w systemie. Tablice indeksujemy od 0. Wszystkie zmienne (w tym elementy tablic) są inicjowane na zero. Wszystkie zmienne są globalne, czyli dostępne dla wszystkich procesów działających w systemie.

Ponadto każdy proces ma dostęp do stałej o nazwie `pid`, której wartością jest identyfikator procesu, będący liczbą z zakresu 0..N-1, gdzie N jest liczbą wszystkich procesów w systemie.

Wyrażenia (arytmetyczne i logiczne)

```
wyrArytm    ::=  wyrProste | wyrProste oper wyrProste
wyrProste   ::=  liczba | zmienna
zmienna     ::=  ident | array(ident, wyrArytm)
oper        ::=  + | - | * | /
wyrLogiczne ::=  wyrProste operRel wyrProste
operRel     ::=  < | = | <>
```

Nazwy wszystkich zmiennych (prostych, tablicowych) to prologowe stałe (niebędące liczbami), np. x, k, chce, 'A'.

Instrukcje

- `assign(zmienna, wyrArytm)`
  Przypisanie na podaną zmienną (prostą lub element tablicy) wartości podanego wyrażenia. Przejście do następnej instrukcji.
- `goto(liczba)`
  Skok (bezwarunkowy) do instrukcji o podanym indeksie.

- `condGoto(wyrLogiczne, liczba)`
  Skok warunkowy (jeśli wartością logiczną wyrażenia jest prawda) do instrukcji o podanym indeksie, a wpp. przejście do następnej instrukcji.

- `sekcja`
  Sekcja krytyczna. Przejście do następnej instrukcji.

Instrukcje indeksujemy od 1. Każdy proces rozpoczyna działanie od instrukcji o indeksie 1.

Założenia

Zakładamy, że wszystkie zmienne przyjmują wartości z pewnego ograniczonego zakresu (np. 0..N, gdzie N to liczba procesów), chociaż tego zakresu jawnie nie podajemy i nie musimy tego sprawdzać. A zatem liczba stanów naszego systemu jest zawsze skończona.

Zakładamy poprawność wykonania programów, tzn. poprawność odwołań do tablic, poprawność obliczania wartości wyrażeń arytmetycznych, poprawność skoków (czyli np. brak dzielenia przez zero, wyjścia poza zakres tablicy, skoku do nieistniejącej instrukcji).

Każdy proces działa w pętli nieskończonej (ostatnią instrukcją każdego programu jest instrukcja skoku bezwarunkowego).

Zadanie

Napisz w Prologu procedurę `verify/2` wywoływaną w następujący sposób:

`verify(N, Program)`,

gdzie: `N` – liczba procesów działających w systemie (>= 1),
`Program` – nazwa pliku z programem (stała).

Procedura powinna sprawdzać poprawność argumentów wywołania (m.in. liczbę procesów, dostępność pliku z programem). Można natomiast założyć poprawność danych wejściowych, czyli samego pliku z programem.

Dla poprawnych danych program powinien wypisać:
– informację o spełnieniu (bądź nie) warunku bezpieczeństwa
– w przypadku braku bezpieczeństwa wypisać przykładowy niepoprawny przeplot wraz z informacją
  – które procesy znajdują się w sekcji krytycznej (indeksy procesów)
  – i (nieobowiązkowo) numer (kolejny) stanu, w którym nastąpiło złamanie bezpieczeństwa (stany liczymy i numerujemy od 1).

Zdefiniuj ponadto predykat `verify/0`, którego zadaniem jest pobranie argumentów z wiersza polecenia, sprawdzenie ich poprawności i wykonanie głównego przetwarzania (czyli wywołanie predykatu `verify/2`).

Plik wykonywalny można wówczas uzyskać pod SWI-Prologiem wykonując następujące polecenie:

```
swipl --goal=verify --stand_alone=true -o <exe> -c <plik.pl>
```

Przypomnienie

Niepoprawny przeplot jest to taka sekwencja wykonania instrukcji programu przez procesy działające w systemie, w wyniku której dwa procesy mogą jednocześnie przebywać w sekcji krytycznej. Proces może wejść do sekcji krytycznej, jeśli jego licznik rozkazów wskazuje na instrukcję `sekcja`. Jeśli licznik rozkazu procesu wskazuje na instrukcję bezpośrednio za instrukcją `sekcja`, to znaczy, że proces już wyszedł z sekcji (czyli jest bezpiecznie).

Na przykład dla programu złożonego z instrukcji:

```
[assign(x, pid), sekcja, goto(1)])
```

niepoprawnym przeplotem jest m.in. następująca sekwencja instrukcji

```
Proces 1:  1
Proces 0:  1
```

Proces nr 1 wykonał instrukcję nr 1 (pierwszą instrukcję programu), następnie proces nr 0 wykonał tę samą instrukcję. Liczniki rozkazów obu procesów są w tym momencie równe 2, czyli równe adresowi instrukcji `sekcja`, więc każdy proces może wejść (wchodzi) do sekcji krytycznej.

Specyfikacja predykatów pomocniczych

Oprócz definicji predykatu głównego (`verify/2`) program powinien zawierać definicje wymienionych poniżej predykatów, przy czym dopuszczalne są drobne zmiany techniczne (np. dodanie jakiegoś parametru).

1. `initState(+Program, +N, -StanPoczątkowy)`

   `Program` – reprezentacja (termowa) programu
   `N` – liczba procesów w systemie
   `StanPoczątkowy` – reprezentacja stanu początkowego.

   Uwaga. W tekście programu powinien być umieszczony komentarz opisujący przyjętą reprezentację stanu systemu.

2. `step(+Program, +StanWe, ?PrId, -StanWy)`

   `Program` – reprezentacja (termowa) programu

   `StanWe` - informacja o stanie systemu (wartości wszystkich zmiennych oraz liczniki rozkazów wszystkich procesów)

   `StanWy` – informacja o stanie systemu po wykonaniu bieżącej instrukcji przez proces o identyfikatorze `PrId`.

Format pliku z programem

Plik tekstowy postaci:

```
variables(ListaNazwZmiennychProstych).
arrays(ListaNazwZmiennychTablicowych).
program(ListaInstrukcji).
```

Wszystkie listy są podawane w notacji prologowej.

Przykładowe programy

1. Implementacja algorytmu Petersona w zdefiniowanym powyżej języku (z lewej, w nawiasach, indeksy instrukcji).

```
(1)   assign(array(chce, pid), 1)
(2)   assign(k, pid)
(3)   condGoto(array(chce, 1-pid) = 0, 5)
(4)   condGoto(k = pid, 3)
(5)   sekcja
(6)   assign(array(chce, pid), 0)
(7)   goto(1)
```

   Reprezentacja powyższego programu (plik `'peterson.txt'`):

```
variables([k]).
arrays([chce]).
program([assign(array(chce, pid), 1),
         assign(k, pid),
         condGoto(array(chce, 1-pid) = 0, 5),
         condGoto(k = pid, 3),
         sekcja,
         assign(array(chce, pid), 0),
         goto(1)]).
```

2. Bardzo prosty niepoprawny program (`'unsafe.txt'`).

```
variables([x]).
arrays([]).
program([assign(x, pid), sekcja, goto(1)]).
```

Orientacyjna punktacja

5 pkt. - (poprawna, dobra) definicja predykatu `step/4`
2 pkt. - binarna informacja czy system spełnia warunek bezpieczeństwa
3 pkt. - znalezienie i wypisanie niepoprawnego przeplotu
-1 pkt - brak (pełnego) opisu wybranej reprezentacji stanu systemu

Ważne uwagi dodatkowe

1. **Programy muszą poprawnie działać pod SWI Prologiem na komputerze students. Programy niespełniające powyższego kryterium nie będą sprawdzane.**

2. W programie wolno korzystać ze standardowych predykatów Prologu używanych na ćwiczeniach (np. `member/2, append/3`) oraz z biblioteki o nazwie `lists` (operacje na listach).
   (Załadowanie biblioteki: na początku pliku źródłowego należy umieścić dyrektywę `:- ensure_loaded(library(lists))`.

   Nie wolno korzystać z predykatów `findall/3 (bagof/3, setof/3)`. Programy korzystające z jakiegokolwiek predykatu powyższego rodzaju będą oceniane w skali 0-4 pkt.

3. Nie jest wymagana optymalizacja, czyli można używać wyłącznie prostszych (czytaj: droższych) struktur danych, np. prologowych list.

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.

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 nazwiskiem autora (anonimów nie czytamy ;).

Przykładowe wyniki analizy

```
?- verify(2, 'peterson.txt').
Program jest poprawny (bezpieczny).

?- verify(2, 'peterson-bad0.txt').
Program jest niepoprawny.
Niepoprawny przeplot:
   Proces 0: 1
   Proces 1: 1
   Proces 1: 2
   Proces 1: 3
   Proces 0: 2
   Proces 0: 3
   Proces 0: 4
Procesy w sekcji: 1, 0.

?- verify(2, dowol).
Error: brak pliku o nazwie - dowol

?- verify(0, 'unsafe.txt').
Error: parametr 0 powinien byc liczba > 0
```