Programowanie

Egzamin poprawkowy

17 września 2002

Za każde zadanie można otrzymać od -25 do 25 punktów. Egzamin trwa trzy godziny zegarowe.

Zadanie 1. Zbuduj co najwyżej pięciostanowy automat skończony rozpoznający język opisany wyrażeniem regularnym

$$\left((ab)^*a + bb^*a + abb^*a\right)^*$$

Uzasadnij poprawność swojej konstrukcji. Uwaga: za automat więcej niż pięciostanowy nie można otrzymać dodatnich punktów.

Zadanie 2. Zaprogramuj w Adzie zadanie `BUFOR` o części publicznej

```
task BUFOR is
   entry WSTAW (X : in INTEGER);
   entry POBIERZ (X : out INTEGER);
end BUFOR;
```

implementujący bufor `ROZMIAR`-elementowy (gdzie `ROZMIAR` jest pewną stałą — możesz ją uczynić parametrem rodzajowym implementacji) elementów typu `INTEGER` (możesz uogólnić typ elementu podając go jako parametr rodzajowy). Wewnętrzną reprezentacją bufora powinna być tablica `ROZMIAR`-elementowa. Uwaga: nie chodzi tu o znajomość detali składniowych (błędy składniowe nie będą karane), lecz o rozumienie mechanizmu spotkań w Adzie.

Zadanie 3. Napisz w Prologu predykat `primes/1` implementujący sito Eratostenesa. Cel `primes`(t) powinien być spełniony, jeśli a) t jest termem stałym reprezentującym wyrażenie arytmetyczne, którego wartością jest liczba pierwsza lub b) gdy t jest nieukonkretnioną zmienną. W przypadku b) przy nawrotach predykat powinien podstawiać pod t kolejne liczby pierwsze. Cel `primes`(t) powinien zawieść, jeżeli t jest termem stałym reprezentującym wyrażenie arytmetyczne, którego wartością jest liczba złożona. W przypadku, gdy t nie jest termem stałym reprezentującym wyrażenie arytmetyczne ani nieukonkretnioną zmienną, wywołanie predykatu powinno zakończyć się błędem arytmetycznym i zerwaniem obliczeń. Możesz korzystać z predykatów: `var/1` sprawdzającego, czy jego parametr jest nieukonkretnioną zmienną i `integer/1` sprawdzającego, czy jego parametr jest atomem całkowitoliczbowym oraz standardowych predykatów, takich jak np. `<` (arytmetycznie mniejsze), czy `=\=` (arytmetycznie różne) i funkcji arytmetycznych, takich, jak `+`, czy `mod`. Zadbaj o to, by predykat działał efektywnie (nie generował wielu nieużytków itp.). Uwaga: program powinien zawierać nie więcej niż 15 klauzul i faktów. Za dłuższe programy nie można otrzymać punktów dodatnich.

Zadanie 4. Napisz w SML-u funkcję `perm : 'a list -> 'a list list` wyznaczającą listę wszystkich permutacji swojego argumentu. Możesz użyć funkcji

```
foldl : ('a * 'b -> 'b) -> 'b -> 'a list -> 'b
```