Programowanie

Egzamin poprawkowy

11 września 2001

W treści zadań podane są liczby punktów (tłustym drukiem w nawiasach), które można uzyskać za ich rozwiązanie. Jeżeli punktacja za zadanie wynosi n, oznacza to, że za to zadanie można otrzymać od $-n$ do n punktów. Punkty ujemne będą przyznawane za umieszczenie w rozwiązaniu odpowiedzi kompromitująco fałszywych. Za brak rozwiązania zadania otrzymuje się 0 punktów. Punktacja za zadania przelicza się na oceny według poniższej tabeli:

$-50 \div 19$	ndst
$20 \div 24$	dst
$25 \div 29$	dst+
$30 \div 34$	db
$35 \div 39$	db+
$40 \div 50$	bdb

Zadanie 1 (20). Niech $m \geq 0$ będzie ustaloną liczbą naturalną. Rozważmy następujący język nad alfabetem $\{0, 1\}$:

$$L_m \;=\; \{u0w \mid u, w \in \{0,1\}^*, |w| = m\}$$

Jest to zbiór tych ciągów zerojedynkowych, w których na $m+1$-szej pozycji od końca występuje 0. Zbuduj (**10**) automat skończony akceptujący ten język. Uwaga: proszę nie rysować tego automatu, nawet dla ustalonego m, tylko podać definicje pięciu elementów $\langle \Sigma, Q, q_0, F, \delta \rangle$, z których on się składa. Pokaż (**10**), że każdy automat, który akceptuje język L_m ma co najmniej 2^{m+1} stanów.

Zadanie 2 (10). Rozważmy sygnaturę jednogatunkową $\Sigma = \{c, f, g, \oplus\}$, gdzie c jest stałą, f i g unarnymi symbolami funkcyjnymi, zaś $\oplus$ symbolem binarnym zapisywanym infiksowo. Niech $x, y, z, \ldots$ będą zmiennymi. Rozważmy następującą specyfikację równościową (przyjmujemy standardową semantykę algebry początkowej):

$$f(x) \oplus y \;=\; f(x \oplus y) \tag{1}$$
$$g(x) \oplus y \;=\; g(x \oplus y) \tag{2}$$
$$c \oplus y \;=\; y \tag{3}$$

Wskaż (**2**) zbiór konstruktorów dla tej specyfikacji równościowej. Zapisz (**2**) zasadę indukcji strukturalnej. Udowodnij (**6**) z jej pomocą, że $(x \oplus y) \oplus z = x \oplus (y \oplus z)$.

Zadanie 3 (10). Oto abstrakcyjna składnia pewnego języka programowania, w której $\mathcal{T}$ jest kategorią składniową opisującą *wyrażenia typowe*, $\mathcal{V}$ — *zmienne*, zaś $\mathcal{E}$ — *wyrażenia*:

$$\begin{aligned}
\mathcal{T} &= \texttt{int} \mid \mathcal{T} \to \mathcal{T} \\
\mathcal{V} &= x \mid y \mid z \mid \ldots \\
\mathcal{E} &= \texttt{0} \mid \texttt{1} \mid \mathcal{V} \mid \mathcal{E} \texttt{ + } \mathcal{E} \mid \texttt{fn}\,(\mathcal{V} : \mathcal{T}) \texttt{ => } \mathcal{E} \mid \mathcal{E}\mathcal{E}
\end{aligned}$$

Pod względem typów język ten przypomina Pascal — nie ma w nim polimorfizmu (zmiennych typowych), a typy trzeba jawnie pisać przy wszystkich parametrach formalnych funkcji. Reguły typowania są również podobne, jak w Pascalu (ten język jest jednak bardziej skomplikowany od Pascala, pozwala bowiem na używanie funkcji wyższych rzędów):

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma} \quad \frac{}{\Gamma \vdash \texttt{0} : \texttt{int}} \quad \frac{}{\Gamma \vdash \texttt{1} : \texttt{int}} \quad \frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 \texttt{ + } e_2 : \texttt{int}}$$

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \quad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1 e_2 : \tau} \quad \frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \texttt{fn}\,(x : \sigma)\texttt{ => } e : \sigma \to \tau}$$

Dowolne zamknięte (nie zawierające zmiennych wolnych) wyrażenie nazywamy *programem*. Program jest *poprawny pod względem typów*, jeśli można dla niego wyprowadzić typ w powyższym systemie. Np. program

$$(\texttt{fn}\ (f : \texttt{int} \to \texttt{int})\texttt{ => } f\ \texttt{1})\ (\texttt{fn}\ (x : \texttt{int})\texttt{ => } x\texttt{ + 1})$$

jest poprawny pod względem typów, zaś

$$(\texttt{fn}\ (f : \texttt{int} \to \texttt{int})\texttt{ => } f\ \texttt{1})\ \texttt{0}$$

nie. Napisz (**10**) algorytm sprawdzający poprawność programów pod względem typów. (Uwaga: podobny algorytm jest używany w kompilatorach Pascala. Tam Γ bywa nazywane *tablicą symboli*.)

Zadanie 4 (10). Algorytm przekształcania wyrażeń infiksowych do postaci postfiksowej czyta ciąg tokenów (liczb, operatorów binarnych i nawiasów) i wypisuje na wyjście liczby i operatory binarne w notacji postfiksowej. Zaprogramuj (**10**) ten algorytm w Prologu. Do reprezentowania tokenów użyj atomów: `+`, `-`, `*`, `/`, `^`, `'('`, `')'` i atomów postaci *'dddd'*, gdzie *d* są cyframi, np. `'1033'`. Wyrażenie w postaci infiksowej reprezentuj jako listę atomów, np.

```
['2', +, '(', '3', *, '5', ')']
```

reprezentuje wyrażenie $2 + (3 * 5)$. Napisz predykat `onp/2`, taki, że jeśli *t* jest listą reprezentującą wyrażenie w postaci infiksowej, zaś `X` jest zmienną, to wywołanie `onp(`*t*`,X)` powoduje podstawienie pod zmienną `X` listy atomów, która reprezentuje wyrażenie w notacji postfiksowej, np.

```
?- onp(['2', +, '(', '3', *, '5', ')'], X).
X = ['2', '3', '5', *, +]
```

Przyjmij przy tym następujące priorytety operatorów:

operator	siła wiązania	kierunek łączności
`+`, `-`	najsłabiej	w lewo
`*`, `/`	mocniej	w lewo
`^`	najmocniej	w prawo

Program nie musi wykrywać błędów składniowych we wczytywanych wyrażeniach (można założyć, że lista wejściowa zawsze reprezentuje poprawne wyrażenie infiksowe). Możesz korzystać z predykatów: `isnumber/1`, sprawdzającego, czy wszystkie znaki podanego atomu są cyframi, `append/3`, `reverse/2` i predykatów porównujących liczby (`<`, `>`, `==`).