Programowanie

Rozwiązania zadań z egzaminu poprawkowego

11 września 2001

Zadanie 1. Niech

alfabet:	$\Sigma = \{0, 1\}$
zbiór stanów:	$Q_m = \{w \in \{0,1\}^* \mid |w| = m+1\}$
stan początkowy:	$q_m = 1^{m+1}$
zbiór stanów akceptujących:	$F_m = \{0u \mid u \in \{0,1\}^*, |u| = m\}$
funkcja przejścia:	$\delta_m(aw, b) = wb$, gdzie $a, b \in \{0,1\}$ oraz $w \in \{0,1\}^*$ i $|w| = m$

Dla dowolnego $m \geq 0$ automat $\mathcal{A}_m = \langle \Sigma, Q_m, q_m, F_m, \delta_m \rangle$ akceptuje język L_m. Idea działania tego automatu jest następująca. Przypuśćmy, że automat przeczytał słowo v długości większej niż m. Wówczas po jego przeczytaniu znajduje się w stanie w, będącym sufiksem o długości $m+1$ słowa v. Stan automatu niejako „pamięta" $m+1$ ostatnio przeczytanych symboli. Istotnie, czytając kolejny znak usuwamy z etykiety stanu skrajny lewy symbol a (automat „zapomina" symbol przeczytany $m+2$ kroków wcześniej) i dopisujemy z prawej strony właśnie przeczytany symbol b. Zatem automat akceptuje słowo (tj. po jego przeczytaniu automat znajduje się w stanie zaczynającym się zerem) wtedy i tylko wtedy, gdy na $m+1$ pozycji, licząc od końca, w słowie v występowało zero, tj. gdy słowo należało do języka. Żadne ze słów długości co najwyżej m nie należy do języka. Nie są one także akceptowane przez automat. Istotnie, jeśli automat przeczyta słowo v, takie, że $|v| = k \leq m$, to znajdzie się w stanie $11^{m-k}v \notin F$.

Formalny dowód jest powtórzeniem powyższego rozumowania i opiera się na dwóch prostych lematach, które udowodnimy indukcyjnie.

Lemat 1. Dla dowolnych słów u, v, w, takich, że $|u| = |v| \leq m+1$ i $|uw| = m+1$, jest $\hat{\delta}_m(uw, v) = wv$.

Dowód indukcyjny względem długości słowa v (indukcja jest tu w skończonym zbiorze liczb nie większych niż $m+1$ uporządkowanym relacją $\leq$). Dla $|v| = 0$ z definicji $\hat{\delta}_m(q, \epsilon) = q$ dla dowolnego stanu $q \in Q_m$. Przypuśćmy, że twierdzenie jest prawdziwe, gdy $|v| = n$. Rozważmy słowo v' długości $n+1$. Niech $|u'| = |v'|$. Zatem $u' = ua$ i $v' = vb$, gdzie $|u| = |v| = n$ i $a, b \in \{0,1\}$. Mamy $\hat{\delta}_m(u'w, v') = \hat{\delta}_m((ua)w, vb) = \delta_m(\hat{\delta}_m(u(aw), v), b) = \delta_m(a(wv), b) = wvb = wv'$.

Lemat 2. Dla dowolnych słów u, v, w, takich, że $|u| = |v| = m+1$, jest $\hat{\delta}_m(u, wv) = v$.

Niech $w' = \hat{\delta}_m(u, w)$. Mamy $\hat{\delta}_m(u, wv) = \hat{\delta}_m(\hat{\delta}_m(u, w), v) = \hat{\delta}_m(w'\epsilon, v) = \epsilon v = v$ na mocy lematu 1.

Rozważmy słowo $v \in \{0,1\}^*$. Pokażemy, że należy ono do języka L_m wtedy i tylko wtedy, gdy jest akceptowane przez automat, tj. gdy $\hat{\delta}_m(q_m, v) \in F_m$. Rozpatrzymy kolejno trzy zbiory słów: 1) słowa postaci $w0v'$, gdzie $|v'| = m$, 2) słowa postaci $w1v'$, gdzie $|v'| = m$, oraz słowa nie dłuższe niż m. Wszystkie słowa postaci 1) należą do języka. Są też akceptowane przez automat, bo $\hat{\delta}_m(q_m, w0v') = 0v' \in F$ na mocy lematu 2. Żadne ze słów postaci 2) nie należy do języka L_m.

Żadne też nie jest akceptowane przez automat, ponieważ $\hat{\delta}_m(q_m, w1v') = 1v' \notin F$, również na mocy lematu 2. W końcu żadne ze słów postaci 3) nie należy do języka i jednocześnie nie jest akceptowane przez automat, bo na mocy lematu 1 jest $\hat{\delta}_m(q_m, u) = \hat{\delta}_m(1^k 11^{m-k}, u) = 11^{m-k}u \notin F$.

Automat $\mathcal{A}_m$ ma 2^{m+1} stanów. Okazuje się, że jest *minimalny* w tym sensie, że każdy automat akceptujący język L_m ma co najmniej 2^{m+1} stanów. Istotnie, przypuśćmy, że pewien automat $\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle$ akceptuje język L_m i rozważmy dwa dowolne różne słowa $u = u_0 \dots u_m$ i $v = v_0 \dots v_m$ długości $m+1$. Skoro $u \neq v$, to istnieje k, takie, że $1 \leq k \leq m+1$ i $u_k \neq v_k$. Ze względu na symetrię przyjmijmy, że $u_k = 0$ i $v_k = 1$. Rozważmy słowa $u' = u1^k$ i $v' = v1^k$. Otóż $m+1$-szym symbolem słowa u', licząc od prawej, jest $u_k = 0$, zatem $u' \in L_m$. Natomiast takim symbolem w słowie v' jest $v_k = 1$, zatem $v' \notin L_m$. Ale $F \ni \hat{\delta}(q_0, u') = \hat{\delta}(q_0, u1^k) = \hat{\delta}(\hat{\delta}(q_0, u), 1^k)$, zaś $F \not\ni \hat{\delta}(q_0, v') = \hat{\delta}(q_0, v1^k) = \hat{\delta}(\hat{\delta}(q_0, v), 1^k)$, stąd $\hat{\delta}(\hat{\delta}(q_0, u), 1^k) \neq \hat{\delta}(\hat{\delta}(q_0, v), 1^k)$. Skoro $\hat{\delta}$ jest funkcją, znaczy to, że $\hat{\delta}(q_0, u) \neq \hat{\delta}(q_0, v)$. Wynika stąd, że stany $\hat{\delta}(q_0, u)$ dla wszystkich słów u długości $m+1$ są parami różne. Automat $\mathcal{A}$ ma więc co najmniej 2^{m+1} stanów (bo tyle jest słów długości $m+1$).

Zadanie 2. Konstruktorami są stała c i symbole f i g. Zasada indukcji jest więc następująca:

$$\frac{\Phi(c) \quad \Phi(y) \Rightarrow \Phi(f(y)) \quad \Phi(y) \Rightarrow \Phi(g(y))}{\Phi(x)}$$

Przez indukcję względem x pokażę teraz, że $(x \oplus y) \oplus z = x \oplus (y \oplus z)$. Dla $x \equiv c$ mamy

$$L \equiv (c \oplus y) \oplus z \overset{(3)}{=} y \oplus z \overset{(3)}{=} c \oplus (y \oplus z) \equiv P$$

Przypuśćmy, że $(x \oplus y) \oplus z = x \oplus (y \oplus z)$. Pokażemy, że $(f(x) \oplus y) \oplus z = f(x) \oplus (y \oplus z)$. Istotnie,

$$L \equiv (f(x) \oplus y) \oplus z \overset{(1)}{=} f(x \oplus y) \oplus z \overset{(1)}{=} f((x \oplus y) \oplus z) \overset{\text{zał. ind.}}{=} f(x \oplus (y \oplus z)) \overset{(1)}{=} f(x) \oplus (y \oplus z) \equiv P$$

Przy tym samym założeniu i w taki sam sposób, korzystając jedynie z równości (2) zamiast równości (1) możemy pokazać, że $(g(x) \oplus y) \oplus z = g(x) \oplus (y \oplus z)$. Na mocy zasady indukcji twierdzenie jest udowodnione.

Zadanie 3. Algorytm sprawdzania typów zapiszemy w postaci rekurencyjnej procedury Typ(Γ, e), której parametrami są wyrażenie e, którego typ ma być sprawdzony i skończony zbiór założeń o typach wszystkich zmiennych wolnych, które mogą wystąpić w wyrażeniu e. Jeśli w wyrażeniu e nie ma błędów typowych, to wynikiem działania procedury Typ jest typ wyrażenia e. W przeciwnym razie procedura Typ przerywa pracę natychmiast po napotkaniu błędu typowego. Ponieważ typy są jawnie podawane w wyrażeniach i badamy tylko termy zamknięte, to jest zbyteczna rekonstrukcja typów i kontekst Γ zawiera od razu pełną informację o typach wszystkich zmiennych występujących w wyrażeniu.

$$
\begin{aligned}
\mathrm{Typ}(\Gamma, \texttt{0}) &= \texttt{int} \\
\mathrm{Typ}(\Gamma, \texttt{1}) &= \texttt{int} \\
\mathrm{Typ}(\Gamma, x) &= \Gamma(x) \\
\mathrm{Typ}(\Gamma, e_1 \texttt{ + } e_2) &= \text{niech } \sigma_1 = \mathrm{Typ}(\Gamma, e_1) \\
&\qquad\quad \sigma_2 = \mathrm{Typ}(\Gamma, e_2) \\
&\quad \text{jeśli } \sigma_1 = \sigma_2 = \texttt{int}, \\
&\qquad \text{to zwróć } \texttt{int}, \\
&\qquad \text{w p.p. przerwij działanie — niepoprawny typ} \\
\mathrm{Typ}(\Gamma, e_1 e_2) &= \text{niech } \rho = \mathrm{Typ}(\Gamma, e_1) \\
&\qquad\quad \sigma = \mathrm{Typ}(\Gamma, e_2) \\
&\quad \text{jeśli } \rho = \sigma \to \tau, \text{ dla pewnego } \tau, \\
&\qquad \text{to zwróć } \tau, \\
&\qquad \text{w p.p. przerwij działanie — niepoprawny typ} \\
\mathrm{Typ}(\Gamma, \texttt{fn}\,(x : \sigma)\,\texttt{=>}\,e) &= \text{niech } \tau = \mathrm{Typ}(\Gamma \cup \{x : \sigma\}, e) \\
&\quad \text{zwróć } \sigma \to \tau
\end{aligned}
$$

Ostatecznie wywołanie $\mathrm{Typ}(\emptyset, e)$ bada poprawność typów w zamkniętym wyrażeniu (programie) e.

Zadanie 4. Tabelkę priorytetów (wejściowego i stosowego) implementujemy w postaci predykatu `prio/3`:

```
prio(+,1,2).
prio(-,1,2).
prio(*,3,4).
prio(/,3,4).
prio(^,6,5).
prio(N,7,8) :- isnumber(N).
prio('(',9,0).
prio(')',0,0).
```

Algorytm przekształcania wyrażenia jest zawarty w predykacie `onp/4`, w którym kolejność argumentów jest następująca. Pierwszy argument jest wejściową listą reprezentującą wyrażenie infiksowe. Drugi argument jest listą będącą stosem roboczym algorytmu. Trzeci argument jest akumulatorem zbierającym atomy wypisywane przez algorytm. Te trzy argumenty są argumentami wejściowymi i muszą być ukonkretnione w chwili wywołania. Ostatni, czwarty, wyjściowy argument reprezentuje wynik obliczeń.

```
onp([],[],Wy,Wy).              % koniec wyrażenia, pusty stos, wynik
                               % kopiowany z akumulatora do zmiennej
                               % wyjściowej
onp([],[_|_],_,_) :-           % koniec wyrażenia, stos niepusty - błąd
   !,
   fail.
onp([X|We],[Y|St],Wy,Res) :-   % priorytet wejściowy mniejszy niż stosowy
   prio(X,N,_),                % wierzchołek stosu przesuwa się na wyjście
   prio(Y,_,M),
   N<M,
   onp([X|We],St,[Y|Wy],Res).
onp([X|We],[Y|St],Wy,Res) :-   % priorytet wejściowy większy niż stosowy
   prio(X,N,_),                % atom bieżący jest wstawiany na stos
   prio(Y,_,M),
   N>M,
```

```
   onp(We,[X,Y|St],Wy,Res).
onp([X|We],[Y|St],Wy,Res) :-  % priorytety wejściowy i stosowy są równe
   prio(X,N,_),               % jest to więc para nawiasów '(' i ')',
   prio(Y,_,M),               % które ulegają anihilacji
   N==M,
   onp(We,St,Wy,Res).
```

Przed uruchomieniem algorytmu na koniec wyrażenia wejściowego należy dopisać `'('`, zaś na stos roboczy wstawić `')'`. Po wykonaniu przekształcenia wynik zawiera atomy w odwrotnej kolejności (dopisuje się je z przodu listy), należy go więc odwrócić. Predykat `onp/2` jest więc następujący:

```
onp(Inp,Res) :-
   append(Inp,[')'],We),
   onp(We,['('],[],Wy),
   reverse(Wy,Res).
```