SpecVer & LEM'12 projects XX (2020) 1–20 1
Dombrova Research

Analiza programu PawelG

Andrzej Salwicki
Dombrova Research
salwicki .at) mimuw edu pl

13 maja 2020

Streszczenie. Badamy krótki program PawelG. Przedstawiamy dwa dowody poprawności tego programu. Program mieści sie na jednej stronie, a oba dowody wymagają znacznie więcej miejsca. Zwracamy uwagę na możliwości oferowane przez rachunek programów.
Przypatrz się, zaobserwuj jaką wiedzą dysponuje programista, jeśli jest świadom co ten program robi, nawet gdy nie przedstawia dowodu.

Wprowadzenie

Naszym celem jest zwrócenie uwagi na fakt, że w rachunku programów można wyrażać semantyczne własności programów przez pewne formuły i przeprowadzać dowody takich formuł. Wymieńmy w tym miejscu, parę najważniejszych własności semantycznych: własność stopu danego programu P, poprawność programu P względem warunku wstępnego α i warunku końcowego β, są i inne, zob. [1, 2].
Dwie są drogi prowadzenia dowodów formuł opisujących własności semantyczne programów: a) w dowodzie badamy własności obliczeń programu, stany obliczenia itp, lub b) dowód przeprowadzamy formalnie, posługując się tylko formułami, aksjomatami i regułami wnioskowania.

1. Program PawelG

Przyjrzyj się poniższemu programowi [5]. Program jest zapisany w Loglanie'82 [4], ale nietrudno przepisać go w ortografii innego języka programowania. NIE uruchamiaj go teraz!

- Nie jest oczywiste czy program ten coś robi, czy też jego obliczenie jest nieskończone.
- Czy potrafisz odgadnąć odpowiedź?
- Czy potrafisz wytłumaczyć dlaczego inna osoba ma uznać Twoją argumentację?

- Czy potrafisz napisać dowód Twojego twierdzenia bez posiłkowania się pojęciami stan obliczenia, obliczenie itp.?

```
program PawelG;  (* autor Paweł Gburzyński, 1983, [5]  *)
    var A: arrayof integer;
    var n, k, j : integer ;
    unit DrukujA: procedure;
        var j: integer
    begin
        for j:=1 to n do write( A(j)) od;
        writeln
    end DrukujA;

    unit F: procedure;
        var i: integer;
    begin
        if k=n+1 then
            call DrukujA;
        else
            for i:= 1 to n
            do
                if A[i]=0 then
                    A[i] := k; k := k+1;
                    call F;
                    k := k-1; A[i]:=0
                fi;
            od;
        fi;  return
    end F;
begin
    readln(n);
    array A dim(1:n);
    for j := 1 to n do A[j] := 0 od;
    k :=1; ;
    call F;
    writeln(”Bywaj“)
end PawelG
```

Spośród kilku pytań jakie nasuwają się po przeczytaniu tego niewielkiego programu warto rozważyć dwa:

- Czy ten program wydrukuje słowo "Bywaj"? tzn. czy obliczenia tego programu są skończone?
- Co ten program właściwie robi?

Prosimy Cię byś spróbował odpowiedzieć na te pytania NIE uruchamiając programu. Spróbuj swych sił. Możesz przepisać ten program na Twój ulubiony język (C ?), NIE uruchamiaj go. Nie testuj! Staraj się odgadnąć i uzasadnić Twą odpowiedź.

Spróbuj swych sił!

Nie ma nic złego w posłużeniu się eksperymentem, tzn. obliczeniem testowym, by sformułować twierdzenie o tym czy program zakończy obliczenie i co on wogóle robi?

Błędem natomiast jest pozostawanie w przekonaniu, że z takiego jednostkowego eksperymentu wynika prawdziwość takowego twierdzenia. Nawet jeśli próbne obliczenia wykonano setki tysięcy razy!

Oto nasza teza, poniżej przedstawiamy dwa dowody tego twierdzenia.

Twierdzenie 1. Dla każdej liczby naturalnej n program PawelG oblicza i drukuje wszystkie permutacje zbioru $\{1, \ldots, n\}$ i kończy obliczenie.

Jest oczywiste, że ze względu na rozmiar wyników nie mieszczą się one w pamięci, są sukcesywnie drukowane. Stanowi to pewien kłopot w sformułowaniu warunku poprawności programu.
Analizę programu zaczniemy od łatwej obserwacji

Fakt 1.

$$\big((output = wrd) \wedge \bigwedge_{j=1}^{n} A[j] = p_j\big) \Rightarrow \{\text{call DrukujA}\}(output = wrd \oplus "\underline{p_1}, \underline{p_2}, \ldots, \underline{p_n}")$$

Znak $\oplus$ oznacza operację konkatenacji (dopisania). Ciąg "$\underline{p_1}, \underline{p_2}, \ldots, \underline{p_n}$" jest ciągiem liczebników reprezentujących aktualne wartości liczb $p_1, p_2, \ldots, p_n$.
Czyli, każde wykonanie instrukcji call DrukujA spowoduje wydrukowanie wszystkich n liczb zapisanych w tablicy A.

Pozostaje do wykazania, że
(i) za każdym razem gdy dochodzi do wykonania instrukcji call DrukujA tablica A zawiera pewną permutację liczb $1, \ldots, n$ oraz
(ii) zostaną wydrukowane wszystkie permutacje tych liczb.

2. Dowód M – matematyczny, prawie formalny

Zadanie jakie sobie teraz stawiamy to przedstawienie dowodu, w którym nie odnosimy się do pojęć: obliczenia, stanu obliczenia etc. Dowód ma zawierać tylko takie formuły, które są albo aksjomatami algorytmicznej teorii liczb naturalnych, por. podrozdział 5.1, zobacz też [2], albo aksjomatami rachunku programów, zob. [1], albo wynikają z zastosowania pewnej reguły wnioskowania rachunku programów do pewnego zbioru formuł wcześniej udowodnionych. Dla wygody czytelnika w Dodatku (5), umieściliśmy aksjomaty i reguły wnioskowania rachunku programów oraz aksjomaty algorytmicznej teorii programów. Wielokrotnie stosujemy prawa rachunku zdań i wykorzystujemy własność ekstensjonalności. Zastępujemy pewną podformułę ψ danej formuły ϕ, formułą θ wiedząc, że równość $\theta \equiv \psi$ posiada dowód. Nie będziemy takich przypadków rozpatrywać zbyt szczegółowo by nie zmącić dowodu. Dopuszczamy zastosowanie reguły ω w ograniczony sposób: możesz mianowicie udowodnić, pewne *metatwierdzenie* stwierdzające, że dla każdego $i \in N$ formuła γ_i posiada dowód i zastosować jedną z ω-reguł rachunku programów R_3, R_6, R_7.

Taki dowód *quasi-formalny*, może więc zawierać nieskończone podzbiory formuł. Tym niemniej, dowody o jakich myślimy mogą być sprawdzane mechanicznie przez proof-checker. Więcej o tym w nieopublikowanym artykule [3].

Co mamy udowodnić? Cel nasz zostanie osiągnięty gdy potrafimy wykazac, że 1°) każde wykonanie polecenia call DrukujA spowoduje wydrukowanie pewnej permutacji liczb $1, \ldots, n$, 2°) liczba wykonanych poleceń call DrukujA jest równa $n!$ oraz 3°) wydrukowane permutacje nie powtarzają się.

Jakie narzędzia, jakie aksjomaty i reguły wnioskowania są niezbędne dla przeprowadzenia dowodu? W analizowanym programie występują zmienne typu integer i operacje na liczbach całkowitych. Czy możemy uznać, że do przeprowadzenia dowodu wystarczy algorytmiczna teoria liczb naturalnych $\mathcal{ATN}$? Zauważ. w programie PawelG występuje atomowa instrukcja procedury call F. Język teorii $\mathcal{ATN}$ nie zawiera takiej instrukcji atomowej, ani (tym bardziej) aksjomatu opisującego działanie tej instrukcji. Zgadzamy się, że deklaracja procedury F wprowadzona w programie opisuje sposób wykonania polecenia call F, determinuje jego semantykę. Ale dla nas wprowadzenie deklaracji procedury F oznacza coś więcej, mianowicie, deklaracja procedury F jest schematem nieskończonie wielu dodatkowych aksjomatów postaci

$$(\{\text{call F}\}\varphi \Leftrightarrow \{\text{Body}_F\}\varphi)$$

gdzie φ jest *dowolną* formułą. Skrót Body_F oznacza treść procedury F. Dokładniej, każda równoważność zbudowana według poniższego schematu jest dodatkowym aksjomatem opisującym procedurę F.

$$\{\textbf{call F}\}\varphi \Leftrightarrow \left\{\begin{array}{l}
(* \text{ poniżej znajduje się treść F tzn. Body_F } *)\\
\textbf{block}\\
\quad \textbf{var } i : \mathit{integer};\\
\textbf{begin}\\
\quad \text{if } k = n+1 \text{ then call } DrukujA\\
\quad \text{else}\\
\qquad \text{for } i := 1 \text{ to } n \text{ do}\\
\qquad\quad \text{if } A[i] = 0 \text{ then}\\
\qquad\qquad A[i] := k;\ k := k+1;\\
\qquad\qquad \text{call } F;\\
\qquad\qquad A[i] := 0;\ k := k-1;\\
\qquad\quad \text{fi}\\
\qquad \text{od}\\
\quad \text{fi}\\
\textbf{end block}
\end{array}\right\} \varphi$$

Co więcej, w powyższej formule, w treści procedury F można wszystkie wystąpienia identyfikatora i równocześnie zastąpić przez dowolny inny identyfikator, por. formułę (isub). Mówimy, że zmienna i zadeklarowana w bloku jest zmienną związaną.

Dowód twierdzenia 1 będziemy przeprowadzać w algorytmicznej teorii $\mathcal{T}'$, która powstaje z algorytmicznej teorii liczb naturalnych $\mathcal{ATN}$ przez dodanie do języka nowej instrukcji atomowej **call** F, i

dodanie do zbioru aksjomatów teorii $\mathcal{ATN}$ nieskończonego zbioru formuł zbudowanych według omówionego powyżej schematu.

Niech $\delta(n, k, i_1, \ldots, i_{k-1})$ bedzie oznaczeniem formuły o następującym schemacie

$$\delta(n,k,i_1,\ldots,i_{k-1}) \stackrel{df}{\equiv} \left(\begin{array}{l} (1 \leq k \leq n+1) \wedge \bigwedge\limits_{j=1}^{k-1} \big((1 \leq i_j \leq n) \wedge (A[i_j] = j)\big) \wedge \\ \left\{ \begin{array}{l} z := 0; \\ \textbf{for } j := 1 \textbf{ to } n \textbf{ do} \\ \quad \textbf{if } A[j] = 0 \textbf{ then } z := z+1 \textbf{ fi} \\ \textbf{od} \end{array} \right\} (z = n-k+1) \end{array} \right)$$

Formułę $\delta(n, k, i_1, \ldots, i_{k-1})$ czyta się w następujący sposób (jest to jej znaczenie semantyczne): spełnione są następujące warunki

(*i*) liczba naturalna n jest dodatnia,

(*ii*) wartością zmiennej A jest n-elementowy obiekt tablicowy,

(*iii*) liczba naturalna k spełnia podwójną nierówność $1 \leq k \leq n+1$ i liczby $1, \ldots, k-1$ są zapisane w dowolnym układzie na pozycjach $i_1, \ldots, i_{k-1}$ tablicy A, tj. $\forall_{j=1}^{k-1} A[i_j] = j$,

(*iv*) pozostałe miejsca tablicy A, oznaczmy je $r_k, \ldots, r_n$, są równe zero, $\forall_{j=k}^{n} A[r_j] = 0$.

Udowodnimy następujący fakt.

Lemat 2. Niech wartością zmiennej A będzie n-elementowa tablica (wektor) liczb naturalnych. Każda formuła o poniższym schemacie jest twierdzeniem teorii $\mathcal{T}'$

$$\mathcal{T}' \vdash \delta(n,k,i_1,\ldots,i_{k-1}) \Rightarrow \{\mathsf{call}\ F\}\, \delta(n,k,i_1,\ldots,i_{k-1}).$$

Dowód. Dowód lematu przebiega przez indukcję ze względu na liczbę $n+1-k$, tj. liczbę wolnych miejsc w tablicy A.

B0)(*baza*) Niech $k = n+1$, tzn. liczba wolnych miejsc jest 0.

Jeśli spełniony jest warunek $(k = n+1) \wedge \delta(n, k, i_1, \ldots, i_n)$ to tablica A zawiera pewną permutację liczb $1, \ldots .n$, ponieważ zachodzi $\bigwedge_{j=1}^{n} A[i_j] = j$.
Jest tautologią formuła

$$\vdash \delta(n, n+1, i_1, \ldots, i_n) \Rightarrow \bigwedge_{j=1}^{n} A[i_j] = j$$

Instrukcja write(x) nie zmienia wartosci żadnej zmiennej. Własnością tej instrukcji jest

$$(y = k) \equiv \{\mathsf{write(x)}\}(y = k)$$

Posługując sie tym faktem dowodzimy, że jest twierdzeniem teorii $\mathcal{T}'$ implikacja

$$\mathcal{T}' \vdash \delta(n, n+1, i_1, \ldots, i_n) \Rightarrow \{call\ DrukujA\}(\bigwedge_{j=1}^{n} A[i_j] = j)$$

Postępując podobnie udowodnimy formułę

$$\big(\delta(n, n+1, i_1, \ldots, i_n) \Rightarrow \{call\ DrukujA\}\, \delta(n, n+1, i_1, \ldots, i_n)\big)$$

Można to zapisać nieco inaczej

$$\mathcal{T}' \vdash \big(\delta(n, k, i_1, \ldots, i_n) \wedge k = n+1\big) \Rightarrow \{call\ DrukujA\}\, \delta(n, n+1, i_1, \ldots, i_n)$$

Korzystamy z reguły wnioskowania (IF+,zob. stronę 18) o instrukcji if.

$$\mathcal{T}' \vdash \big(\delta(n, k, i_1, \ldots, i_n) \wedge k = n+1\big) \Rightarrow \left\{ \begin{array}{l} \textbf{if } k = n+1 \\ \textbf{then call } DrukujA \\ \textbf{else} \\ \quad \text{(* tu wstaw cokolwiek *)} \\ \textbf{fi} \end{array} \right\} \delta(n, n+1, i_1, \ldots, i_n)$$

Wstawiamy to co potrzeba, dbając o to by zmienna $i_{\underline{n}}$ występowała tylko w tej instrukcji for.

$$\mathcal{T}' \vdash (\delta(n, k, i_1, \ldots, i_n) \wedge k = n+1) \Rightarrow \left\{ \begin{array}{l} \textbf{if } k = n+1 \\ \textbf{then call } DrukujA; \\ \textbf{else} \\ \quad \text{for } i_{\underline{n}} := 1 \text{ to } n \text{ do} \\ \quad\quad \text{if } A[i_{\underline{n}}] = 0 \text{ then} \\ \quad\quad\quad A[i_{\underline{n}}] := k;\ k := k+1; \\ \quad\quad\quad \text{call } F; \\ \quad\quad\quad A[i_{\underline{n}}] := 0;\ k := k-1; \\ \quad\quad \text{fi} \\ \quad \text{od} \\ \textbf{fi} \end{array} \right\} \delta(n, n+1, i_1, \ldots, i_n)$$

Ponieważ zmienna $i_{\underline{n}}$ występuje tylko w tych kilku liniach to możemy zastosować aksjomat instrukcji

bloku i otrzymamy

$$\mathcal{T}' \vdash \delta(n, n+1, i_1, \ldots, i_n) \Rightarrow \left\{ \begin{array}{l} \textbf{block} \\ \quad \textbf{var}\ i :\ integer \\ \textbf{begin} \\ \quad \textbf{if}\ k = n+1 \\ \quad \textbf{then call}\ DrukujA; \\ \quad \textbf{else} \\ \qquad \textbf{for}\ i := 1\ \textbf{to}\ n\ \textsf{do} \\ \qquad\ \ \textbf{if}\ A[i] = 0\ \textbf{then} \\ \qquad\quad A[i] := k;\ k := k+1; \\ \qquad\quad \textbf{call}\ F; \\ \qquad\quad A[i] := 0;\ k := k-1; \\ \qquad\ \ \textbf{fi} \\ \qquad \textbf{od} \\ \quad \textbf{fi} \\ \textbf{end block} \end{array} \right\} \delta(n, n+1, i_1, \ldots, i_n) \quad \text{(isub)}$$

Program występujący w powyższej formule (isub) to treść procedury F, możemy więc zastosować aksjomat = deklarację procedury F

$$\mathcal{T}' \vdash \delta(n, n+1, i_1, \ldots, i_n) \Rightarrow \{\textsf{call}\ F\}\ \delta(n, n+1, i_1, \ldots, i_n)$$

co kończy dowód przypadku gdy $k = n+1$.

W dalszej części dowodu wykorzystamy dwie niewielkie obserwacje. Zauważmy, że twierdzeniem rachunku programów, a więc i teorii $\mathcal{T}'$ jest poniższa formuła 1

Fakt 2.

$$\big(\delta(n, k, i_1, \ldots, i_{k-1}) \wedge A[p] = 0\big) \Rightarrow \{A[p] := k;\ k := k+1\}\ \big(\delta(n, k+1, i_1, \ldots, i_{k-1}, i_k) \wedge i_k = p\big) \tag{1}$$

Fakt ten jest łatwą konsekwencją zastosowania aksjomatu instrukcji przypisania Assign. Z założenia $\delta(n, k, i_1, \ldots, i_{k-1})$ wynika, że tablica A zawiera wszystkie liczby $1, \ldots, k-1$ czyli $\bigwedge\limits_{j=1}^{k-1} A[i_j] = j$. Ponadto miejsce $A[p]$ tablicy A jest wolne, a więc $p \neq i_j$ dla $j = 1, \ldots, k-1$.

Stąd wynika, że $\{A[p] := k\}(\bigwedge\limits_{j=1}^{k} A[i_j] = j) \wedge A[p] = k$. Z kolei, $(z = n-k+1) \Rightarrow \{k := k+1\}(n-k)$. Wynika stąd

$$\big((\bigwedge_{j=1}^{k-1} A[i_j] = j) \wedge z = n-k+1\big) \Rightarrow \{A[p] := 0;\ k := k+1\}\big((\bigwedge_{j=1}^{k} A[i_j] = j) \wedge (z = n-k)\big)$$

Podobnie, następująca formuła (2) jest twierdzeniem teorii $\mathcal{T}'$.

Fakt 3.

$$\big(\delta(n,k+1,i_1,\ldots,i_{k-1},p)\ \wedge\ A[p]=k\big)\Rightarrow\{A[p]:=0;\ k:=k-1\}\,\delta(n,k,i_1,\ldots,i_{k-1}) \qquad (2)$$

I) (*krok indukcyjny*)
(założenie) zakładamy, że dla każdego układu $p_1,\ldots,p_{k-1}$ liczb naturalnych, następująca implikacja jest twierdzeniem teorii $\mathcal{T}'$

$$\mathcal{T}'\vdash\delta(n,k,p_1,\ldots,p_{k-1})\Rightarrow\{\mathsf{call}\ F\}\,\delta(n,k,p_1,\ldots,p_{k-1}).$$

(teza) Wykażemy, że dla dowolnego układu liczb naturalnych $i_1,\ldots,i_{k-2}$ i $n-k+2$ miejsc zerowych w tablicy A można udowodnić, że

$$\mathcal{T}'\vdash\delta(n,k-1,i_1,\ldots,i_{k-2})\Rightarrow\{\mathsf{call}\ F\}\,\delta(n,k-1,i_1,\ldots,i_{k-2}).$$

Oznaczmy miejsca zerowe w tablicy A przez $r_{k-1},\ldots,r_n$ czyli dla $j=k-1,\ldots,n$ zachodzi $A[r_j]=0$. Rozpatrzymy po kolei te miejsca zerowe. Zauważmy, że dla $j=k-1,\ldots,n$ mozna udowodnić implikacje

$$A[r_j]=0\wedge\delta(n,k-1,i_1,\ldots,i_{k-2})\Rightarrow\left\{\begin{array}{l}A[r_j]:=k;\ k:=k+1;\\ call\ F;\\ A[r_j]:=0;k:=k-1\end{array}\right\}\delta(n,k-1,i_1,\ldots,i_{k-2})$$

Wynika to z Faktów (2) i (3). Dla każdego $j=k-1,\ldots,n$ warunek $A[r_j]=0\wedge\delta(n,k-1,i_1,\ldots,i_{k-2})$ pociąga za sobą $\{A[r_j]:=k;\ k:=k+1\}\delta(n,k,i_1,\ldots,i_{k-2},r_j)$ (Fakt(2)). Z założenia indukcyjnego

$$\delta(n,k,i_1,\ldots,i_{k-1},r_j)\Rightarrow\{\mathsf{call}\ F\}\,\delta(n,k,i_1,\ldots,i_{k-2},r_j)$$

Teraz wykorzystamy Fakt (3)

$$\delta(n,k,i_1,\ldots,i_{k-2},r_j)\Rightarrow\{A[r_j]:=0;\ k:=k-1\}\delta(n,k-1,i_1,\ldots,i_{k-2}).$$

Z kolei dla $A[i]\neq 0$ zachodzi w oczywisty sposób

$$A[i]\neq 0\wedge\delta(n,k-1,i_1,\ldots,i_{k-2})\Rightarrow\delta(n,k-1,i_1,\ldots,i_{k-2}).$$

Przyjmijmy następujące oznaczenia

$$\theta_i(n,k-1,i_1,\ldots,i_{k-2})\stackrel{df}{\equiv}\left\{\begin{array}{ll}A[i]\neq 0\wedge\delta(n,k-1,i_1,\ldots,i_{k-2}) & \text{gdy}A[i]\neq 0\\ A[i]=0\wedge\left\{\begin{array}{l}A[i]:=k;\\ k:=k+1;\\ call\ F;\\ A[i]:=0;\\ k:=k-1\end{array}\right\}\delta(n,k-1,i_1,\ldots,i_{k-2}) & \text{gdy}A[i]=0\end{array}\right.$$

W ten sposób, wykorzystując aksjomat instrukcji warunkowej if, wykazaliśmy, że dla każdego $i = 1, \ldots, n$ twierdzeniem teorii $\mathcal{T}'$ jest

$$\mathcal{T}' \vdash \delta(n, k-1, i_1, \ldots, i_{k-2}) \Rightarrow \left\{ \begin{array}{l} \textbf{if } A[i] = 0 \textbf{ then} \\ \quad A[i] := k; \\ \quad k := k+1; \\ \quad \mathit{call}\ F; \\ \quad A[i] := 0; \\ \quad k := k-1 \\ \textbf{fi} \end{array} \right\} \delta(n, k-1, i_1, \ldots, i_{k-2})$$

Zastosujemy następującą regułę wnioskowania pozwalającą na wprowadzenie kwantyfikatora ogólnego ograniczonego po instrukcji for

$$\frac{\bigwedge_{i=1}^{n} \{P(i)\}\, \vartheta(i),\ \bigwedge_{1 \le i < j \le n} \big(\{P(i);\, P(j)\}\vartheta(i) \equiv \{P(i)\}\, \vartheta(i)\big)}{\left\{ \begin{array}{l} \textbf{for } i \leftarrow 1 \textbf{ to } n \\ \textbf{do} \\ \quad P(i) \\ \textbf{od} \end{array} \right\} \bigwedge_{i=1}^{n} \vartheta(i)} \qquad \text{(wprkwog)}$$

by uzyskać

$$\mathcal{T}' \vdash \delta(n, k-1, i_1, \ldots, i_{k-2}) \Rightarrow \left\{ \begin{array}{l} \textbf{for } i := 1 \textbf{ to } n \textbf{ do} \\ \quad \textbf{if } A[i] = 0 \textbf{ then} \\ \quad\quad A[i] := k; \\ \quad\quad k := k+1; \\ \quad\quad \mathit{call}\ F; \\ \quad\quad A[i] := 0; \\ \quad\quad k := k-1 \\ \quad \textbf{fi} \\ \textbf{od} \end{array} \right\} \delta(n, k-1, i_1, \ldots, i_{k-2})$$

Wiemy, że $k-1 \neq n+1$ jeszcze raz zastosujemy aksjomat instrukcji if

$$\mathcal{T}' \vdash \delta(n, k{-}1, i_1, \ldots, i_{k-2}) \Rightarrow \left\{ \begin{array}{l} \textbf{if } k = n+1 \textbf{ then } call\ DrukujA \textbf{ else} \\ \textbf{for } i := 1 \textbf{ to } n \textbf{ do} \\ \quad \textbf{if } A[i] = 0 \textbf{ then} \\ \quad\quad A[i] := k; \\ \quad\quad k := k+1; \\ \quad\quad call\ F; \\ \quad\quad A[i] := 0; \\ \quad\quad k := k-1 \\ \quad \textbf{fi} \\ \textbf{od} \\ \textbf{fi} \end{array} \right\} \delta(n, k{-}1, i_1, \ldots, i_{k-2})$$

Zadbaliśmy o to by zmienna i różniła się od wszystkich innych, możemy więc zastosowac aksjomat instrukcji bloku

$$\mathcal{T}' \vdash \delta(n, k{-}1, i_1, \ldots, i_{k-2}) \Rightarrow \left\{ \begin{array}{l} \textbf{block} \\ \quad \textbf{var } i : integer; \\ \textbf{begin} \\ \textbf{if } k = n+1 \textbf{ then } call\ DrukujA \textbf{ else} \\ \textbf{for } i := 1 \textbf{ to } n \textbf{ do} \\ \quad \textbf{if } A[i] = 0 \textbf{ then} \\ \quad\quad A[i] := k; \\ \quad\quad k := k+1; \\ \quad\quad call\ F; \\ \quad\quad A[i] := 0; \\ \quad\quad k := k-1 \\ \quad \textbf{fi} \\ \textbf{od} \\ \textbf{fi} \\ \textbf{end block} \end{array} \right\} \delta(n, k{-}1, i_1, \ldots, i_{k-2})$$

Teraz skorzystamy z deklaracji=aksjomatu instrukcji call F i uzyskamy

$$\mathcal{T}' \vdash \delta(n, k-1, i_1, \ldots, i_{k-2}) \Rightarrow \{\mathsf{call}\ F\}\, \delta(n, k-1, i_1, \ldots, i_{k-2}).$$

co kończy dowód lematu 2. □

W ten sposób udowodniliśmy, że dla każdej liczby naturalnej, dodatniej $n > 0$ program PawelG zakończy obliczenie.

Pozostaje do wykazania, że program ten drukuje wszystkie permutacje liczb $1, \ldots, n$.

Zauważyliśmy wcześniej, że każde wykonanie polecenia call DrukujA drukuje pewna permutację liczb $1, \ldots, n$, zob. str. 5.
Wykażemy, że polecenie call DrukujA jest wykonane $n!$ razy.
Ułatwimy sobie zadanie, wprowadzając do programu trzy niewielkie zmiany: 1°) dodajemy deklarację zmiennej licz obok deklaracji zmiennych n,k,j, 2°) obok polecenia call DrukujA dopisujemy ; licz:=licz+1, 3°) w programie głównym, przed instrukcją k:=1 wstawiamy instrukcję licz:=0;.
Definiujemy warunek początkowy

$$\alpha(n, k, i_1, \ldots, i_{k-1}, w) \stackrel{df}{\equiv} \big(\delta(n, k, i_1, \ldots, i_{k-1}) \wedge licz = w\big)$$

i warunek końcowy

$$\beta(n, k, i_1, \ldots, i_{k-1}, w) \stackrel{df}{\equiv} \big(\delta(n, k, i_1, \ldots, i_{k-1}) \wedge licz = (n-k+1)! + w\big)$$

Nietrudno teraz udowodnić odmieniony wariant lematu 2.

Lemat 3.

$$\mathcal{T}' \vdash \alpha(n, k, i_1, \ldots, i_{k-1}, w) \Rightarrow \{\text{call } F\}\, \beta(n, k, i_1, \ldots, i_{k-1}, w).$$

Dowód tego lematu naśladuje dowód lematu 2.
Gdy $k = n$ to wykonaniu polecenia call DrukujA towarzyszy instrukcja licz:=licz+1.
Jeśli $k < n$ i teza jest udowodniona dla $k + 1$ to instrukcja for jest równoważna $(n - k + 1)$- krotnemu wykonaniu polecenia złozonego {A[i]:=k; k:=k+1; call F; A[i]:=0; k:=k-1 }, zmienna i przyjmuje wartości $r_k, \ldots, r_n$ o których mówiliśmy wcześniej.
Z założenia indukcyjnego, każde takie polecenia powoduje zwiększenie licznika licz o $(n - k)!$. Razem licznik licz zostaje zwiększony o $(n - k + 1)!$.

Koniec dowodu lematu 3

Pozostaje upewnić się, że żadna permutacja nie została wydrukowana dwukrotnie.
Rzeczywiście, potrafimy wykazać, że jeśli przyjąć zmieniony warunek końcowy

$$\beta'(n, k, i_1, \ldots, i_{k-1}, w) \stackrel{df}{\equiv} \left(\begin{array}{l} \delta(n, k, i_1, \ldots, i_{k-1}) \wedge licz = (n-k+1)! + w \wedge \\ \text{żadna z tych permutacji nie powtarza się} \end{array} \right)$$

Nietrudno teraz udowodnić odmieniony wariant lematu 2.

Lemat 4.

$$\mathcal{T}' \vdash \alpha(n, k, i_1, \ldots, i_{k-1}, w) \Rightarrow \{\text{call } F\}\, \beta'(n, k, i_1, \ldots, i_{k-1}, w).$$

Nie będziemy formalizować tego dowodu. Możesz spróbować zrobić to samodzielnie. Zacznij od napisania formuły wyrażającej własnośc dwie permutacje są rózne. Nasz argument jest prosty. Podczas wykonywania instrukcji for powtarzamy $(n - k + 1)$ razy czynność następującą: dla $j = k, \ldots, n$, zapisz liczbę k na miejscu $A[r_j]$ i na pozostałych $(n - k)$ wolnych miejscach wytwórz $(n - k)!$ **różnych**, permutacji. Widzimy, że utworzone w ten sposób $(n - k + 1)!$ permutacje są różne, nie ma powtórzeń.

Koniec dowodu lematu 4 i koniec dowodu twierdzenia 1.

3. Dowód G – semantyczny, graficzny – "*jak to działa*"?

W tym rozdziale przedstawiamy argumenty na rzecz twierdzenia 1, korzystając z wiedzy o tym jak wykonywany jest program. Analiza obliczeń programu PawelG musi być poprzedzona zrozumieniem jak przebiega obliczenie, co składa się na stan obliczenia (słowa *konfiguracja* i *stan obliczenia* będą używane zamiennie). Nasze argumenty nie opierają się na aksjomatach, rozumowanie wykorzystuje intuicje wspólne programistom z pewnym doświadczeniem. By uzyskać pewność, że autor i czytelnik podzielają wspólne intuicje, przytaczamy kilka definicji. Intuicje z nimi związane są podstawą naszych argumentów w tym rozdziale.

W tekście programu wyodrębniamy dwa moduły: program i zawarty w nim moduł (tj. unit) deklaracji procedury F. Podczas wykonywania programu utworzony zostanie rekord $Main$ aktywacji programu i pewna liczba rekordów aktywacji procedury F.

Definicja 1. Rekord aktywacji jest układem czterech elementów:

1. stanu lokalnej pamięci rekordu
2. listy instrukcji ,
3. wskaźnika SL wskazującego rekord aktywacji w którym należy szukać zmiennych nielokalnych,
4. wskaźnika DL wskazującego do którego rekordu należy przejść w trakcie realizacji polecenia return,

Każdy rekord aktywacji zajmuje osobną, skończoną porcję pamięci. (Na rysunkach każdy rekord aktywacji zajmuje oddzielny fragment rysunku obwiedziony ramką.)

> Czytelnik zauważy, że rekord $Main$ nie ma wskaźników SL ani DL. Zakończenie wykonywania instrukcji w rekordzie $Main$ kończy wykonywanie programu – stąd nie ma potrzeby tworzenia wskaźnika DL. Natomiast wystąpienie nazwy nielokalnej w rekordzie $Main$ oznacza, że tekst programu jest niepoprawny, taki błąd zostanie zasygnalizowany przez kompilator.

Definicja 2. Konfiguracja (tj. stan) obliczenia programu PawelG jest zbiore rekordów aktywacji procedury F i rekordu *Main* programu głównego.

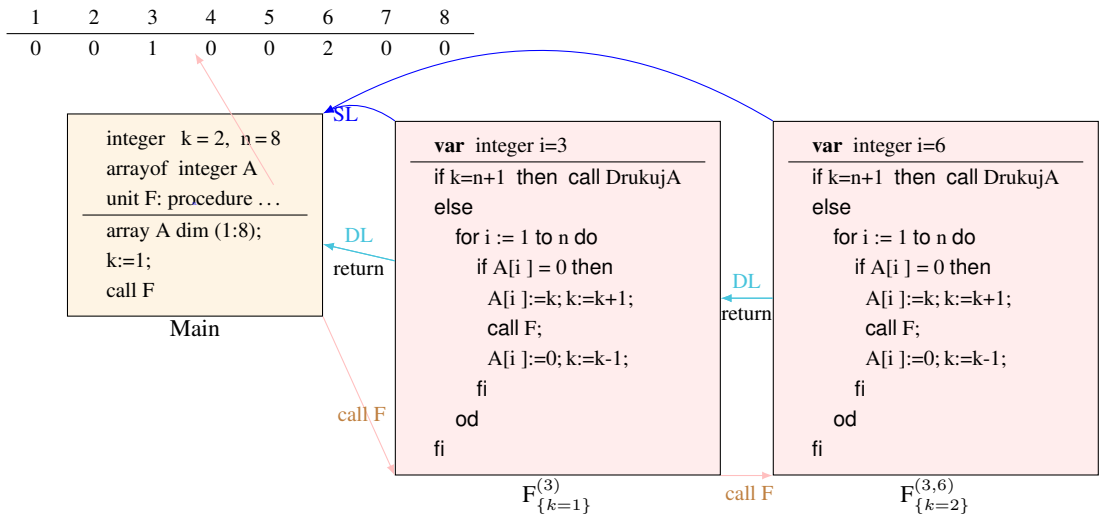
N.B. Niektórzy mówią: instancja procedury zamiast rekord aktywacji.

Definicja 3. *Protokół* call wykonania instrukcji procedury call F – realizacja instrukcji procedury polega na utworzeniu rekordu aktywacji procedury F:

1. przydzielenie porcji pamięci komputera wystarczającej dla:
2. zapisania wartościowania zadeklarowanych w module zmiennych,
3. zapisania wskaźników SL i DL (w przypadku rekordów aktywacji procedury F),
4. rozpoczęcia wykonywania treści procedury F,

5. po wykonaniu wszystkich poleceń tej instancji procedury F procesor (tj. wirtualny komputer) wznawia obliczenie w rekordzie aktywacji wskazanym przez wskaźnik DL – tj. wykonuje instrukcje return. Komputer usunie niepotrzebny już zakończony rekord aktywacji.

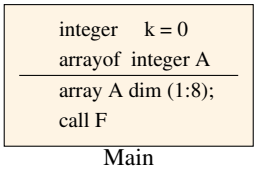
To co widać na rysunku 1 jest *łańcuchem dynamicznym* rekordów aktywacji. Instrukcje call F powodują tworzenie kolejnych, nowych rekordów aktywacji i dołączanie ich do łańcucha wskaźnikiem DL. Zakończenie rekordu procedury instrukcją return, powoduje przejście do poprzedniego rekordu wzdłuż strzałki DL i usunięcie rekordu zakończonego, już niepotrzebnego.



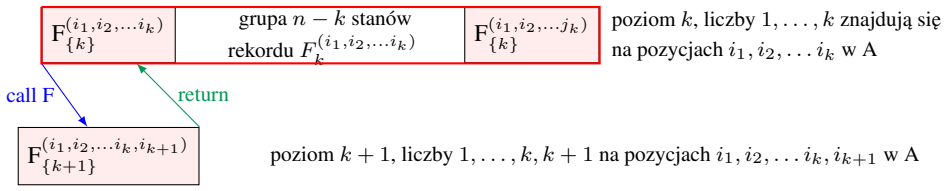
Rysunek 1. Łańcuch dynamiczny rekordów aktywacji.
Procesor wykonuje instrukcje rekordu aktywacji procedury F leżącego na prawo. Dostępne są lokalna zmienna i oraz zmienne n, k, j z rekordu aktywacji programu głównego, a także tablica A, - zob. SL. W aktywnym rekordzie procedury F wykonywana jest siódma iteracja pętli for. W środkowym, nieaktywnym w tej chwili, rekordzie F wykonano 3 powtórzenia pętli for.

Na rysunku 1 przyjęto następującą umowę:

Definicja 4. *Drzewo możliwych aktywacji* procedury F (tj. historii obliczenia).
(*i*) *Korzeniem* drzewa jest rekord aktywacji programu głównego Pawel, por. rysunek 2.
(*ii*) Każdy rekord aktywacji procedury F jest na poziomie wyznaczonym przez wartość zmiennej globalnej k i ma $n - k + 1$ stanów. Wynika to z analizy instrukcji for i ....
Stany te zgrupowane razem to węzeł drzewa H. Stan rekordu aktywacji na poziomie k oznaczamy $F_k^{(i_1,i_2,\ldots i_{k-1})}$ – wskaźnik u dołu to poziom węzła, wskaźnik u góry to informacja, że w tablicy A na miejscu i_j zapisana jest liczba j, dla $j = 1, \ldots, k-1$.
(*iii*) Węzeł na poziomie n ma jednego syna, jest nim rekord aktywacji procedury DrukujA.



Rysunek 2. Korzeń drzewa aktywacji



Rysunek 3. relacja ojciec – syn

Dla każdego węzła w drzewa H aktywacji procedury F, każde wykonanie instrukcji procedury call F w tym rekordzie aktywacji spowoduje utworzenie nowego rekordu aktywacji procedury w' – syna węzła w i rozpoczęcie wykonywania treści procedury F w otoczeniu rekordu w'. W żargonie mówi się o "przejściu"do nowego rekordu aktywacji. Zobacz rysunek 3.

Dla ustalonego n drzewo aktywacji H ma więc taki kształt jak na rysunku 4.

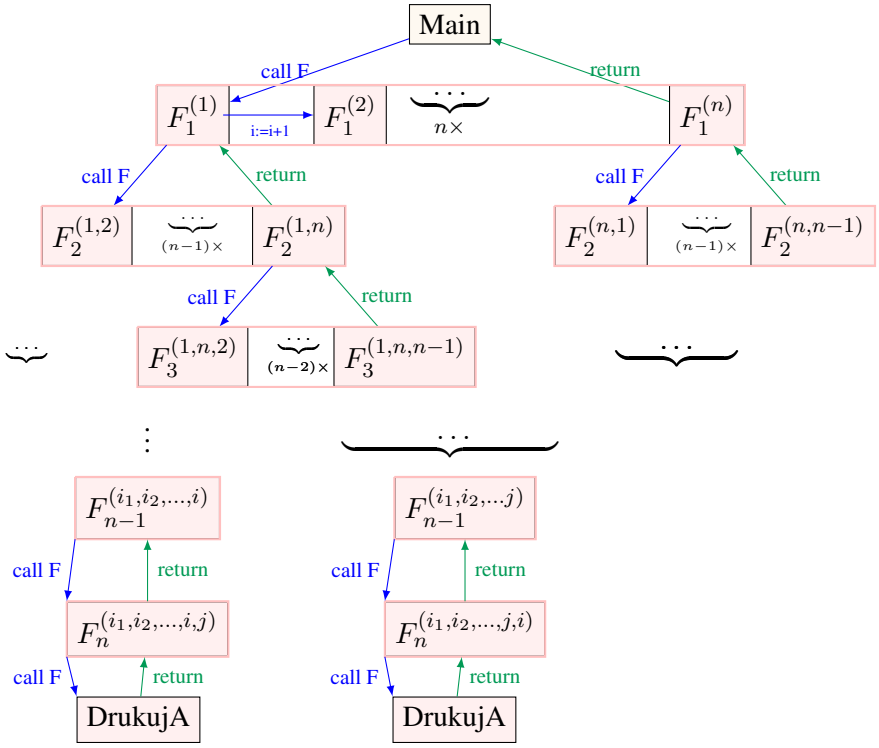
Dowód twierdzenia 1 wynika wprost ze zgromadzonych poniżej obserwacji.

Fakt 4. Wykonanie programu Pawel jest równoznaczne z utworzeniem i odwiedzeniem wszystkich węzłów drzewa H (przeszukiwanie w głąb DFS).

Fakt 5. W każdym wierzchołku wykonywana jest liczba instrukcji ograniczona przez $n * 6$. Na każdym poziomie $k, k \leq n$ węzeł ma $n - k + 1$ stanów i tyleż synów. Przejście od jednego stanu do następnego wymaga odtworzenia stanu tablicy A i zmiennej k a następnie wyszukania następnego pustego miejsca w tablicy A, zapisania w nim wartości zmiennej k i zwiększenia k. Na rysunkach 3 i 4 węzeł jest obwiedziony czerwoną linią grupującą w ten sposób stany tego węzła.

Fakt 6. Liczba liści drzewa aktywacji jest równa $n!$.

Stan tablicy A podczas wykonywania tej procedury z wartością zmiennej $k = n + 1$ jest taki, że wypełnione jest każde miejsce w tej tablicy. Wartościami zapisanymi w tej tablicy są liczby $1, \ldots, n$ i żadna liczba się nie powtarza. Podsumowując, wydrukowane zostaną wszystkie permutacje liczb $1, \ldots, n$.



Rysunek 4. Drzewo aktywacji programu Pawel (*szkic*)

4. Wnioski

Który dowód jest lepszy? G czy M? Naszym zdaniem nie warto tego roztrząsać.
Dowód G wydaje się bardziej przyjazny człowiekowi. Dowód M może odstraszać. Jest dłuższy (celowo uwypukliliśmy prawie wszystkie detale) i odwołuje się do reguł i aksjomatów mało znanego rachunku programów tj. logiki algorytmicznej.
Najważniejsze jest stwierdzenie, że do osiągnięcia celu jakim jest dowód twierdzenia prowadzą *dwie*, różne drogi. To jest wniosek z naszego ćwiczenia.
Ta prawidłowośc jest ogólna: Dla każdego programu P,

> Niech pewna własność semantyczna w programu P będzię wyrażalna formułą (algorytmiczną) ψ.
>
> Wtedy nastepujące zdania (i),(ii) są równoważne:
>
> (i) W algorytmicznej teorii $\mathcal{T}$ istnieje dowód formuły ψ,
>
> (ii) Formuła ψ jest prawdziwa w każdym modelu teorii $\mathcal{T}$.

O tym właśnie mówi twierdzenie o *pełności* rachunku programów tj. logiki algorytmicznej, por. [1].
Ważne. Z udowodnionego twierdzenia 1 wynika, że przy zachowaniu założenia o tablicy A instrukcja procedury wykona się w skończonym czasie. Samo przyjęcie deklaracji procedury *nie wystarcza*. Można napisać deklarację procedury i przyjąć odpowiedni zbiór aksjomatów, ale brak dowodu własności stopu *dyskwalifikowałby* naszą pracę.

Zanotuj w pamięci! Jeśli w programie wprowadzasz deklarację procedury (odpowiednio, deklarację funkcji), to dzieją się dwie rzeczy:

(*i*) Dodajesz do języka programowania nową instrukcję atomową, czyli wzbogacasz język. W naszym przypadku był to identyfikator F.

(*ii*) przyjmujesz nowe aksjomaty o tej nowej instrukcji. Zabieg ten nie jest obojętny dla zdrowia teorii. Powinieneś przeprowadzić dowód twierdzeń o istnieniu i o jednoznaczności działania $F_{\mathfrak{A}}$.

W wielu przypadkach dowód taki jest banalny. Sprawa może się skomplikować gdy wprowadzana definicja jest niejawna lub gdy w deklarcji występuje konstrukcja while.
programu PawelG, to i komputer będzie musiał Cię posłuchać i nie tylko zakończy obliczenia (tj. nie zapętli się), ale i zwróci poprawny wynik obliczenia. Uwaga ta odnosi się do każdego programu P i wszelkich poprawnych dowodów semantycznych własności programu, takich jak np. stop, poprawność i in.

4.1. Co dalej?

- Warto pomyśleć o zbieraniu wiedzy o istniejących programach i modułach programów. Tworzone są (w nadmiarze) biblioteki klas i procedur.
  - Nie istnieją biblioteki specyfikacji klas.
  - Nie istnieją biblioteki twierdzeń o procedurach i klasach.
- Warto pomyśleć o mało znanych zawodach:
  - audytor oprogramowania,
  - architekt (projektant) specyfikacji oprogramowania
- Czy nie warto wprowadzić rachunek programów do curriculum studiów informatycznych?

  W dalszej perspektywie, wydaje się, że uczniowie i studenci powinni nabywać umiejętność budowania dowodów poprawności programów. Powinni umieć wyrazić odpowiednią formułą badaną własność semantyczną badanego programu P i przeprowadzić jej dowód w odpowiednio dobranej teorii $\mathcal{T}$.
- Warto stworzyć nowy proof-checker – odpowiedni dla rachunku programów. Jest to zadanie całkowicie wykonalne (w odróżnieniu od zadania stworzenia provera).

Pomysły na ćwiczenia:

- Co się stanie jeśli w deklaracji procedury F (w jej nagłówku) słowo **procedure** zastapimy przez **class**, a polecenie **call** F zastąpimy przez **new** F?
  - – wprowadź kill, co się zmieni?
- Co się stanie gdy zamiast procedure napiszemy coroutine i odpowiednio zmienimy kilka miejsc w programie?
- Czy ma sens rozważanie procesów zamiast procedury? To może być interesujące.

4.2. porównanie

Poniżej zestawiliśmy trzy wersje programu obok siebie

```
program PawelGCor;
  var A: arrayof integer;
  var CF: arrayof F, n, k, j : integer ;

  unit DrukujA: procedure;
    var j: integer
  begin
    for j:=1 to n do write( A(j)) od;
    writeln
  end DrukujA;

  unit F: coroutine;
    var i: integer;
  begin
    return;
    do
      for i:= 1 to n
      do
        if A[i]=0 then
          A[i] := k; k := k+1;
          if k<n+1 then
            attach(CF(k))
          else call DrukujA fi;
          k := k-1; A[i]:=0
        fi;
      od;
      detach
  od
  end F;

begin
  readln(n);
  array A dim(1:n);
  for j := 1 to n do A[j] := 0 od;
  array CF dim(1:n);
  for j := 1 to n do CF[j] := new F od;
  k :=1; ;
  attach(CF(k));
  writeln("Bywaj")
end PawelGCor
```

```
program PawelG;
  var A: arrayof integer;
  var n, k, j : integer ;

  unit DrukujA: procedure;
    var j: integer
  begin
    for j:=1 to n do write( A(j)) od;
    writeln
  end DrukujA;

  unit F: procedure;
    var i: integer;
  begin


    for i:= 1  to n
    do
      if A[i]=0 then
        A[i] := k; k := k+1;
        if k<n+1 then
          call F
        else call DrukujA fi;
        k := k-1; A[i]:=0
      fi;
    od;
    return

  end F;

begin
  readln(n);
  array A dim(1:n);
  for j := 1 to n do A[j] := 0 od;


  k :=1; ;
  call F;
  writeln("Bywaj")
end PawelG
```

```
program PawelGCl;
  var A: arrayof integer;
  var n, k, j : integer ;

  unit DrukujA: procedure;
    var j: integer
  begin
    for j:=1 to n do  write(A(j))  od;
    writeln
  end DrukujA;

  unit F: class;
    var i: integer;
  begin


    for i:= 1 to n
    do
      if A[i]=0 then
        A[i] := k; k := k+1;
        if k<n+1 then
          new F;
        else call DrukujA fi;
        k := k-1; A[i]:=0
      fi;
    od;
    return

  end F;

begin
  readln(n);
  array A dim(1:n);
  for j := 1 to n do A[j] := 0 od;


  k :=1; ;
  new F;
  writeln("Bywaj")
end PawelGCl
```

5. Narzędzia pomocne w konstrukcji i weryfikacji dowodów

Zachęcamy do samodzielnej pracy i w tym celu przytaczamy niektóre narzędzia. Mogą Ci się one przydać. Dla wygody czytelnika, poniżej wyliczamy niektóre aksjomaty i reguły wnioskowania rachunku programów.

Zaczynamy od przypomnienia kilku informacji o rachunku programów.
Każdy język $\mathcal{L}$ rachunku programów jest splotem języka odpowiedniej teorii pierwszego rzędu (tu będzie to język arytmetyki następnika) oraz języka programowania o tej samej sygnaturze (tu język programów while z deklaracjami procedur). Zbiór formuł (programista pewnie woli powiedzieć wyrażeń logicznych) zawiera oprócz formuł pierwszego rzędu także formuły algorytmiczne.
Niech napis $\{K\}$ będzie programem (instrukcją), a napis α formułą, wtedy napis $\{K\}\alpha$ jest formułą algorytmiczną. Zbiór formuł algorytmicznych jest zamknięty ze względu na operacje koniunkcji, alternatywy, negacji, implikacji. Dopuszczane są także działania nieskończone kwantyfikatory zwyczajne oraz kwantyfikatory iteracji.

Aksjomat instrukcji przypisania

$$\boxed{\big(\{x := \tau\}\varphi \Leftrightarrow \varphi(x/\tau)\big)} \tag{Assign}$$

Przykład. $\{v := a^2 - b^2\}(\frac{v}{a+b}) \equiv (\frac{a^2-b^2}{a+b})$

Aksjomat instrukcji złożonej

$$\boxed{\{K;\, M\}\,\varphi \Leftrightarrow \big(\{K\}(\{M\}\varphi)\big)} \tag{Compose}$$

Aksjomat instrukcji warunkowej **if**

$$\boxed{\{\mathbf{if}\ \gamma\ \mathbf{then}\ K\ \mathbf{else}\ M\ \mathbf{fi}\}\varphi \Leftrightarrow \big((\gamma \wedge \{K\}\varphi) \vee (\neg\gamma \wedge \{M\}\varphi)\big)} \tag{AxIF}$$

Pomocnicze reguły wnioskowania dla instrukcji **if**

$$\boxed{\frac{\gamma,\ \{K\}\varphi}{\{\mathbf{if}\ \gamma\ \mathbf{then}\ K\ \mathbf{else}\ M\ \mathbf{fi}\}\varphi}} \tag{IF+}$$

i

$$\boxed{\frac{\neg\gamma,\ \{M\}\varphi}{\{\mathbf{if}\ \gamma\ \mathbf{then}\ K\ \mathbf{else}\ M\ \mathbf{fi}\}\varphi}} \tag{IF–}$$

Znaczenie instrukcji **for** opisują trzy schematy aksjomatów:

$$\boxed{(B < A) \Rightarrow \big(\{\mathbf{for}\ i := A\,\mathbf{to}\,B\,\mathbf{do}\,I\,\mathbf{od}\}\varphi \Leftrightarrow \varphi\big)} \tag{F1}$$

$$\boxed{(B = A) \Rightarrow \big(\{\mathbf{for}\,i := A\ \mathbf{to}\,B\,\mathbf{do}\,I\,\mathbf{od}\}\varphi \Leftrightarrow (\{i := A;\, I\}\varphi)\big)} \tag{F2}$$

$$\boxed{\Big(\{\mathbf{for}\,i := A\,\mathbf{to}\,B+1\,\mathbf{do}\,I\,\mathbf{od}\}\varphi \Leftrightarrow \{\mathbf{for}\,i := A\,\mathbf{to}\,B\,\mathbf{do}\,I\,\mathbf{od};\, i := i+1;\, I\}\varphi\Big)} \tag{F3}$$

Reguły wnioskowania

rachunek zdań

R_1 $\quad \dfrac{\alpha, (\alpha \Rightarrow \beta)}{\beta}$ (reguła znana jako modus ponens)

rachunek predykatów

R_6 $\quad \dfrac{(\alpha(x) \ \Rightarrow \ \beta)}{((\exists x)\alpha(x) \ \Rightarrow \ \beta)}$

R_7 $\quad \dfrac{(\beta \ \Rightarrow \ \alpha(x))}{(\beta \Rightarrow (\forall)\alpha(x))}$

rachunek programów – AL

R_2 $\quad \dfrac{\alpha \Rightarrow \beta}{\{K\}\alpha \Rightarrow \{K\}\beta}$

R_3 $\quad \dfrac{\left\{\{\textbf{if } \gamma \textbf{ then } K \textbf{ fi}^i\}(\neg\gamma \wedge \alpha) \Rightarrow \beta\right\}_{i\in N}}{\left(\{\textbf{while } \gamma \textbf{ do } K \textbf{ od}\}\ \alpha \Rightarrow \beta\right)}$

R_4 $\quad \dfrac{\{(\{K^i\}\alpha \Rightarrow \beta)\}_{i\in N}}{(\bigcup\{K\}\alpha \Rightarrow \beta)}$

R_5 $\quad \dfrac{\{(\alpha \Rightarrow \{K^i\}\beta)\}_{i\in N}}{(\alpha \Rightarrow \bigcap\{K\}\beta)}$

5.1. Aksjomaty algorytmicznej teorii liczb naturalnych $\mathcal{ATN}$

$$\forall_x\, x+1 \neq 0 \tag{I}$$

$$\forall_x \forall_y\, x+1 = y+1 \Rightarrow x = y \tag{M}$$

$$\forall_x\, \{y := 0; \textbf{while } y \neq x \textbf{ do } y := y+1 \textbf{ od}\}(x = y) \tag{S}$$

$$x - 1 \stackrel{df}{=} \{\ w := 0;\ \textbf{if } x \neq 0 \textbf{ then } \textbf{while } w+1 \neq x \textbf{ do } w := w+1 \textbf{ od fi}\ \}w \tag{P}$$

Literatura

[1] Grażyna Mirkowska and Andrzej Salwicki. *Algorithmic Logic*. "http://lem12.uksw.edu.pl/wiki/Algorithmic_Logic", 1987. "[Online; accessed 7-August-2017]".

[2] Grażyna Mirkowska and Andrzej Salwicki. *Logika algorytmiczna dla programistów*. WNT, Warszawa, 1992.

[3] Andrzej Salwicki. *A new proof of Euclid's algorithm*. "http://lem12.uksw.edu.pl/wiki/OnEuclid'salgorithm".

[4] Andrzej Salwicki and Andrzej Zadrożny. *Loglan'82 - website*. "http://lem12.uksw.edu.pl/wiki/Loglan'82_project", 2013. "[Online; accessed 27-July-2017]".

[5] Antoni Kreczmar and Paweł Gburzyński et.al. *Loglan Summer School*. Warszawa, 1983. Warsaw University, Institute of Informatics.