O luce pomiędzy algorytmiką i praktyką programowania

Analiza programu PawelG

Andrzej Salwicki

Instytut Informatyki

26 maja 2020

Streszczenie

- Luka
  Czy nie jest zastanawiające, że ludzie powierzają swój los i pieniądze programom o których nic nie wiedzą?
- A jednak można przeprowadzać dowody poprawności programów. Istnieją narzędzia.
- Wnioski i sugestie płynące z doświadczenia z programem PawelG.

Wprowadzenie

Luka

Zastanawiający jest fakt, że ludzkość wytworzyła miliony programów i powierzyła im swój los, zdrowie i pieniądze bez śladu troski o jakość tego oprogramowania. Wciąż jednak twórcy oprogramowania nie dają gwarancji jakości swych produktów w skali podobnej do tej oferowanej przez budowniczych, wytwórców aut itp. Testowanie oprogramowania wydaje się jedynym sposobem na zapewnienie, że produkt programistyczny jest właściwej jakości. Ale co to znaczy? 1) Czy wytwórca oprogramowania wie jakie własności ma mieć jego produkt? Czy kupujący oprogramowanie wie czego potrzebuje i co uzyskuje za swoje pienądze? 2) Czy ktoś skontrolował produkt programistyczny i stwierdził zgodność z zamówieniem?

Pomiędzy algorytmiką a praktyką programowania jest luka. Mamy do dyspozycji wspaniałe teksty Knutha, Aho, Hpcroft i Ullman, Cormen, Leiserson,Rivest,...

W tych książkach znajdziesz mnóstwo przydatnych i ważnych informacji. Nie znajdziesz tam programów. ... Inne książki np. Sedgewick, Drozdek, przytaczają programy, jednak bez argumentacji na rzecz poprawności tych programów.

Wstęp

Wiele lat temu dr Paweł Gburzyński wymyślił takie zadanie. Niewielu profesorów UW potrafiło odpowiedzieć na pytanie: co sie dzieje podczas obliczenia tego programu?
Program został umieszczony w materiałach Międzynarodowej Szkoły Loglanu, Zaborów'83. Program wyróżnia się tym, że występują w nim rekurencyjna procedura *F* i instrukcja powtarzania **for**. Pozostałe instrukcje są banalnie proste.
Przykład ten *ilustruje* lukę pomiędzy algorytmiką i programowaniem. W programie studiów informatycznych uczymy pisania tekstów programów (składni), uczymy też analizy algorytmów i struktur danych.

Ale kto uczy rozumienia co program robi? czy program robi too co powinien?

Spis treści

1. Wstęp
2. Przykład ilustrujący lukę – zagadka: PawelG
3. Odpowiedź M – Dowód matematyczny
4. Odpowiedź G – Dowód semantyczny, graficzny
5. Narzędzia - Rachunek programów
6. Wnioski
7. *Propozycje dla dydaktyki*

Program – zagadka

program PawelG

```
program PawelG;
(* autor Paweł Gburzyński, 1983,  *)
    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;
      (* tu tekst procedury 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
```

```
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;
```

Co ten program robi?

Nie martw się jeśli nie umiesz odpowiedzieć na to pytanie. Wiele mądrych głów także nie umiało na nie odpowiedzieć.

Twierdzenie (Tw. 1)

Program PawelG drukuje wszystkie permutacje liczb $1, \ldots, n$ *i zatrzymuje się.*

Przedstawimy **dwa** sposoby analizowania tego programu.

Dowód M – matematyczny

Cel

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.

O narzędziach

Jakie narzędzia, jakie aksjomaty i reguły wnioskowania są niezbędne dla przeprowadzenia dowodu? ▸ Popatrzymy na narzędzia
Aksjomaty algorytmicznej teorii liczb naturalnych $\mathcal{ATN}$

$$\forall_x\, x+1 \neq 0 \tag{I}$$
$$\forall_x \forall_y\, x+1 = y+1 \Rightarrow x = y \tag{M}$$
$$\forall_x\, \{y := 0; \textbf{while } y \neq x \textbf{ do } y := y+1 \textbf{ od}\}(x = y) \tag{S}$$
$$x - 1 \stackrel{df}{=} \{\ \ w := 0;\ \textbf{if } x \neq 0 \textbf{ then } \ \textbf{while } w+1 \neq x \textbf{ do } w := w+1 \textbf{ od fi}\ \ \}w \tag{P}$$

Czy możemy uznać, że 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.

Deklaracja F jest aksjomatem

$$\{\text{call F}\}\varphi \Leftrightarrow \left\{ \begin{array}{l} \text{block} \\ \quad \text{var } i : \ \mathit{integer}; \\ \text{begin} \\ \quad \text{if } k = n + 1 \text{ then call } \mathit{DrukujA} \\ \quad \text{else} \\ \quad\quad \text{for } i := 1 \text{ to } n \text{ do} \\ \quad\quad\quad \text{if } A[i] = 0 \text{ then} \\ \quad\quad\quad\quad A[i] := k;\ k := k + 1; \\ \quad\quad\quad\quad \text{call } F; \\ \quad\quad\quad\quad A[i] := 0;\ k := k - 1; \\ \quad\quad\quad \text{fi} \\ \quad\quad \text{od} \\ \quad \text{fi} \\ \text{end block} \end{array} \right\} \varphi$$

Dokładniej o schemacie aksjomatów

Niech φ będzie dowolną formułą. Deklarację procedury F możemy odczytywać dwojako:

- jako przepis w jaki sposób wykonać polecenie $\boxed{\text{call } F}$, lub
- jako schemat (nieskończenie wielu) aksjomatów:

  warunek φ zachodzi po wykonaniu polecenia $\boxed{\text{call } F}$

  wtedy i tylko wtedy gdy

  warunek φ zachodzi po wykonaniu instrukcji bloku z treścią procedury F.

Co więcej, w powyższej formule, w treści procedury F można wszystkie wystąpienia identyfikatora i *równocześnie* zastąpić przez dowolny inny identyfikator,

Teoria T'

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.

Formuła delta

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

$$\delta(n, k, i_1, \ldots, i_{k-1}) \stackrel{df}{\equiv}$$

$$\left(\begin{array}{l} (1 \leq k \leq n+1) \wedge \\ \quad \bigwedge_{j=1}^{k-1} \Big((1 \leq i_j \leq n) \wedge \ (A[i_j] = j) \Big) \wedge \\ \left\{ \begin{array}{l} z := 0; \\ \text{for } j := 1 \text{ to } n \text{ do} \\ \quad \text{if } A[j] = 0 \text{ then } z := z + 1 \text{ fi} \\ \text{od} \end{array} \right\} (z = n - k + 1) \end{array} \right)$$

δ powiada: liczby $1, \ldots . k-1$ są w tablicy A, pozostałe miejsca to zera.

Niezmiennik

Lemat (1)

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

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

Dowód 1/

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

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

Dowód 2/

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

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

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

$$\mathcal{T}' \vdash \delta(n, n+1, i_1, \ldots, i_n) \implies \{call\ DrukujA\}(\bigwedge_{j=1}^{n} A[i_j] = j) \quad (2)$$

Dowód 3/

Postępując podobnie udowodnimy formułę

$$\left(\delta(n, n+1, i_1, \ldots, i_n) \implies \{call\ DrukujA\}\, \delta(n, n+1, i_1, \ldots, i_n)\right) \quad (3)$$

Można to zapisać nieco inaczej

$$\mathcal{T}' \vdash \left(\begin{array}{l} \delta(n, k, i_1, \ldots, i_n) \\ \wedge\, k = n+1 \end{array} \right) \implies \{call\ DrukujA\}\, \delta(n, n+1, i_1, \ldots, i_n) \quad (4)$$

Dowód 4/

Skorzystamy z reguły wnioskowania IF+ o instrukcji if.

$$\left(\begin{array}{l} \delta(n,k,i_1,\ldots,i_n) \\ \wedge\, k = n+1 \end{array} \right) \Longrightarrow \left\{ \begin{array}{l} \text{if } k = n+1 \\ \text{then call } \mathit{DrukujA} \\ \text{else} \\ \quad \text{(* tu cokolwiek *)} \\ \text{fi} \end{array} \right\} \delta(n,n+1,i_1,\ldots,i_n) \tag{5}$$

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

Dowód 5/

Wstawiamy to czego nam potrzeba, dbając o to by zmienna $\underline{i_n}$ występowała tylko w tej instrukcji for.

$$\left(\begin{array}{l} \delta(n,k,i_1,\ldots,i_n) \\ \wedge\, k = n+1 \end{array} \right) \Longrightarrow \left\{ \begin{array}{l} \text{if } k = n+1 \\ \text{then call } DrukujA; \\ \text{else} \\ \quad \text{for } \underline{i_n} := 1 \text{ to } n \text{ do} \\ \quad\quad \text{if } A[\underline{i_n}] = 0 \text{ then} \\ \quad\quad\quad A[\underline{i_n}] := k;\ k := k+1; \\ \quad\quad\quad \text{call } F; \\ \quad\quad\quad A[\underline{i_n}] := 0;\ k := k-1; \\ \quad\quad \text{fi} \\ \quad \text{od} \\ \text{fi} \end{array} \right\} \delta(n,n+1,i_1,\ldots,i_n) \quad (6)$$

Dowód 6/

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

$$\delta(n, n{+}1, i_1, \ldots, i_n) \implies \left\{ \begin{array}{l} \textbf{block} \\ \quad \textbf{var } i : \textit{ integer} \\ \textbf{begin} \\ \quad \textbf{if } k = n+1 \\ \quad \textbf{then } \textbf{call } \textit{DrukujA}; \\ \quad \textbf{else} \\ \quad\quad \textbf{for } i := 1 \textbf{ to } n \textbf{ do} \\ \quad\quad\quad \textbf{if } A[i] = 0 \textbf{ then} \\ \quad\quad\quad\quad A[i] := k;\ k := k+1; \\ \quad\quad\quad\quad \textbf{call } F; \\ \quad\quad\quad\quad A[i] := 0;\ k := k-1; \\ \quad\quad\quad \textbf{fi} \\ \quad\quad \textbf{od} \\ \quad \textbf{fi} \\ \textbf{end block} \end{array} \right\} \delta(n, n{+}1, i_1, \ldots, i_n)$$

(7)

Dowód 7/

Program występujący w powyższej formule (7) to treść procedury F, możemy więc zastosować aksjomat *czyli* deklarację procedury F

$$\mathcal{T}' \vdash \delta(n, n+1, i_1, \ldots, i_n) \implies \{\text{call } F\}\ \delta(n, n+1, i_1, \ldots, i_n) \quad (8)$$

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

▸ Przeskocz do końca dowodu Lematu 1

Dowód 8/

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

Fakt (1)

$$\left(\delta(n,k,i_1,\ldots,i_{k-1}) \wedge A[p]=0\right) \implies \left\{ \begin{array}{l} A[p] := k; \\ k := k+1 \end{array} \right\} \left(\delta(n,k+1,i_1,\ldots,i_{k-1},i_k) \wedge i_k = p\right) \tag{9}$$

Dowód 9/

Fakt (1) udowodnimy, dwukrotnie stosując aksjomat instrukcji przypisania. Z założenia $\delta(n, k, i_1, \ldots, i_{k-1})$ wynika, że tablica A zawiera wszystkie liczby $1, \ldots, k-1$ czyli $\bigwedge_{j=1}^{k-1} A[i_j] = j$.

Ponadto miejsce $A[p]$ tablicy A jest wolne, a więc $p \neq i_j$ dla $j = 1, \ldots, k-1$.

Stąd wynika, że $\{A[p] := k\}(\bigwedge_{j=1}^{k} A[i_j] = j) \wedge A[p] = k)$.

Z kolei, $(z = n - k + 1) \implies \{k := k + 1\}(n - k)$.
Wynika stąd

$$\left(\begin{array}{l} (\bigwedge_{j=1}^{k-1} A[i_j] = j) \\ \wedge z = n - k + 1 \end{array} \right) \implies \left\{ \begin{array}{l} A[p] := 0; \\ k := k + 1 \end{array} \right\} \left(\begin{array}{l} (\bigwedge_{j=1}^{k} A[i_j] = j) \\ \wedge\ (z = n - k) \end{array} \right) \qquad (10)$$

Dowód 10/

Podobnie, następująca formuła (11) jest twierdzeniem teorii $\mathcal{T}'$.

Fakt (2)

$$\left(\begin{array}{l} (\delta(n, k+1, i_1, \ldots, i_{k-1}, p)) \\ \wedge A[p] = k \end{array} \right) \Longrightarrow \left\{ \begin{array}{l} A[p] := 0; \\ k := k-1 \end{array} \right\} \left(\ \delta(n, k, i_1, \ldots, i_{k-1}) \ \right) \qquad (11)$$

Dowód tego faktu przebiega podobnie do wczesniejszego.

Dowód 11/ – krok indukcyjny

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

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

(*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

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

Dowód 12/

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

$$\left(\begin{array}{l} A[r_j] = 0 \wedge \\ \delta(n, k-1, i_1, \ldots, i_{k-2}) \end{array} \right) \Longrightarrow \left\{ \begin{array}{l} A[r_j] := k;\ k := k+1; \\ call\ F; \\ A[r_j] := 0; k := k-1 \end{array} \right\} \delta(n, k-1, i_1, \ldots, i_{k-2}) \quad (12)$$

Wynika to z Faktów (1) i (2).

Dowód 13/

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

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

Teraz wykorzystamy Fakt (2)

$$\delta(n, k, i_1, \ldots, i_{k-2}, r_j) \Rightarrow \left\{ \begin{array}{l} A[r_j] := 0; \\ k := k-1 \end{array} \right\} \delta(n, k-1, i_1, \ldots, i_{k-2}). \tag{13}$$

Dowód 14/

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

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

Przyjmijmy następujące oznaczenia

$$\theta_i(n, k-1, i_1, \ldots, i_{k-2}) \stackrel{df}{\equiv} \begin{cases} A[i] \neq 0 \wedge \delta(n, k-1, i_1, \ldots, i_{k-2}) & \text{gdy} A[i] \neq 0 \\ A[i] = 0 \wedge \left\{ \begin{array}{l} A[i] := k; \\ k := k+1; \\ call\ F; \\ A[i] := 0; \\ k := k-1 \end{array} \right\} \delta(n, k-1, i_1, \ldots, i_{k-2}) & \text{gdy} A[i] = 0 \end{cases} \tag{14}$$

Dowód 15/

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

$$\mathcal{T}' \vdash \delta(n, k-1, i_1, \ldots, i_{k-2}) \implies \left\{ \begin{array}{l} \text{if } A[i] = 0 \text{ then} \\ \quad A[i] := k; \\ \quad k := k + 1; \\ \quad \mathit{call}\ F; \\ \quad A[i] := 0; \\ \quad k := k - 1 \\ \text{fi} \end{array} \right\} \delta(n, k-1, i_1, \ldots, i_{k-2}) \tag{15}$$

Dowód 16/

Zastosujemy następującą regułę (**FA**) wnioskowania pozwalającą na wprowadzenie kwantyfikatora ogólnego ograniczonego po instrukcji for

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

Dowód 17/

by uzyskać

$$\mathcal{T}' \vdash \delta(n, k-1, i_1, \ldots, i_{k-2}) \implies \left\{ \begin{array}{l} \text{for } i := 1 \text{ to } n \text{ do} \\ \quad \text{if } A[i] = 0 \text{ then} \\ \quad\quad A[i] := k; \\ \quad\quad k := k + 1; \\ \quad\quad \mathit{call}\ F; \\ \quad\quad A[i] := 0; \\ \quad\quad k := k - 1 \\ \quad \text{fi} \\ \text{od} \end{array} \right\} \delta(n, k-1, i_1, \ldots, i_{k-2}) \tag{16}$$

Dowód 18/

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

$$\mathcal{T}' \vdash \delta(n, k-1, i_1, \ldots, i_{k-2}) \implies \left\{ \begin{array}{l} \text{if } k = n+1 \text{ then} \\ \quad \textit{call } \textit{DrukujA} \\ \text{else} \\ \text{for } i := 1 \text{ to } n \text{ do} \\ \quad \text{if } A[i] = 0 \text{ then} \\ \qquad A[i] := k; \\ \qquad k := k+1; \\ \qquad \textit{call } F; \\ \qquad A[i] := 0; \\ \qquad k := k-1 \\ \quad \text{fi} \\ \text{od} \\ \text{fi} \end{array} \right\} \delta(n, k-1, i_1, \ldots, i_{k-2}) \tag{17}$$

Dowód 19/

Zmienna i różni się od wszystkich innych, możemy zastosowac aksjomat instrukcji bloku

$$\mathcal{T}' \vdash \delta(n, k-1, i_1, \ldots, i_{k-2}) \implies \left\{ \begin{array}{l} \text{block} \\ \quad \text{var } i : \textit{integer}; \\ \text{begin} \\ \text{if } k = n+1 \\ \text{then} \\ \quad \textit{call DrukujA} \\ \text{else} \\ \text{for } i := 1 \text{ to } n \text{ do} \\ \quad \text{if } A[i] = 0 \text{ then} \\ \qquad A[i] := k; \\ \qquad k := k+1; \\ \qquad \textit{call } F; \\ \qquad A[i] := 0; \\ \qquad k := k-1 \\ \quad \text{fi} \\ \text{od} \\ \text{fi} \\ \text{end block} \end{array} \right\} \delta(n, k-1, i_1, \ldots, i_{k-2}) \qquad (18)$$

Dowód 20/

Teraz skorzystamy z deklaracji *czyli* aksjomatu instrukcji call F i uzyskamy

$$\mathcal{T}' \vdash \delta(n, k-1, i_1, \ldots, i_{k-2}) \implies \{\text{call } F\}\, \delta(n, k-1, i_1, \ldots, i_{k-2}). \quad (19)$$

co kończy dowód lematu 1.

□

Dowód twierdzenia 1

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 $1, \ldots, n$.

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;.

Lemat 2

Definiujemy warunek początkowy

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

i warunek końcowy

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

Nietrudno teraz udowodnić odmieniony wariant lematu 1.

Lemat (2)

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

Dowód lematu 2

Dowód tego lematu naśladuje dowód lematu 1.
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 2

Lemat 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 (3)

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

Nie będziemy formalizować tego dowodu.

Dowód lematu 3 – Koniec dowodzenia

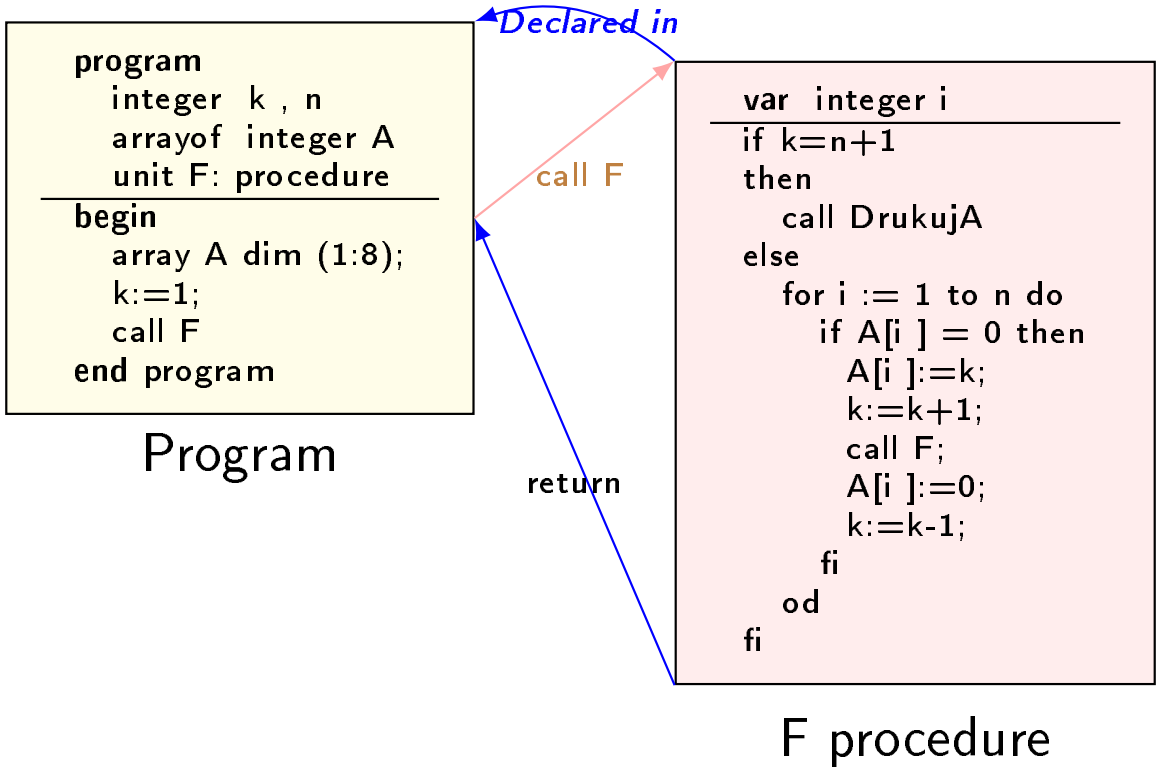
Nasz argument jest prosty. Podczas wykonywania instrukcji for powtarzamy $(n-k+1)$ razy czynność następującą: dla $j = k, \ldots, n$, zapisz liczbę k na miejscu $A[r_j]$ i na pozostałych $(n-k)$ wolnych miejscach wytwórz $(n-k)!$ **różnych**, permutacji. Widzimy, że utworzone w ten sposób $(n-k+1)!$ permutacje są różne, nie ma powtórzeń.

Koniec dowodu lematu 3
i koniec dowodu twierdzenia 1.
□

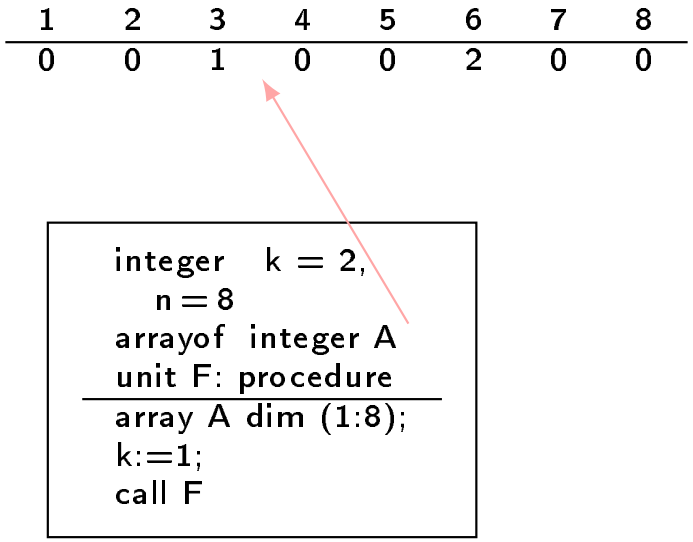
Dowód G – semantyczny, graficzny

Statyczna struktura modułów programu

Statyczna struktura modułów programu



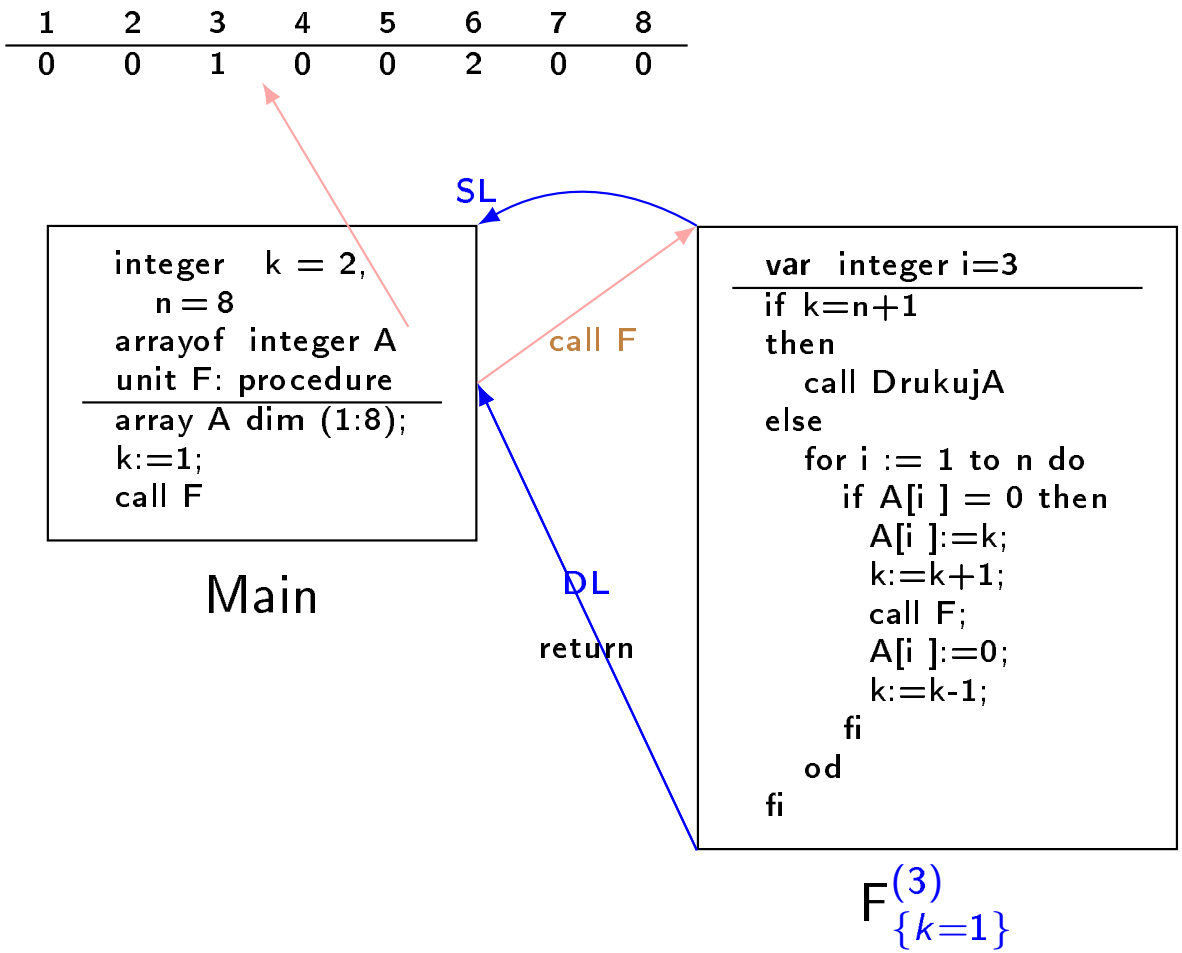
Obliczenie - pierwszy krok - program główny



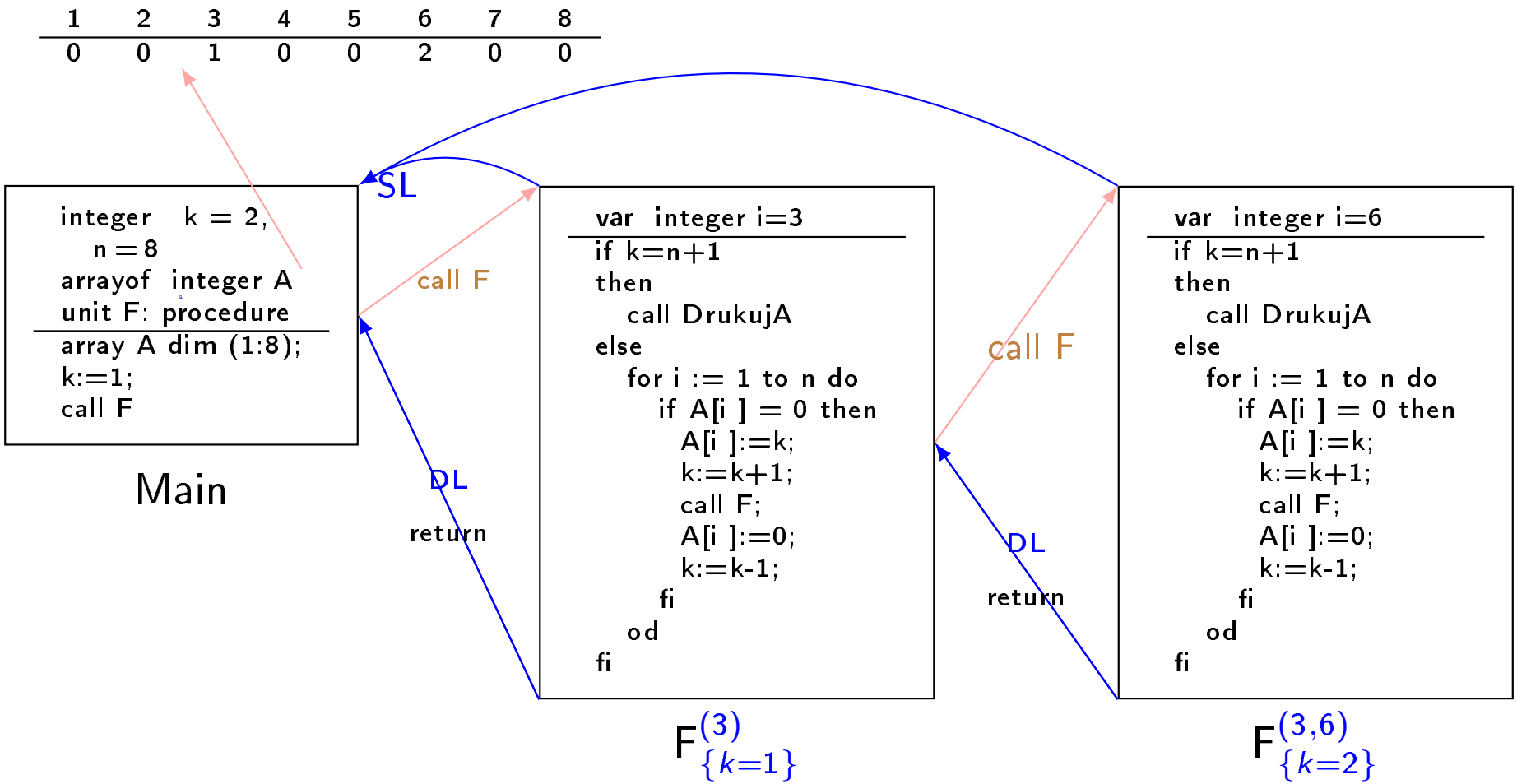
Wykonywanie programu głównego.

Odczytanie danej – n, utworzenie obiektu tablicowego A o rozmiarze n, kładziemy $k = 1$ i rozpoczynamy wykonywanie polecenia F zdefioniowanego w deklaracji procedury.

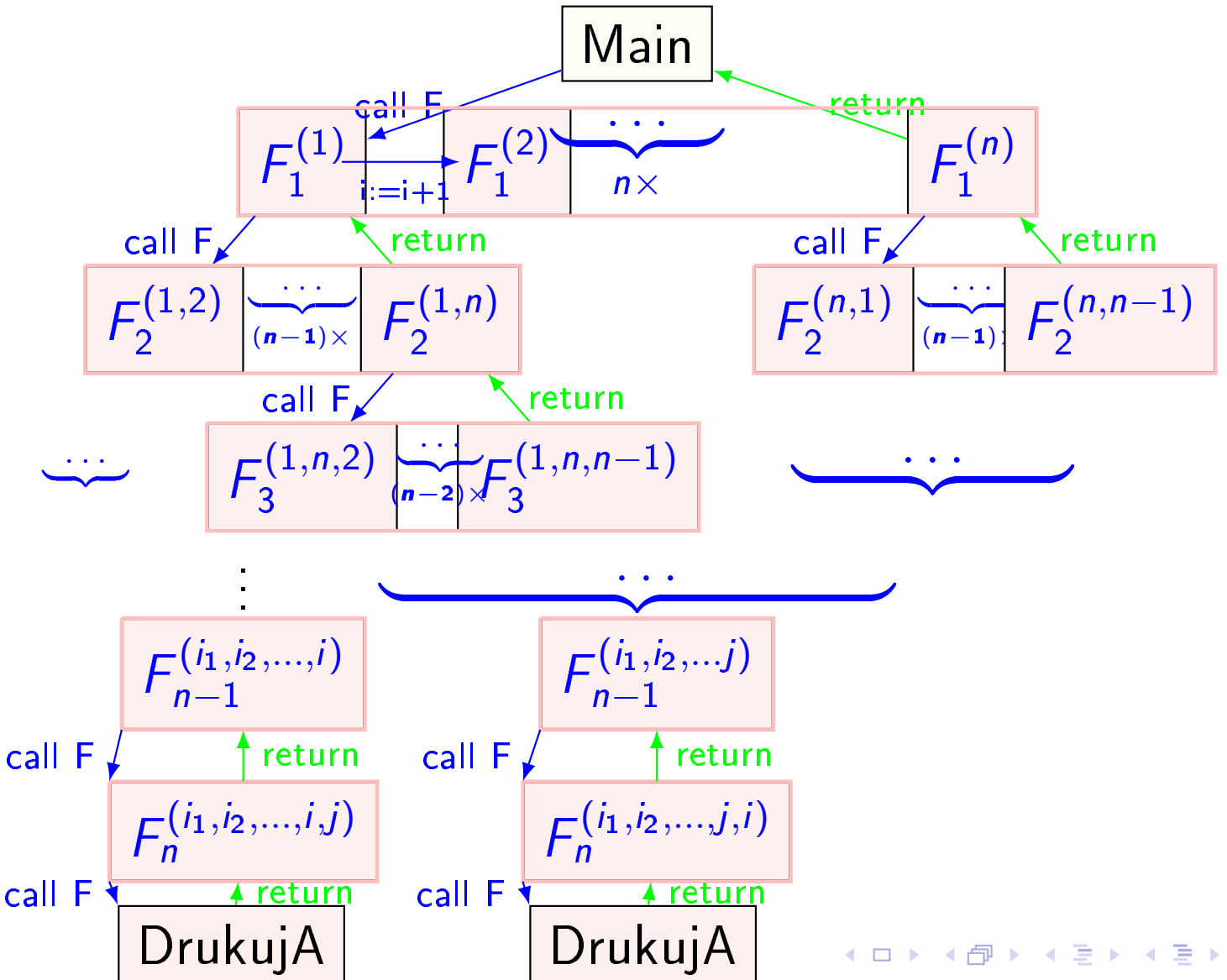
Czas wykonania – instrukcja F - za którymś razem



Czas wykonania – trochę później



Drzewo obliczeń



Rekord aktywacji

Definicja

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.)

Stan obliczenia

Definicja

Konfiguracja (tj. stan) obliczenia programu PawelG jest zbiorem rekordów aktywacji procedury F i rekordu Main programu głównego.

Protokół **call**

Definicja

*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.*

Ostrzeżenie

Definicja powyższa jest uproszczona.
Nie opisaliśmy protokołu przekazywania parametrów – argumentów i wyników.

Drzewo możliwych stanów obliczenia

Definicja

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.

Dowód

Dowód twierdzenia ?? wynika wprost ze zgromadzonych poniżej obserwacji.

Fakt

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

Uwaga 2

Fakt

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 ?? i ?? węzeł jest obwiedziony czerwoną linią grupującą w ten sposób stany tego węzła.*

Uwaga 3

Fakt

Liczba liści drzewa aktywacji jest równa $n!$.

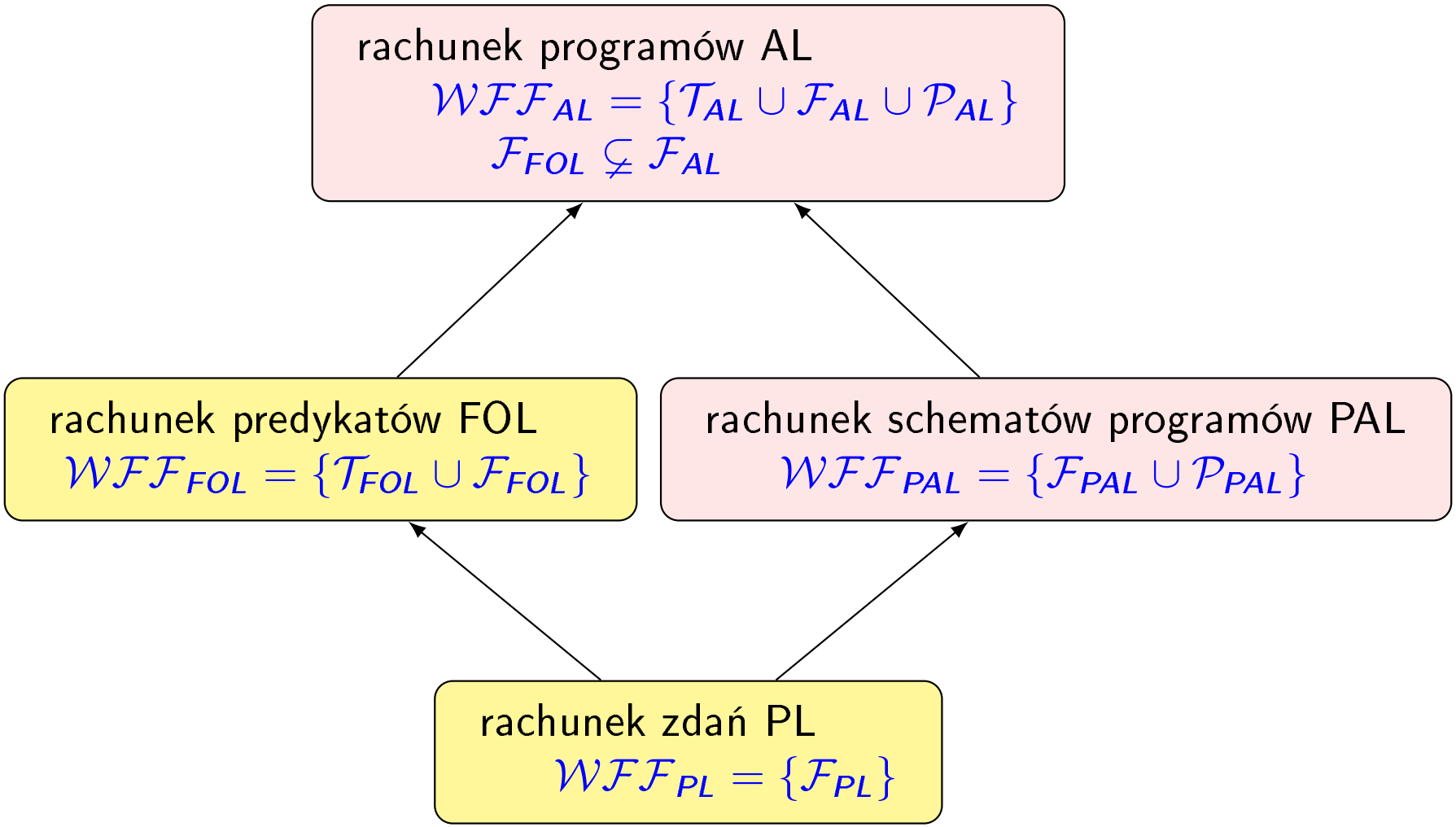
Stan tablicy A podczas wykonywania tej procedury z wartością zmiennej $k = n + 1$ jest taki, że wypełnione jest każde miejsce w tej tablicy. Wartościami zapisanymi w tej tablicy są liczby $1, \ldots, n$ i żadna liczba się nie powtarza. Podsumowując, wydrukowane zostaną wszystkie permutacje liczb $1,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

Rachunek programów – Narzędzia programisty

Pan Jourdain: Daję słowo, zatem ja już przeszło czterdzieści lat mówię prozą, nie mając o tem żywnego pojęcia! Jestem panu najszczerzej obowiązany, żeś mnie pouczył.

Mieszczanin szlachcicem

Miejsce rachunku programów



▸ Powracamy

Aksjomat instrukcji przypisania

$$\boxed{\left(\{x := \tau\}\varphi \Leftrightarrow \varphi(x/\tau)\right)} \qquad \text{(Assign)}$$

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

Aksjomat instrukcji złożonej

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

Aksjomat instrukcji warunkowej **if**

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

Pomocnicze reguły wnioskowania dla instrukcji **if**

$$\frac{\gamma,\ \{K\}\varphi}{\{\text{if } \gamma \text{ then } K \text{ else } M \text{ fi}\}\varphi} \qquad \text{(IF+)}$$

$$\frac{\neg\gamma,\ \{M\}\varphi}{\{\text{if } \gamma \text{ then } K \text{ else } M \text{ fi}\}\varphi} \qquad \text{(IF–)}$$

Instrukcja **for** – trzy schematy aksjomatów

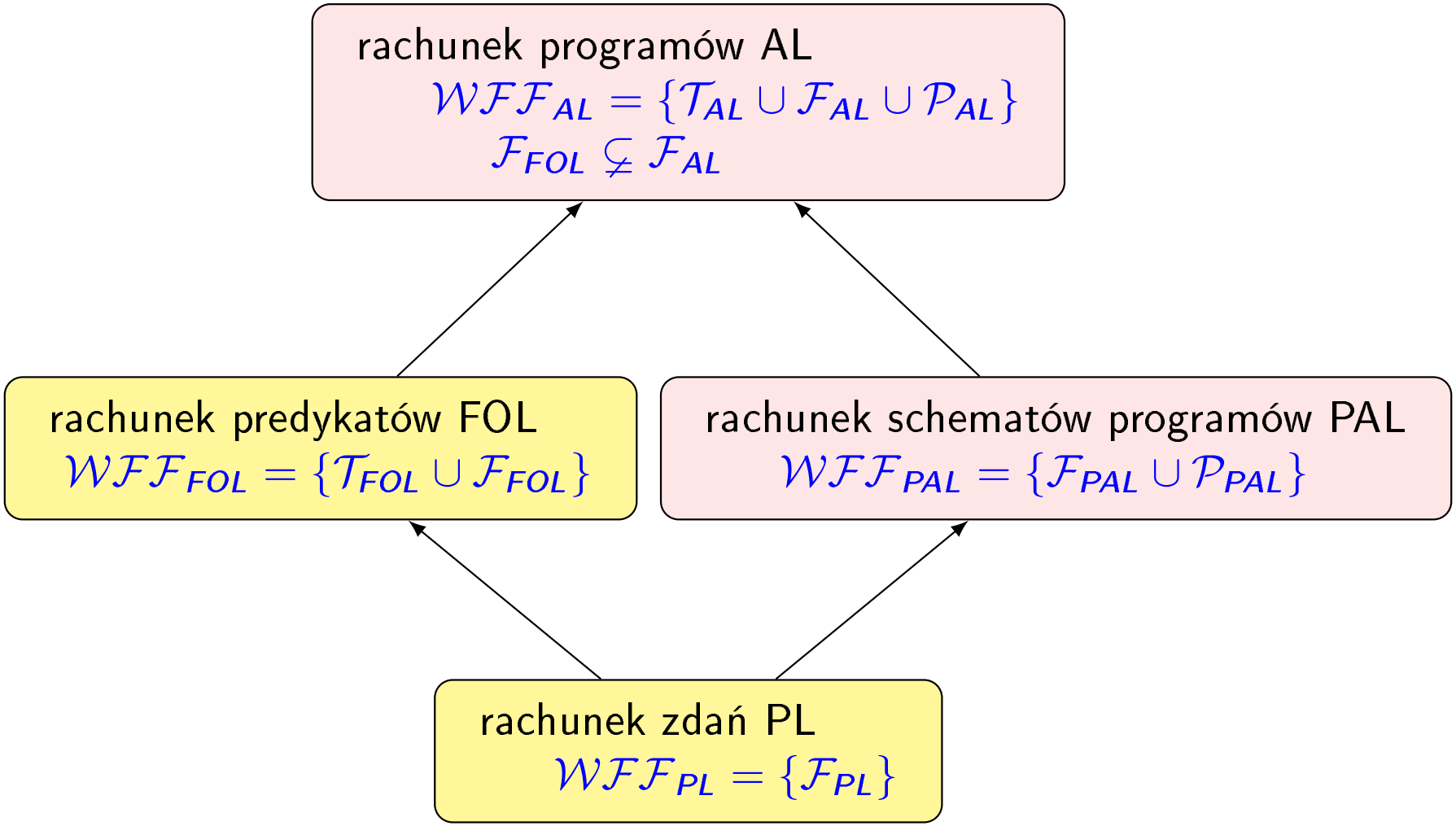
$$\boxed{(B < A) \Rightarrow \big(\{\textbf{for}\ \ i := A\ \textbf{to}\ B\ \textbf{do}\ I\ \textbf{od}\}\varphi \Leftrightarrow \varphi\big)} \qquad \text{(F1)}$$

$$\boxed{(B = A) \Rightarrow \big(\{\textbf{for}\ i := A\ \ \textbf{to}\ B\ \textbf{do}\ I\ \textbf{od}\}\varphi \Leftrightarrow (\{i := A;\ I\}\varphi)\big)} \qquad \text{(F2)}$$

$$\boxed{\Big(\{\textbf{for}\ i := A\ \textbf{to}\ B + 1\ \textbf{do}\ I\ \textbf{od}\}\varphi \Leftrightarrow \{\textbf{for}\ i := A\ \textbf{to}\ B\ \textbf{do}\ I\ \textbf{od};\ i := i + 1}$$

(F3)

Porównanie 4 rachunków



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

  unit DrukujA: procedure;

  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")
```

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

  unit DrukujA: procedure;

  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")
```

Wnioski

Salwicki: Jeśli udowodnisz, że program jest poprawny to i komputer Cię posłucha! i postąpi tak jak to udowodniłeś.

Morał

Co warto zapamiętać? 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:

Twierdzenie o pełności rachunku programów AL

Dla każdego programu P,
Niech pewna własność semantyczna w programu P będzie wyrażalna formułą (algorytmiczną) ψ.
Wtedy nastepujące zdania są równoważne:

- *W* algorytmicznej teorii $\mathcal{T}$ istnieje dowód formuły ψ,
- *Formuła* ψ jest prawdziwa w każdym modelu teorii $\mathcal{T}$.

O tym właśnie mówi twierdzenie o *pełności* rachunku programów tj. logiki algorytmicznej.

Waga aksjomatu liczb naturalnych S

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.

Własność stopu wynika z poniższego aksjomatu (S) struktury liczb naturalnych $\mathfrak{N}$.

$$\mathfrak{N} \models \forall_x \{y := 0; \textbf{while } y \neq x \textbf{ do } y := y + 1 \textbf{ od}\}(x = y) \qquad (S)$$

A jeżeli wiemy, że ten program ma obliczenie skończone to własność stopu innego programu wyprowadzimy z własnośći (S).

Metodologia stosowania procedur

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 była to instrukcja $\boxed{\textbf{call } F}$.

(*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 call $\boxed{F_{\mathfrak{A}}}$.

W wielu przypadkach dowód taki jest banalny. Sprawa może się skomplikować gdy wprowadzana definicja jest niejawna lub gdy w deklaracji występuje konstrukcja while.

Zapewnienie

Zapamiętaj

Jeżeli dokonasz analizy programu PawelG, i udowodnisz twierdzenie 1, 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.

Może innym razem?

Sugestie dla dydaktyki

Dziękuję za uwagę.

Komentarze? Pytania?

Typy pierwotne

- Integer – ten typ to pierścień z dodatkowym ważnym aksjomatem:

  Dla każdej liczby dodatniej $x > 0$ program
  {y:=0; **dopóki** y $\neq$ x **powtarzaj** y:=y+1 **doskutku** }
  ma obliczenie skończone,

- Boolean –
- Character –
- Real – ten typ to ciało spełniające prawo Archimedesa
  $\forall_{0<a<b}\{c := a;\ \text{while } c < b \text{ do } c := c + a \text{ od}\}(b \leq c)$
- String – operacja konkatenacji i aksjomaty Tarskiego, tw. Grzegorczyka

Instrukcja przypisania i programy liniowe

Aksjomat ten po raz pierwszy pojawił się 1963 r. w książce "*Mathematics of metamathematics*"Heleny Rasiowej i Romana Sikorskiego. Ciekawe?
Program liniowy – ciąg instrukcji przypisania. Programy liniowe a skierowane grafy acykliczne (dag).
Przykłady.

Formuły algorytmiczne – najsłabsze warunki wstępne

Deklaracje procedur i funkcji – Definicje rozszerzające język

Sygnały i ich obsługa

Klasy czyli definiowanie typów danych