13. přednáška

Tvorba dokázaných programů podle Dijkstry

Odkazy

- Text přednášky je obsažen v:
  - Skripta „Vybrané kapitoly...“, kap. 13
  - Studijní opora IAL – kap. 6.3

Základní matematický aparát

- Na rozdíl od předchozích přístupů, v nichž se dokazuje správnost intuitivně zapsaných algoritmů, je zde algoritmus, který vzniká na základě matematických transformací, ve své závěrečné podobě úplný a správný.
- Pro tvorbu dokázaných programů zavedl Dijkstra vlastní algoritmický jazyk a vlastní matematický aparát. Po stručném úvodu těchto formálních nástrojů bude uvedeno několik ilustrativních příkladů dokázaných algoritmů, které se vztahují k předmětu IAL.

Předpokládejme, že v průběhu výpočtu největšího společného dělitele dvou přirozených čísel X,Y projdeme stavy x,y, pro něž platí:

NSD(x,y) = NSD(X,Y) and (0<x<=X) and (0<y<=Y)

kde NSD je označení funkce největšího společného dělitele, X,Y jsou konstanty pro určitý výpočet a určují počáteční hodnoty pracovních proměnných x,y. Podobným vztahům budeme říkat “podmínky” nebo “predikáty”.

Jestliže se systém po skončení své aktivity určitě dostane do stavu splňujícího podmínku **P** pak říkáme, že systém určitě ustaví pravdivost **P**.

Každý predikát **P** je **definován v každém bodu** stavového prostoru za předpokladu, že v **každém bodu** tohoto prostoru má hodnotu **“true”** nebo **“false”**. Nadále budeme predikáty používat pro označení *množiny takových bodů stavového prostoru, v nichž je* ***predikát pravdivý*****.**

O predikátech **P** a **Q** říkáme, že jsou si rovny (“**P**=**Q**”), jestliže označují stejnou podmínku nebo jestliže označují stejnou množinu stavů. Dále budeme používat dva speciální predikáty s vyhrazeným označením “**T**” a “**F**”.

T je predikát pravdivý ve všech bodech uvažovaného prostoru. Odpovídající množinou je universum.

F je predikát nepravdivý ve všech bodech uvažovaného prostoru a odpovídá mu množina prázdná.

Předpokládejme výpočetní mechanismus označený **S** a podmínku **R**, kterou musí splňovat stav mechanismu po skončení své aktivity. Podmínku **R** nazvěme "**konečná podmínka**" (*postcondition*).

Pak zápis **wp(S,R)** bude označovat **nejslabší počáteční podmínku** (*weakest precondition*), která zaručuje, že mechanismus se dostane v konečné době do stavu splňujícího konečnou podmínku **R**.

Pozn. Není-li nejslabší počáteční podmínka splněna, nelze zaručit, že se mechanismus **S** dostane do stavu splňujícího **R**, i když to nesplnění podmínky nevylučuje. Při nesplnění podmínky (**S**,**R**) se může mechanismus dostat do stavu nesplňujícího **R** nebo do stavu nekonečné aktivity.

Množina všech možných konečných podmínek pro daný mechanismus je tak rozsáhlá, že její znalost, např. v tabelární podobě, která by určila rychlé určení (**S**,**R**) je prakticky nepoužitelná.

Proto je definice sémantiky mechanismu daná ve formě **pravidel**, popisujících jak odvodíme k dané konečné podmínce **R** odpovídající *nejslabší počáteční podmínku* **ws(S,R).** Pro daný mechanismus **S** a daný predikát **R** je takové pravidlo, jež dá za výsledek (**S**,**R**) označováno jako "**transformace predikátu**" a definuje se jí sémantika mechanismu **S**.

Nejčastěji nás však nezajímá úplná sémantiku mechanismu. Mechanismu **S** používáme pro ustavení pravdivosti určité konečné podmínky **R**, pro niž byl mechanismus navržen. Ani pro tuto určitou konečnou podmínku **R** nás nezajímá přesná a úplná forma **wp**(**S**,**R**), ale obvykle o něco silnější "**postačující**" podmínka **P**, pro niž platí:

$$\mathbf{P} \rightarrow \mathbf{wp}(\mathbf{S},\mathbf{R}) \text{ pro všechny stavy}$$

Pak P je postačující počáteční podmínka. V terminologii množin to znamená, že množina stavů označená symbolem **P** je podmnožinou stavů, kterou označuje zápis **wp**(**S**,**R**).

Chápeme-li transformaci predikátu **wp**(**S**,**R**) jako funkci konečné podmínky **R**, pak má tato funkce několik základních vlastností:

1) Pro každý mechanismus **S** platí **wp**(**S**,**F**)=**F**

Této vlastnosti se říká "zákon vyloučeného zázraku".

2) Pro každý mechanismus S a konečné podmínky **Q** a **R** takové, že platí **Q** → **R** pro všechny stavy, platí také

$$wp(S,Q) \rightarrow wp(S,R)$$

pro všechny stavy. Této vlastnosti se říká "zákon monotónnosti".

3) Pro každý mechanismus **S** a konečné podmínky **Q** a **R** platí:

wp(*S*,*Q*) *and* ***wp*(*S*,*R*)** =***wp*** **(*S*,*Q*** *and* ***R*)**

pro všechny stavy a také

wp(*S*,*Q*) *or* ***wp*(*S*,*R*)** = ***wp*(*S*,*Q*** *or* ***R*)**

Definice základních mechanismů Dijkstrova jazyka

Definujme pro tvorbu základních mechanismů tyto elementární mechanismy:

1) Prázdný příkaz "**skip**„. Jeho sémantika je dána transformací: **wp**(*"skip"*,***R***) = ***R***

2) Příkaz zastavení v důsledku chybového stavu "**abort**", se sémantikou ***wp***(*"abort"*,***R***) = ***F***

3) Přiřazovací příkaz ***wp***(*"x:=**E**"*,***R***) = R_x^E

kde zápisem R_x^E se rozumí **textová kopie** **R**, v níž je každý výskyt proměnné **x** nahrazen výrazem **E**

např. : **wp**("x:=7", x=7) = (7=7) = **T**

nebo **wp**("x:=7", x=6) = (7=6) = **F**

nebo

wp ("x:=x-1", x^2=>1) = $((x-1)^2$=>1) = (x=>2 or x<=0) = **(x<>1)** (* pro celá čísla *)

Pomocí BNF lze příkaz definovat zatím takto:

<příkaz>::="skip"|"abort"|<přiřazovací příkaz>

<přiřazovací příkaz>::=<proměnná>:=<výraz>

Pro některé účely rozšíříme přiřazovací příkaz o možnost paralelního přiřazení takto:

<přiřazovací příkaz>::=<proměnná>:=<výraz>|
<proměnná>,<přiřazovací příkaz>,<výraz>

Tento příkaz umožní např. zápisem **x1,x2:=E1,E2** přiřadit dvěma proměnným současně hodnoty dvou výrazů nebo zápisem **X,Y:=Y,X** provést vzájemnou výměnu hodnot dvou proměnných.

Definice složeného příkazu

Nejjednodušším způsobem, jak složit ze dvou funkcí jednu funkci novou je způsob, v němž hodnota první funkce slouží jako argument druhé. Již tradičně má notace takového složení tvar "$\mathbf{S_1;S_2}$" a jeho sémantika je dána vztahem:

$$wp("S_1;S_2",R) = wp(S_1,wp(S_2,R))$$

Tato definice se často označuje jako **"sémantická definice středníku"**. Jinými slovy říká: jestliže v posloupnosti "$S_1;S_2$" má mechanismus S_2 dosáhnout určitého konečného stavu splňujícího konečnou podmínku R, pak jeho nejslabší počáteční podmínku musí zaručit konečný stav mechanismu S_1. Nejslabší počáteční podmínka složeného mechanismu **"$S_1;S_2$"** k dosažení konečného stavu **R** je tedy dána nejslabší počáteční podmínkou mechanismu S_1.

Příklad: Sekvence příkazů: “x:=x+y; y:=x-y; x:=x-y realizují vzájemnou výměnu hodnot x a y bez další pomocné proměnné, tedy mechanismus, který v Dijsktrově jazyku může být popsán příkazem “x,y:=y,x”.

Důkaz: Dosadíme do vztahu pro složený příkaz a s pomocí vztahu pro přiřazovací příkaz (***wp***(*"x:=**E**",**R**) = R_x^{E}) dostaneme:

wp ("x:=x+y; y:=x-y; x:=x-y", (x=X) and (y=Y)) =

= **wp** ("x:=x+y; y:=x-y", wp("x:=x-y", (x=X) and (y=Y))) =

= **wp** ("x:=x+y; y:=x-y", (x-y=X) and (y=Y)) =

= **wp** ("x:=x+y", wp("y:=x-y", ((x-y)=X) and (y=Y))) =

= **wp** ("x:=x+y", ((x-(x-y))=X) and (((x-y)=Y)) =

= **wp** ("x:=x+y", (y=X) and (x-y)=Y) =

= (y=X) and ((x+y)-y)=Y =

= (y=X) and (x=Y) Q.E.D.

Řízené příkazy

Složitější kompozicí jednoduchých příkazů jsou řízené příkazy alternativní a repetiční řídicí struktury (**if** a **do**).

<příkaz> ::= ...|**if** <soubor řízených příkazů> **fi** |

do <soubor řízených příkazů> **od**

<soubor řízených příkazů> ::= <řízený příkaz> {!<řízený příkaz>}

<řízený příkaz>::=<řídicí hlavička příkazu>{;<příkaz>}

<řídicí hlavička příkazu>::=<Booleovský výraz> →→ <příkaz>

kde symbol "!" má funkci oddělovače jednotlivých alternativ, jejichž pořadí v souboru řízených příkazů nemá žádný význam.

Alternativní příkaz „**if**" má několik důležitých vlastností:

a) Všechny řídicí Booleovské výrazy musí být definované.

a) Obecně vede řídicí struktura "**if**" k nedeterminovanosti, protože pro každý počáteční stav, který způsobí, že více než jeden Booleovský výraz je pravdivý, může vybrán kterýkoliv z řízených výrazů.

c) Není-li v počátečním stavu žádný z Booleovských řídicích pravdivý, pak aktivace povede k zastavení s chybou a v tom případě je řídicí struktura "**if**" ekvivalentní příkazu "abort". K témuž vede i příkaz "**if**" s prázdným souborem řízených příkazů, tedy konstrukce "**if fi**".

Nejslabší počáteční podmínka příkazu “**if**” je stanovena takto: Nechť “**if**” je označení příkazu, jehož tvar je:

if

$B_1 \rightarrow\rightarrow S_1$

| $B_2 \rightarrow\rightarrow S_2$

| ...

| $B_n \rightarrow\rightarrow S_n$

fi

kde S_i je seznam příkazů řízených Booleovským výrazem B_i, pak pro libovolnou konečnou podmínku **R** platí:

wp *("**if**",**R**) = ((**Exist** j:1<=j<=n)[B_j]) and*

*((**ForAll** j:1<=j<=n)[B_j] →* **wp** *(S_j,**R**))*

Formální definice příkazu “**do**“

Nechť „**do**" je označení alternativního příkazu se svým souborem řízených příkazů. Nechť podmínky H_k(**R**) jsou definovány takto:

$$H_0(\mathbf{R}) = \mathbf{R} \text{ and not } (\mathbf{Exist}\ j{:}1<=j<=n)[B_j]$$

a pro k>0 $H_k(\mathbf{R}) = \mathbf{wp}\ (\text{"}\mathbf{if}\text{"}, H_{k-1}(\mathbf{R}))\ \text{or}\ H_0(\mathbf{R})$

pak **wp** („**do**",**R**) = (**Exist** k:k>=0)[H_k(**R**)]

H_k(**R**) je nejslabší poč. podmínka zaručující konec příkazu "**do**" po maximálně **k** průchodech cyklem. Každý průchod je určen výběrem některého z řídicích výrazů a aktivuje odpovídající řízené příkazy. H_k(**R**) současně zabezpečuje, že po ukončení příkazu "**do**" bude systém ve stavu splňujícím konečnou podmínku **R**.

Důkaz definice je uveden v opoře nebo ve skriptech.

Repetiční příkaz "**do**", jehož počáteční stav splňuje podmínku:

$H_0(\mathbf{R}) = \mathbf{R}$ *and not(**Exist** j:1<=j<=n)*$[B_j]$

je ekvivalentní prázdnému příkazu "skip". K témuž vede i prázdný soubor řízených příkazů, tedy konstrukce **do od**.

Teorém alternativního příkazu "if„

Nechť je dán příkaz "**if**" a predikát BB pro nějž platí:

BB*=(*Exist* *j:1<=j<=n)[B_j].

S použitím uvedených konvencí lze teorém příkazu "**if**" vyjádřit takto: nechť **P** a **Q** jsou predikáty pro něž platí:

*(ForAll j:1<=j<=n)[(**P** and B_j)→* **wp** *(S_j,**Q**)]*

a také

P* → *BB pak tedy platí

P* → wp *("if",Q)

Důkaz – viz opora: Podle definice příkazu "**if**" platí:

wp *("**if**",**Q**) =*

*((**Exist** j:1<=j<=n)[B_j]) and ((**ForAll** j:1<=j<=n)[B_j]* → **wp** *(S_j,**Q**))*

Musíme tedy dokázat:

P → *(**Exist** j:1<=j<=n)[B_j] and (**ForAll** j:1<=j<=n)[B_j]* → **wp** *(S_j,**Q**)*

Teorém invariance cyklu

Významný teorém programování, který odvodil C.A.R.Hoare,

Zaveďme pomocné formální prostředky:

Zápis ***wdec(S,t)*** (*weakest decrement*) je nejslabší počáteční podmínka pro takový počáteční stav, který zaručuje, že mechanismus **S** v konečné době sníží hodnotu **t**, kde **t** je celočíselná funkce proměnných programu. pak lze psát:

wdec(S,t) = **wp** ***("tau:=t;S", t<tau)***

Tento vztah lze podle definice složeného příkazu rozvést na:

wdec(S,t)= **wp** ***("tau:=t",*** **wp** ***(S,t<tau))*** a konečně na

wdec(S,t)= $[\mathbf{wp}\ (S,t<tau)]^{tau}_{t}$

kde pravá strana konečného vztahu se interpretuje tak, že ve výrazu podmínky bude každé ***tau*** nahrazeno hodnotou **t**.

Použití ***wdec(S,t)*** ilustruje následující příklad:

Nechť ***wdec(***"*x:=x-y, x+y)* je nejslabší podmínka toho, že příkaz

"x:=x-y"

sníží hodnotu funkce x+y.

Podle definice složeného příkazu lze psát:

$$\textbf{wdec}(\text{"}x:=x-y\text{"}, x+y) = [(\text{"}x:=x-y\text{"}, x+y < \textbf{tau}]^{tau}_{x+y} =$$

$$[x-y+y < tau\ ]^{tau}_{x+y} = x < x+y = y > 0$$

Je-li y>0, pak příkaz x:=x-y sníží hodnotu funkce ***x+y***.

Nechť je repetiční příkaz "DO" definován zápisem

do

$B_1 \rightarrow\rightarrow S_1$

! $B_2 \rightarrow\rightarrow S_2$

! ...

! $B_n \rightarrow\rightarrow S_n$

od

a predikát **BB** je definován:

$$\textbf{\textit{BB}} = (\textbf{\textit{Exist}}\ j{:}1<=j<=n)[B_j]$$

Pak lze teorém invariance repetičního příkazu formulovat takto:

Nechť **P** je predikát takový, že platí

$$(\textbf{ForAll}\ j{:}1<=j<=n)[(\textbf{P}\ \text{and}\ B_j)\rightarrow(\textbf{wp}(S_j,\textbf{P})\ \text{and}\ \textbf{wdec}(S_j,\textbf{t}))]$$

a současně platí

$$\textbf{P}\rightarrow \textbf{t}>0$$

pak platí:

$$\textbf{P}\rightarrow \textbf{wp}\ (\text{"DO"},\textbf{P}\ \text{and}\ (\text{not BB}))$$

Konečné tvrzení říká, že pokud počáteční stav zaručuje pravdivost predikátu **P** a jestliže je vybrán kterýkoliv z řídicích výrazů a jeho řízené příkazy jsou provedeny, pak po skončení aktivity zůstává predikát **P** pravdivý. Predikát **P** tedy zůstává pravdivý (invariantní, neměnný) bez ohledu na počet, kolikrát bude ten či onen řídicí výraz vybrán a jeho řízené příkazy provedeny.

Po skončení aktivity příkazu "**DO**", kdy už žádný z řídicích výrazů B není pravdivý, zaručuje konečný stav pravdivost tvrzení:

***P** and (not BB)*

První ze dvou vztahů zaručuje neustálé snižování funkce **t** a druhý zaručuje, že její hodnota je kladná. Funkce **t** je celočíselná funkce a je spolehlivým prostředkem zakončení aktivity repetiční konstrukce.

Závažnost teorému invariance pro cyklus spočívá v tom, že pravdivost jeho výroků nezávisí na počtu průchodů. Z toho vyplývá, že lze postavit o cyklu tvrzení i v případě, kdy počáteční stav neurčuje počet průchodů. Umožňuje to provést důkaz správnosti repetiční konstrukce, jehož délka není úměrná počtu průchodů cyklu.

Příklady

Největší společný dělitel dvou celých čísel

Nechť X,Y jsou celá čísla větší než 0. Hledáme největší společný dělitel (dále jen NSD) těchto čísel. Řešení má formálně tvar:

R : Z=NSD(X,Y) and (X>0) and (Y>0) (1)

Z definice NSD dvou celých čísel vyplývá:

NSD(X,Y) = NSD(Y,X) (2)

a **NSD(X,X) = X** (3)

Lze dokázat, že platí také

NSD(X,Y) = (NSD(X,Y-X) and (Y>X)) (4)

a z toho lze dále odvodit, že platí

NSD(X,Y) = NSD(X-Y,Y) and (X>Y)) (5)

NSD(X,Y) = NSD(X+Y,Y) = NSD(X,Y+X) (6)

Pro získání řešení je důležitý vztah (3) a jestliže konečný stav mechanismu zajistí relaci **x=y**, pak tento stav zajistí také řešení NSD(x,y)=x. Mechanismus musí také zajistit neměnnost (invarianci) relace:

P : NSD(X,Y) = NSD(***x***,y) and (0<***x***<X) and (0<y<Y) (7)

Z počátečního stavu **x=X** a **y=Y** bude mechanismus „*zpracovávat*“ (upravovat) **x** a **y** tak, aby se zachovala invariance **P** s cílem dosažení relace **x=y**. Vztah (4) resp. (5) nabízí změnu x a y jejich rozdílem. Prozkoumejme podmínku, za níž příkaz **"x:=x-y"** dosáhne žádoucí konečné podmínky **P**.

wp ("x:=**x-y**",P)=**(**NSD(**x-y**,y)=NSD(X,Y)**) and** (0<(**x-y**)<=X)
and (0<y<=Y) (8)

Jinými slovy, pro mechanismus "x:=x-y" se hledá podmínka **P** taková, aby platila pravá strana rovnice, která říká, že pro každou úpravu argumentů operace NSD musí být zajištěna shodnost výsledků (na základě platnosti P viz (7).

Z teorému invariance pro cyklus vyplývá, že najdeme-li **P** a B_i takové, že platí

$$(\mathbf{P} \text{ and } \mathbf{B}_i) \rightarrow \mathbf{wp}\ (S_i, P) \text{ and } \mathbf{wdec}(\mathbf{S}_i, \mathbf{t})$$

a

$$\mathbf{P} \rightarrow \mathbf{t} > 0$$

pak

$$\mathbf{P} \rightarrow \mathbf{wp}\ (\text{"}\mathbf{DO}\text{"}, \mathbf{P} \text{ and } (\text{not } \mathbf{BB}))$$

Připomeňme, že ***BB***=(***Exist*** *j:1<=j<=n)*[$\mathbf{B}_j$].

Ze vztahu (7)

P : NSD(X,Y) = NSD(***x***,y) and (0<***x***<X) and (0<y<Y) **(7)**

je vidět, že **P** implikuje všechny členy pravé strany vztahu (8)

wp ("x:=**x-y**",P)=**(**NSD(**x-y**,y)=NSD(X,Y)**) and** (0<(**x-y**)<=X) **and** (0<y<=Y) **(8)**

s výjimkou 0<(x-y).

Z toho vyplývá, že:

(**P** and (x>y)) → **wp** ("x:=x-y",**P**) (9)

a v důsledku symetrie tedy také:

(**P** and (y>x)) → **wp** ("y:=y-x",**P**)

Zbývá najít funkci **t**, která vyhovuje podmínkám teorému invariance.

Nechť **t**=x+y. Ze vztahu (7) platí, že

P→ **t**>0 a pak lze tedy odvodit:

wdec("x:=x-y",x+y) =

[**wp** (x:=x-y,**tau**>(x+y)]$^{\mathbf{tau}}_{\mathbf{t}}$ =

[t**au**>(x-y)=y]$^{tau}_{x+y}$ =

(x+y)>x = **y>0**

a pak tedy ze vztahu (7) platí, že

P → **wdec**("x:=x-y", x+y).

Výsledný program má tuto strukturu:

"Ustav počáteční podmínku **P**";

do "snižuj hodnotu funkce **t** při invarianci podmínky **P**"

od (* **P** and (not **B**) → **R** ,

jinými slovy pravdivost podmínky **P** a nepravdivost podmínky **B** implikuje řešení **R** *)

Konečný tvar odvozeného programu s dokázánou správností je:

```
x:=X; y:=Y;  (* P *)
  do
    x>y   →→ x:=x-y   (* P and B1 *)
  | y>x   →→ y:=y-x   (* P and B2 *)
  od;
  z:=x;  (* splnění P and (not B) zaručuje dosažení výsledku *)
```

Tento zápis lze přepsat do podoby:

```
x:=X;
y:=Y;
while x<>y do
begin
  if x>y then x:=x-y;
  if y>x then y:=y-x
end;
z:=x
```

Na příklad pro součin se podívejte do opory.

Binární vyhledávání (Dijkstrova varianta)

Nechť je dáno pole celých čísel, pro které platí:

A[0] <= A[1] <= ... <= A[N-1] < A[N]

a nechť je dána hodnota klíče x, pro který platí *A[0]<=x<A[N].*

Nalezněme algoritmus, který ustaví pravdivost Booleovské proměnné SEARCH v případě, že x se rovná hodnotě některého prvku zadaného pole, tedy řešení ve tvaru:

***R** : SEARCH=(**Exist** i:(0<=i<N))[x=A[i]]* (1)

Vzhledem k seřazenosti pole skončí repetiční proces dosažením podmínky

$$\boldsymbol{R'} : A[i] <= x < A[i+1] \qquad (2)$$

Pak tedy platí

$$(\boldsymbol{R'} \text{ and } SEARCH=(x=A[i])) \rightarrow \boldsymbol{R} \qquad (3)$$

Invariantní relaci zavedeme pomocí proměnné j a vztahu (2) s cílem, aby

$$\boldsymbol{P} \text{ and } (j=i+1) \rightarrow \boldsymbol{R'}.$$

Pak tedy bude o invariantní relaci **P** platit:

$$\boldsymbol{P} : (A[i] <= x < A[j]) \text{ and } (0 <= i < j <= N) \qquad (4)$$

Cílem cyklu je zpracovat hodnoty i a j při platnosti invariance **P** tak, aby se dosáhlo platnosti relace j=i+1. Program bude mít tedy následující strukturu:

```
i,j:=0,N; (* Ustavení P *)
do
  j<>(i+1) →→ "úprava i a j při
                zachování platnosti
                invariance P "
od; (* P and (j=(i+1)) *)
SEARCH:= (x=A[i]); (* Ustavení řešení R *)
```

Připomeňme si, že

P *: (A[i]<=x<A[j]) and (0<=i<j<=N)* (4)

Nechť v důsledku úprav uvnitř cyklu nabude proměnná i nebo proměnná j nové hodnoty m. Nalezněme nejslabší počáteční podmínku pro mechanismus "i:=m" resp. "j:=m":

wp *("**i**:=m",**P**)=(A[**m**]<=x<A[j]) and (0<=**m**<j<=N) = **P** and (A[**m**]<=x) and (**m**<j)* (5)

Dosadíme za P

Dosadíme za P

wp *("**j**:=m",**P**)=(A[i]<=x<A[**m**]) and (0<=i<**m**<=N) = **P** and (x<A[**m**]) and (i<**m**)* (6)

Nechť je dána funkce **t**=j-i. Nalezněme podmínky, za nichž zvolený mechanismu zaručí konečnost aktivity cyklu.

$$\mathbf{wdec}(\text{"i:=m"},\mathbf{tau}>(j-i))=[tau>(j-m)]^{\mathbf{tau}}_{\mathbf{j-i}}=(j-i)>(j-m)=m>i \quad (7)$$

$$\mathbf{wdec}(\text{"j:=m"},\mathbf{tau}>(j-i))=[tau>(m-i)]^{\mathbf{tau}}_{\mathbf{j-i}}=(j-i)>(m-i)=j>m \quad (8)$$

jak vyplývá ze vztahů (5-8), musí nová hodnota m splňovat podmínku

$$i<m<j \quad (9)$$

Z hlediska symetrie je pro m vhodnou hodnota, která půlí interval (i,j), tedy

$$m:=half(i+j)$$

Vztah ***P*** *and (j<>(i+1)),* který platí po celou dobu cyklu, zajišťuje pro takové m platnost vztahu (6.3.9). Protože maximálně smí nabýt **i** hodnoty **j-2**, pak

$$i_{max}=half(i+j) = half(2*j-2) = j-1$$

a také

$$j_{min}=half(i+j) = half(2*i+2) = i+1$$

Celý program pak bude mít konečný tvar:

```
i,j:=0,N; (* ustavení podmínky P *)
do
   j<>(i+1)→→m:=half(i+j); (* platí i<m<j *)
    if
       A[m]<x→→i:=m
                    (* P and (A[m]<=x) and (m<j) *)
     | x<A[m]→→j:=m
                    (* P and (x<A[m]) and (i<m) *)
    fi
od;   (* P and j=(i+1) ustavuje R'*)
  SEARCH:=(x=A[i] (* ustavuje R *)
```

Děkuji za pozornost

Přeji Vám úspěšné zvládnutí zkoušky lAL a potom zasloužené

Veselé Vánoce