DOKAZOVANJE PRAVILNOSTI PROGRAMOV

Ivan Bratko

AKSIOMATSKA SEMANTIKA

- Pomen stavka, ali programa, definiramo z veljavnostjo vhodnega in izhodnega pogoja

Pomen je izražen s trditvijo (aksiomom):

Če pred stavkom velja P0, potem po izvedbi stavka velja P

- Vzemimo stavek S: $x := x + 1$

- Če pred S velja $x \geq 5$,

  potem po stavku S velja $x \geq 6$

$$x \geq 5$$

$$x := x + 1$$

$$x \geq 6$$

- Če pred S velja $x \leq 10$,

  potem po stavku S velja $x \leq 9$

- V splošnem:

  Za to, da po S velja pogoj p(x), mora pred S veljati p(x+1)

- Na primer p(x): $x \geq 6$

  potem mora pred S veljati p(x+1): $x+1 \geq 6$, oz. $x \geq 5$

AKSIOMATSKI POMEN

- Stavek $x := x + 1$ lahko torej razumemo kot aksiom:

  Če pred S velja p(x+1), potem po S velja p(x)

PRIMERI

Če velja: $x > y + 1$

pred stavkom: $y := y + 1$

potem velja po stavku: $x > y$

Če velja: $(x - 1)^2 > 10$

pred stavkom: $y := x - 1$

potem velja po stavku: $y^2 > 10$

PRIREDITVENI STAVEK y := f(x)

p(x, f(x, y))	vhodni pogoj, predpogoj
y := f(x, y)	stavek
p(x, y)	izhodni pogoj

NAJŠIBKEJŠI PREDPOGOJ

Če velja:	x > y + 1	Najšibkejši predpogoj
pred stavkom:	y := y + 1	Stavek
potem velja po stavku:	x > y	Izhodni pogoj

„Najšibkejši predpogoj“ je potreben in zadosten za to, da po stavku velja izhodni pogoj .

„Najšibkejši predpogoj“ je najšibkejši zadostni pogoj.

Angl.: Weakest precondition

x > y + 10 Zadostni predpogoj, ne najšibkejši

y := y + 1

x > y

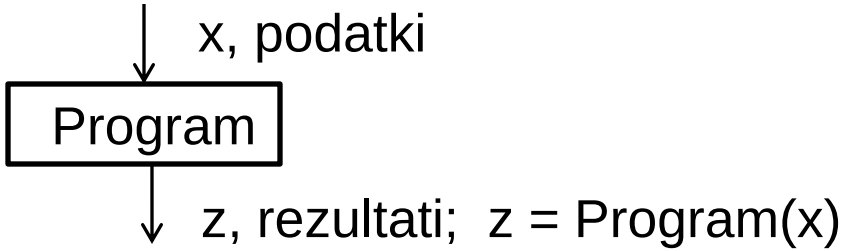
Predvsem nas zanimajo najšibkejši predpogoji

DOKAZOVANJE PRAVILNOSTI PROGRAMA

- Izhajamo iz aksiomatske semantike
- Aksiomatska semantika definira pomen stavka s tem, da definira najšibkejši predpogoj za stavek in izhodni pogoj
- Program P ustreza množici aksiomov A
- Dokažemo, da iz A sledi lastnost programa, ki nas zanima

SPECIFIKACIJA PROGRAMA

- Specifikacija pove, kaj naj bi program naredil



- Specifikacija z dvema predikatoma:

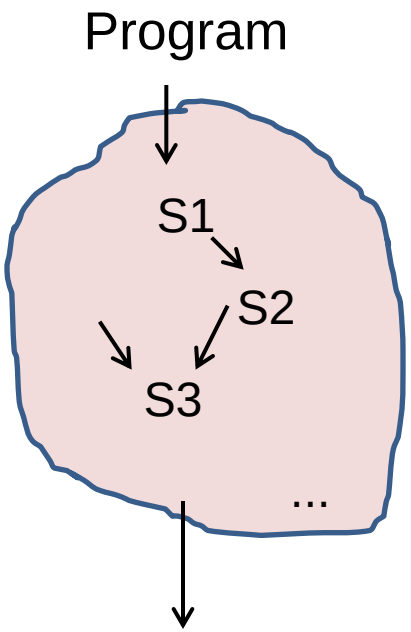
  vhodni predikat Precond

  izhodni predikat Postcond

- Če velja Precond(x), potem velja Postcond(x,z)

AKSIOMATSKA SEMANTIKA JEZIKA: ZAKAJ AKSIOMATSKA?

- Vsak stavek v programu gledamo kot aksiom
- Aksiomatska semantika pove, kakšnemu logičnemu aksiomu ustreza stavek v programu: stavek S_i deklarira aksiom A_i



Program ~ A1 & A2 & A3 & ....

DEFINICIJE PRAVILNOSTI PROGRAMA

- Naj bo P program specificiran z:

  Fi(x) vhodni predikat

  Psi(x,z) izhodni predikat

1. P ***se izteče*** glede na Fi, če za vsak x velja:

   če Fi(x), potem se izvajanje programa konča

2. P je ***parcialno pravilen*** glede na Fi in Psi, če za vsak x velja:

   če Fi(x) in P se konča za x, potem velja Psi(x, P(x))

3. P je ***totalno pravilen*** glede na Fi in Psi, če se (a) P izteče glede na Fi in (b) P je parcialno pravilen glede na Fi in Psi.

POSTOPEK

- Totalno pravilnost dokazujemo v dveh korakih:

  (a) dokažemo parcialno pravilnost

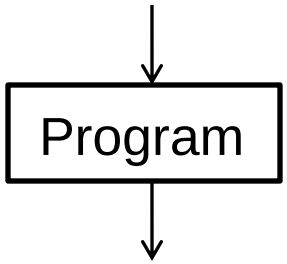
  (b) dokažemo, da se izteče

- V nadaljevanju se bomo ukvarjali povečini z dokazovanjem parcialne pravilnosti

DOKAZ PARCIALNE PRAVILNOSTI, V GROBEM

Precond (Podano)

WeakestPrecond (Določimo)



Postcond (Podano)

1. Za dani Program in Postcond določimo WeakestPrecond
2. Dokažemo izrek: Precond ==> WeakestPrecond

PRIMER DOKAZOV Z DOKAZOVALNIKOM

Primer 1

```
?- Program =
        [   begin,   a  :=  b,  x  :=  a,  end   ],


  verify( (b=5),        % Precondition
          Program,
          (x > 4) ).    % Postcondition
....
Prove: b=5   ==>   b > 4
```

Vaja: Izpelji gornji najšibkejši predpogoj b > 4

Najbolj praktično: Najšibkejši predpogoj izpeljemo od spodaj navzgor začenši z izstopnim pogojem x > 4

MAJHEN ALGORITMIČNI JEZIK

- Primer programa (celoštevilčno deljenje *a* deljeno z *b*):

```
[
  begin,
    rez := 0,
    ost := a,
      while (ost >= b) do
        [ begin, rez := rez + 1, ost := ost - b, end],
  end
 ]
```

SINTAKSA JEZIKA

- Program zapisan kot seznam v prologu, ukazi ločeni z vejicami
- Uvedemo spodnje operatorje - program postane sintaktično pravilen prologov objekt

```
:- op( 990, fx, invariant). % Announce invariant annotation for while
:- op( 989, xfx, while).     % A symbol of prog. language (annotated while)
:- op( 300, fy, not).
:- op( 980, fx, if).
:- op( 979, xfy, else).
:- op( 978, xfx, then).
:- op( 900, yfx, and).
:- op( 988, xfx, do).        % A symbol of prog. language
:- op( 700, xfx, :=).        % Assignment in prog. language
```

AKSIOMATSKA SEMANTIKA ZA TA JEZIK

- Pomen stavkov jezika je podan s predikatom wp/3:

  wp(Cond, Statement, Postcond)

  To pomeni: Cond je najšibkejši predpogoj, ki mora veljati pred stavkom Statement, za to da bo po izvršitvi Statement veljal PostCond

- Potrebujemo definicije za prireditveni stavek, zaporedje stavkov, if-then stavek, while stavek ...

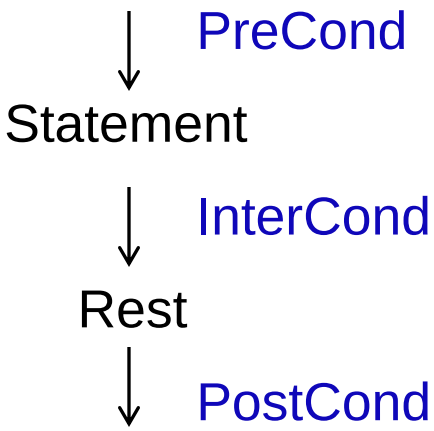
POMEN PRAZNEGA PROGRAMA

```
wp( Cond, [begin, end], Cond).     % Empty program

% Prazen program ne spremeni ničesar,
% zato je najšibkejši predpogoj kar enak izhodnemu pogoju
```

Sekvenčna kompozicija stavkov

```
wp( PreCond, [begin, Statement | Rest], PostCond)  :-
      wp( InterCond, [begin | Rest], PostCond),
      wp( PreCond, Statement, InterCond).
```

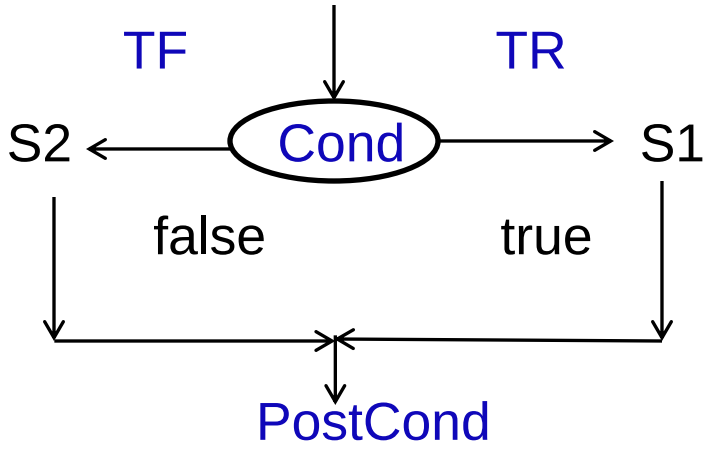


Najbolj praktično je, da pogoje določamo od spodaj navzgor!

if Cond then S1 else S2

```
wp( (Cond => TR) and (not Cond => TF),     % Weakest precondition
    (if Cond then S1 else S2),             % If-then statement
    PostCond)  :-                          % Post condition
  wp( TR, S1, PostCond),
  wp( TF, S2, PostCond).
```

(Cond ==> TR) and (~Cond ==> TF)



Predpogoj

(Cond => TR) and (not Cond => TF)

lahko ekvivalentno zapišemo tudi kot:

(Cond and TR) or (not Cond and TF),

PRIREDITVENI STAVEK

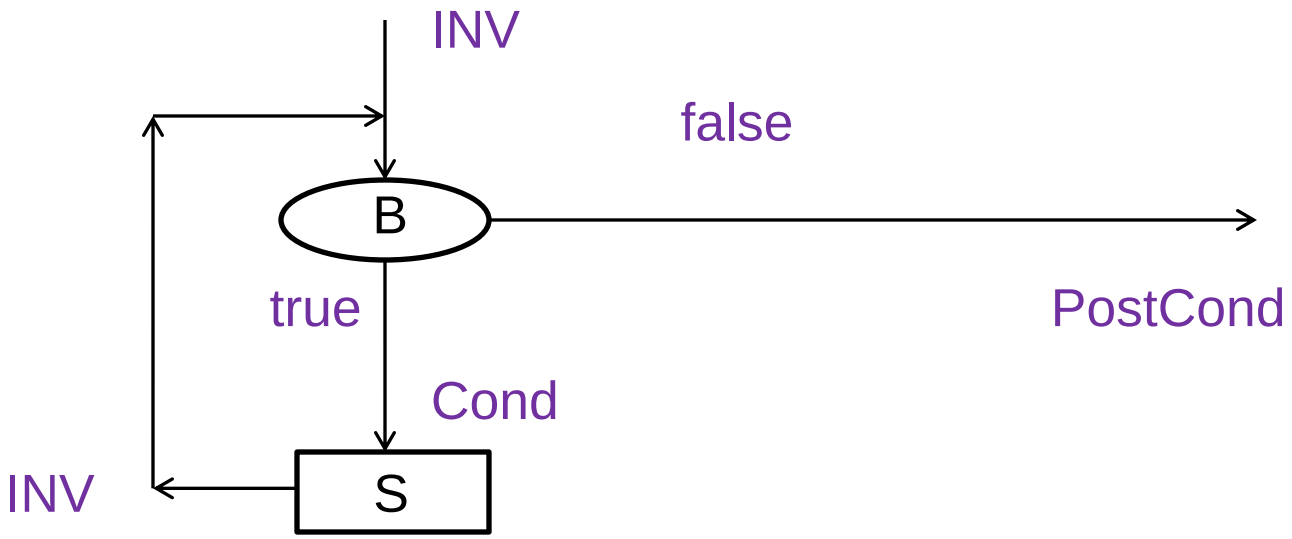
```
wp( PreCond, X := Y, PostCond)  :-        % Assignment X := Y
  replace( PostCond, X, Y, PreCond).      % Replace X with Y
```

OZNAČENI WHILE STAVEK

- While stavku dodamo „***invarianto***“, invariantni pogoj
- Invarianta je pogoj, ki *vedno* velja ob vstopu v zanko

```
wp( INV, invariant INV while B do S, PostCond)  :-   % Annotated while
  wp( Cond, S, INV),
  theorem( B and INV => Cond),                       % Within loop
  theorem( not B and INV => PostCond).               % Exit from loop
```

```
wp( INV, invariant INV while B do S, PostCond)  :-   % Annotated while
  wp( Cond, S, INV),
  theorem( B and INV => Cond),                       % Within loop
  theorem( not B and INV => PostCond).               % Exit from loop
```



ZANČNE INVARIANTE

- Kako jih dobimo?
- Z ugibanjem! (splošno sprejeti odgovor)
- Je pa še ena, malo znana možnost: avtomatsko z ILP (induktivno logično programiranje)
- Kdaj vemo, da smo invariante pravilno uganili?
- Ko se dokaz pravilnosti izteče

DOKAZOVALNIK PRAVILNOSTI
Glej program: dokazovalnik_programov.pl

```
verify( Precond, Statement, PostCond)  :-
  wp( Cond, Statement, PostCond),      % Weakest precondition
  theorem( Precond => Cond).

theorem( T)  :-
  simplify( T, T1),                    % Poenostavi T v T1, zelo komplicirano !
  write('Prove: '),
  write( T1), nl, nl.                  % Uporabnik naj dokaže, da je T1 izrek

simplify( T, T).                       % Ena možnost, da simplify ne naredi nič
                                       % S tem prevali vse delo na uporabnika
```

PRIMERI DOKAZOV Z DOKAZOVALNIKOM

Primer 1

```
?- Program =
        [   begin,   a  :=  b,  x  :=  a,  end   ],

   verify( (b=5),       % Precondition
           Program,
           (x > 4) ).    % Postcondition

....
Prove: b=5   ==>   b > 4
```

Vaja: Izpelji gornji najšibkejši predpogoj b > 4

Primer 2

```
?-  Program =  [ begin, t := a, a := b, b := t, end ],
    verify( (a = a0) and (b = b0),
            Program,
            (a = b0) and (b = a0) ).

...
Prove: a = a0  and  b = b0   ==>   b = b0  and  a = a0
```

Primer 3: Celoštevilčno deljenje

```
?-  Program  =                          % rez := a div b, ost := a mod b
       [  begin,
            rez := 0,  ost := a,
            invariant ( rez*b + ost = a )  and ( 0 =< ost )
              while (ost >= b) do
                    [ begin, rez := rez + 1, ost := ost - b, end],
          end  ],


 verify(
           ( a>0 and b>0),
           Program,
          ( rez*b + ost = a )  and  ( 0 =< ost )  and  ( ost < b ) ).
```

Primer 3, nad.

...

Prove: ost>=b and (rez*b+ost=a and 0=<ost) ==> (rez+1)*b+(ost-b)=a and 0=<ost-b

Prove: not (ost>=b) and (rez*b+ost=a and 0=<ost) ==> rez*b+ost=a and 0=<ost and ost<b

Prove: a>0 and b>0 ==> 0*b+a=a and 0=<a

Vaja:

1. Določi ročno gornje tri verifikacijske pogoje z uporabo aksiomatske semantike tega jezika

2. Dokaži, da so vsi trije pogoji izreki (veljajo za vse a, b, rez in ost)

Primer 4: Vsota celih števil od 1 do n

```
?-  Program =
        [ begin,
            s := 0,  i := 0,
            invariant  (i =< n) and (s = i * ( i + 1) / 2)
              while   i < n  do
                  [ begin,  i := i + 1,  s := s + i, end],
           end
        ] ,

 verify( (n >= 0),                  % Precondition
        Program,
        s = n * ( n + 1) / 2  ).    % Postcondition
```

Primer 4, nad.

Prove: i<n and (i=<n and s=i*(i+1)/2) ==> i+1=<n and s+(i+1)=(i+1)*(i+1+1)/2

Prove: not (i<n) and (i=<n and s=i*(i+1)/2) ==> s=n*(n+1)/2

Prove: n>=0 ==> 0=<n and 0=0*(0+1)/2

Vaja:

Ročno določi gornje tri verifikacijske pogoje za ta program in dokaži, da vedno veljajo

IZTEKANJE PROGRAMA

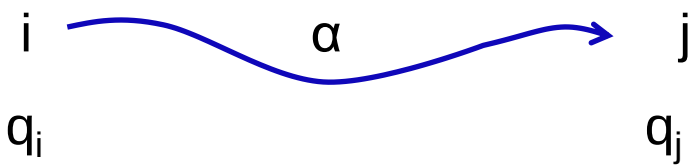
- V splošnem: za vsako zanko posebej pokažemo, da se izteče
- Program si predstavimo z diagramom poteka
- V diagram uvedemo “prerezne točke“ tako, da s temi točkami prekinemo vse zanke (v vsaki zanki je vsaj ena prerezna točka)
- Za vsako prerezno točko uvedemo novo spremenljivko u, tako da je u funkcija spremenljivk v programu:

  u = f(spremenljivke v programu)
- Za potrebe dokaza iztekanja mora biti zaloga vrednosti funkcije f „dobro utemeljena množica“ (well-funded set), tj.: mora biti urejena in nima neskončnih padajočih zaporedij

- Pokazati hočemo, da se vsakič, ko med izvajanjem programa ponovno pridemo v prerezno točko, u zmanjša. To pa se mora enkrat končati, ker u ne more padati v nedogled.
- To dokažemo za vsako pot med prereznimi točkami.

- Za to spet potrebujemo ustrezne invariante, ki jih „uganemo“
- Vendar te invariante niso nujno enake tistim za dokazovanje parcialne pravilnosti, saj služijo različnemu namenu
- Pri dokazovanju parcialne pravilnosti nam je cilj dokazati, da izstopni pogoj ob koncu izvajanja programa vedno velja
- Pri dokazovanju iztekanja nam je cilj dokazati, da vrednosti u stalno padajo

- Vzemimo pot α v diagramu poteka med prereznima točkama i in j:



- q_i, q_j sta invarianti. Najprej je treba dokazati, da te invariante vedno veljajo

- Za vsako pot od točke i do točke j konstruiramo pogoj poti R_α. R_α je konjunkcija pogojev na poti. Če izvajanje programa teče po tej poti, potem R_α zagotovo velja, sicer bi izvajanje teklo po drugi poti

- Z vsako pot α od i do j pokažemo, da vrednosti u padajo. Formalno:

$$q_i \wedge R_\alpha \Longrightarrow u_i(\text{začetek poti}) > u_j(\text{konec poti}))$$

- Natančneje: v tem verifikacijskem pogoju je treba upoštevati, da so se vrednosti spremenljivk med izvajanjem poti spremenile. Naj bo V množica spremenljivk v programu. Če izrazimo pogoje z vrednostmi spremenljivk v točki i, dobimo:

- $q_i(V) \wedge R_\alpha(V) \Longrightarrow u_i(V) > u_j(r_\alpha(V))$

  $r_\alpha(V)$ je kumulativna prireditev vrednosti spremenljivk na poti α:

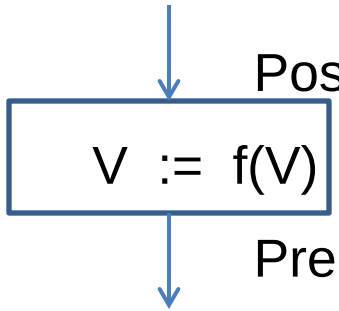
  $V := r_\alpha(V)$

KAKO KONSTRUIRAMO R_α in r_α

- Postopek inicializiramo v točki j, kjer postavimo:

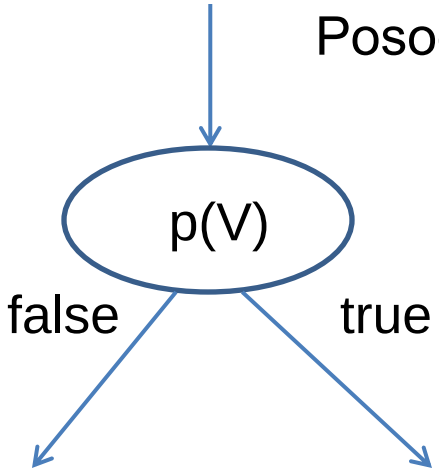
  R_α := true, r_α := V

Potem se pomikamo po poti α v vzvratni smeri od j do i in posodabljamo R_α in r_α po spodnjih pravilih:



Posodobljeno: $R_\alpha(f(V))$ in $r_\alpha(f(V))$

Pred pomikom nazaj: $R_\alpha(V)$ in $r_\alpha(V)$

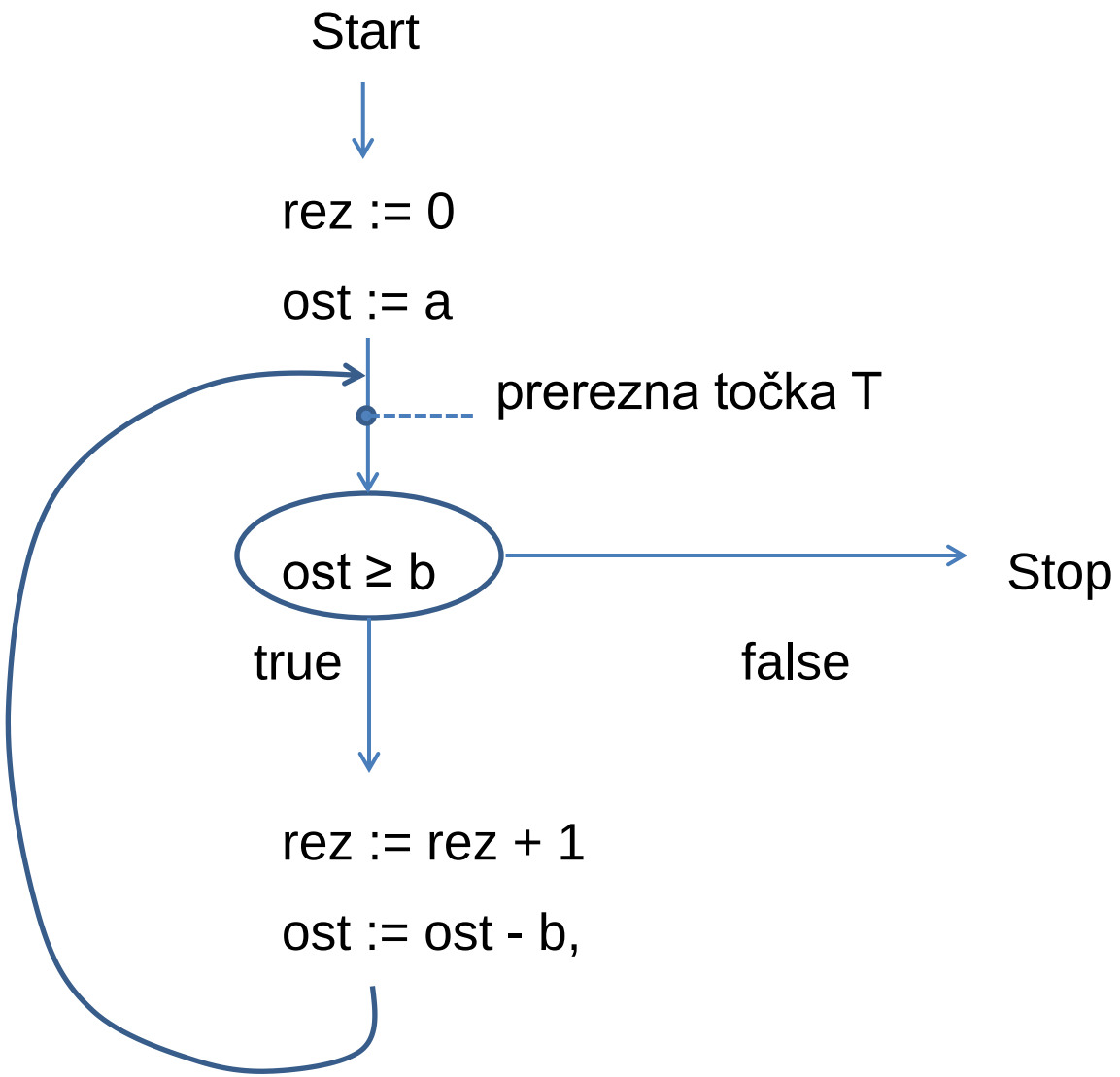


Posodobljeno: $p(V) \wedge R_\alpha(V)$ in $r_\alpha(V)$

Pred pomikom nazaj: $R_\alpha(V)$ in $r_\alpha(V)$

Analogno za vejo false, dodamo negirani pogoj ~p(V)

PRIMER: IZTEKANJE PROGRAMA 3, CELOŠTEVILČNO DELJENJE



IZTEKANJE: PROGRAM 3

- Za program 3 zadošča ena sama prerezna točka, T
- Izberemo funkcijo u za točko T:

  u = f(a, b, rez, ost) = ost
- Zaloga vrednosti u je množica nenegativnih celih števil, urejenih z relacijo <
- Invarianto v točki T uganemo takole: q(a,b,rez,ost): b > 0
- To invarianto trivialno dokažemo, saj je del vstopnega pogoja Fi, pri čemer se b sploh ne spreminja, zato velja v vsaki točki programa

- Za pot od T do T sta preslikava r in predikat R:

  r(a,b,rez,ost) = (a, b, rez +1, ost - b)

  R(a,b,rez,ost): ost ≥ b

- Pogoj iztekanja za našo zanko:

  q ∧ R ==> u(a,b,rez,ost) > u(r(a,b,rez,ost))

  (b > 0) ∧ (ost ≥ b) ==> ost > ost – b

- Ta pogoj velja za vse b in ost. (QED)