Principi programskih jezikov (PPJ/PPJSP) – izpit 13. sept. 2012

Navodila: Uporaba zapiskov, literature in elektronskih naprav ni dovoljena.
Točkovanje: Vse naloge so enakovredne.
Čas: 80 minut.
Ustni izpit: sreda, 19. sept. po razporedu, ki bo objavljen kasneje

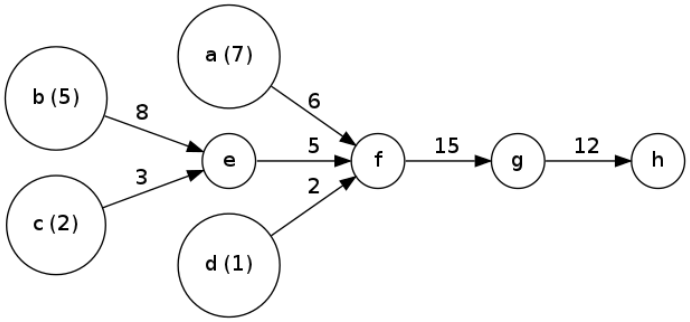
---

1. V nekem starem kanalizacijskem omrežju se pogosto zgodi, da imajo cevi premajhno kapaciteto glede na to, koliko odplak priteče skoznje. Kanalizacijsko omrežje ima obliko obrnjenega drevesa, torej je graf brez ciklov, kjer se poti lahko združijo, nikoli pa se ne razdružijo (glej sliko). Odplake tečejo od virov proti glavnemu odtoku, ko se več cevi združi v enem vozlišču, se pretoki odplak, ki tečejo po njih, seštejejo. Omrežje je podano z dejstvi oblike:

`source(Node, Outflow)`
vozlišče `Node` je izvor odplak, `Node` nima vhodnih povezav, tok odplak iz `Node` je `Outflow`

`pipe(Node1, Node2, Cap)`
med vozliščema `Node1` in `Node2` je cev s kapaciteto `Cap`.

Primer:



Ta mreža je v programu definirana z naslednjimi dejstvi:

```
source(a, 7).     source(b, 5).     source(c, 2).     source(d, 1).

pipe(a, f, 6).    pipe(b, e, 8).    pipe(c, e, 3).    pipe(d, f, 2).
pipe(e, f, 5).    pipe(f, g, 15).   pipe(g, h, 12).
```

(a) Zapiši vprašanje prologu, ki vrne seznam vseh izvornih vozlišč v grafu.

(b) Zapiši vprašanje prologu, ki izračuna, koliko je vseh izvornih vozlišč (za gornji primer 4).

(c) Napiši predikat `down( N1, N2)`, ki velja, če za dani vozlišči `N1` in `N2` odplake iz `N1` pritečejo v `N2`.
Npr.:

```
?-  down( b, g).
     yes
?-  down( b, d).
     no
```

(d) Spiši proceduro `inflow(Node, Inflow)`, ki za dano vozlišče `Node` izračuna, kolikšen dotok odplak naj bi pritekel vanjo, če bi bile kapacitete vseh cevi dovolj velike. Za vsako vozlišče, ki je podano kot vir odplak, torej `source( N, F)`, je dotok `Inflow = F`.
Na primer:

```
?- inflow(d, IF).
     IF = 7.
?- inflow( f, IF).
     IF  = 15
```

(e) Spiši predikat `pipeOK(Node1, Node2)`, ki drži takrat, ko ima cev med vozliščema `Node1` in `Node2` dovolj veliko kapaciteto za tok odplak, ki naj bi tekel po njej.
Na primer:

```
?- pipeOK(a, f).
    no
?- pipeOK(f, g).
    yes
```

2. Podan je naslednji predikat:

```
pf( X, P):-
    P #> 1,
    P #< X,
    X #= P * _,
    indomain(P).
```

(a) Kaj je domena logičnega programiranja z omejitvemi, ki ga uporablja ta predikat?

(b) Kaj odgovori prolog na naslednji vprašanji? Naštejte vse odgovore v pravilnem vrstnem redu!

```
?- pf( 12, P).
?- pf( 13, P).
```

(c) Kakšen je pomen predikata `pf/2`?

3. Izrazi pomen spodnjih stavkov v logiki (uporabi zapis v logiki ali pa ekvivalentni zapis v sintaksi prologa z izrazi oblike **exists(X, …)** ali **all(X, …)**):

(a) John likes music and beer.
(b) Tom caught a big fish.
(Uporabi predikata caught/2 in big/1.)
(c) Tom caught a fish, and so did Ann; Ann's fish was bigger than Tom's.
(Uporabi predikata caught/2 in bigger/2. Naj te ne moti, če se bo struktura logične formule tu precej razlikovala od strukture stavka v angleščini.)

Slovenski prevodi teh stavkov:
(a) John ima rad glabo in pivo.
(b) Tom je ujel veliko ribo.
(c) Tom je ujel ribo, prav tako Ana; Anina riba je bila večja od Tomove.

4. Dana je spodnja gramatika za gibanje robota v ravnini x-y. Robotovi premiki so omejeni na smeri x ali y, dolžina premika je ena enota (npr. 1 cm). Korak "east" pomeni premik za 1 cm v desno, premik "south" za 1 cm navzdol itd. Prostor robotovega gibanja ni omejen. Sintaktična fraza "path" pomeni dovoljene poti robota, podane s spodnjo gramatiko:

```
path  -->   move.
path  -->   move, path.
move  -->  [east].
move  -->  [north].
move  -->  [west].
move  -->  [south].
```

(a) Postavi prologu vprašanje, ali je `[east, west, south]` dovoljena pot za robota.

(b) Postavi prologu vprašanje, ki bo generiralo vse dovoljene poti dolge natanko 3 korake.

(c) Dopolni gornjo gramatiko z argumenti tako, da bo gramatika za dano začetno lokacijo robota `X0/Y0` generirala dovoljeno pot ter izračunala končno lokacijo robota po opravljeni poti.
Navodilo: neterminalu path dodaj dva argumenta takole:

```
path( X0/Y0, X/Y)
```

kjer je:

`X0/Y0` je začetna lokacija (koordinati x in y) robota v ravnini x-y
`X/Y` je končna lokacija robota po opravljeni poti

Npr.:

```
?- path( 1/2, Final, [ east, north, north, west, west], []).
    Final = 0/4
```

Ustrezno dodaj neterminalu `move` dodatni argument: `move( Dx/Dy)`, tako, da je `Dx` sprememba x-koordinate po koraku `move`, `Dy` pa sprememba y-koordinate po koraku. Npr. za korak "east" je sprememba 1/0, za korak "south" pa 0/(-1).

(d) Zdaj uvedemo v robotov svet ovire, podane s predikatom `obstacle( X/Y)`. Npr. `obstacle( 2/3)` pomeni, da robot ne sme stopiti na lokacijo 2/3. Dopolni gramatiko iz naloge (c) tako, da bo gramatika dovoljevala robotu le varne poti. Pri tem predpostavimo, da robot vedno začne iz dane varne lokacije.

(e) Zapiši vprašanje za gramatiko iz naloge (d), ki ugotovi, ali iz lokacije 1/1 lahko robot doseže lokacijo 3/3 po varni poti v ne več kot 6 korakih.

5. Podana je naslednja gramatika, ki definira sintakso in pomen preprostega programskega jezika za enostavni računalnik z vsega eno pomnilno celico – register R (podoben jezik, kot smo ga obravnavali kot primer na predavanjih).

```
program( (R0 --> OutTape) ) -->
[begin], instructs(( (R0,[]) --> (R,OutTape) )), [end].

instructs( ( (R0,Tape0) --> (R,Tape) ) ) -->
instr(( (R0,Tape0) --> (R,Tape) )).

instructs( ( (R0,Tape0) --> (R,Tape) ) ) -->
instr(( (R0,Tape0) --> (R1,Tape1) )),
instructs(( (R1,Tape1) --> (R,Tape) )).

instr( ( (R0,Tape) --> (R,Tape) ) ) -->
[ square ], { R is R0*R0}.

instr( ( (R0,Tape) --> (R,Tape) ) ) -->
[ add(X) ], { R is R0 + X}.

instr( ( (R,Tape0) --> (R,Tape) ) ) -->
[ print ], { conc( Tape0, [R], Tape)}.
```

(a) `?- program((1 --> X), [begin, print, end], []).`
(b) `?- program((1 --> X), [square, add(1), print], []).`
(c) `?- program((2 --> X), [begin, square, square, print, add(5), print, end], []).`
(d) Spodnje Prolog vprašanje ne deluje. Zakaj?

```
?-  program((1 --> [2]), [begin, add(X), end], []).
```

(e) Dodaj temu programskemu jeziku ukaz `set(X)`, ki nastavi vrednost registra na `X` (dodaj ukasz v gramatiko in mu določi pomen).

(f) Poenostavi dano gramatiko tako, da bo definirala le sintakso jezika in ne tudi pomena jezika.