Modalna logika

programski jezik

M. Petrinjak, I. Josip, R. Grabovac

18. lipnja 2023.

Sadržaj

Uvod

Ovaj rad napisan je za zadaću iz kolegija *Interpretacija programa* diplomskog studija *Matematika i računarstvo* na Matematičkom odsjeku PMF-a. Opisuje programski jezik namijenjen istraživanjima konačnih modela *Modalne logike* koji pruža mogućnost implementacije niza (semantički iznimno korisnih) algoritama od kojih su najvažniji oni za provjeru istinitosti ili valjanosti formule na definiranom modelu. Ipak, kako korisnost ovog programskog jezika ne bi bila previše apstraktna, prikazana je njegova upotreba u sinkronizaciji jednostavnog semafora, no više o tome bit će izloženo u nastavku. Nadalje, podrazumijevamo da je čitatelj upoznat barem s osnovama Modalne logike te zbog toga nismo navodili precizne matematičke definicije objekata s kojima smo radili, ali smo se trudili što češće citirati literaturu u kojoj se sve to može pronaći.

1. Dokumentacija

a. Ljuska

Pokretanjem programa u konzoli korisniku će biti predstavljeno okruženje za rad. Naredbe se unose jedna za drugom te završavaju točkazarezom. Dvostrukim pritiskom na tipku *Enter* blok naredbi se izvršava. Svrha dvostrukog prelaska u novi red je i definiranje funkcija koristeći više redaka.

Primjer 1.1: Primjer rada u linux komandnom prozoru. U zadnjoj napisanoj liniji pristupa se varijabli a i može se koristiti funkcija `_velik`. Ne bi bilo moguće bez dva uzastopna prelaska u novi red jer se naredbe ne bi obradile.

```
user@PC:~/IP$ python3 main.py
> int #a = 7;
> formula f = T;
> fun _velik(int #a) {
>       if(#a > 1000) vrati T;
        vrati F;
> }
>
> f = _velik(#a);
>  ispisi  << f;
```

◁

Moguće je napisati skriptu, odnosno cjelokupan program u datoteku ekstenzije *.mir*. Sve u njoj interpretirat će se kao niz naredbi. Isti efekt ima unošenje tekstualnog sadržaja u konzolu liniju po liniju. **Važno**: pokretanje *vanjskog* programa podrazumijeva da korisnik prije toga neće koristiti interaktivni način rada jer tada postoji mogućnost da se npr. deklarira varijabla istog imena kao u skripti koja se namjerava izvršiti što dovodi do pogreške, odnosno redeklaracije varijable što nije podržano.

Primjer 1.2: Ako je sadržaj datoteke *skripta.mir*

```
1 ispisi << 43770;
```

naredbom `napravi skripta.mir` ispisat će se

```
> napravi skripta.mir
43770
```

. ◁

b. Aritmetika

b.1. Numerička

Na numeričkim tipovima navedenim u e moguće je zbrajanje, oduzimanje, množenje i potenciranje. Osim toga, implementiran je operator ostatka pri cjelobrojnom dijeljenju, kao i samo cjelobrojno dijeljenje.

b.2. Logička

U svijetlu teorije opisane u [VP20, pogl. 1.2] implementirani su *logički veznici* i *modalni operatori*. Detaljnije u f.

c. Petlje

Ključne rječi `for`, `if`, `else` standardno služe kontroli izvođenja u programu. Glavna razlika je automatsko deklariranje varijable u `for` bloku jer često takve varijable jednostavno služe samo kao brojači i neće se koristiti dalje od bloka petlje u kojima se nalaze. No, ime se mora razlikovati od dotad korištenih. Korištenje `break`, `continue` izvan *for* i *foreach* petlje nije dozvoljeno. Ako slijedi samo jedna naredba, ne mora se naći unutar vitičastih zagrada.

Primjer 1.3: Operatori podržani unutar uvjetnih dijelova:

- `for: >, <, ++, -, +=, -=`
- `if: ==, <, >`

◁

Primjer 1.4: Primjer toka programa.

```
for(#i=0; #i<100; #i++) {
    if(#i == 8) continue;   // nemoj ispisati 8
    if(#i < 20) ispisi << #i;
    if(#i == 10) break;     // stani na 10
}
```

◁

Postoje dva oblika `foreach` petlje. Odnose se na model koji se trenutno koristi (o načinu korištenja bit će više u g.1) , a ako nije definiran pojavljuje se semantička greška.
Iteracija po svim svjetovima modela:

```
foreach @world {
    // @world je u svakoj iteraciji drugi svijet
}
```

Iteracija po svim propozicionalnim varijablama modela:

```
foreach $var {
    // $var je u svakoj iteraciji druga p.var.
}
```

d. Funkcije

Funkcije se definiraju početnom riječi *fun* iza koje slijedi naziv s početnim znakom „_”, popis argumenata unutar zatvorenih oblih zagrada i tijelo funkcije. Poziv se obavlja navođenjem imena i argumenata odgovarajućeg tipa. Povratna vrijednost (ako postoji) mora biti `T` ili `F`, a to je potaknuto činjenicom da nas u logici općenito zanima samo je li nešto istina ili laž. Nadalje, svaka funkcija ima svoju lokalnu, odnosno s glavnim programom disjunktnu memoriju.

```
// pretpostavljamo da smo iznad definirali model na kojem postoji
   svijet @svijet
fun _foo(formula f, int #n) {

    formula t = f;
    for(#i = 0; #i < #n; #i++) {
        f = []t;
        t = f;
    }
    if(f ? @svijet) vrati T;
    else vrati F;
}
// poziv
formula rez = T;
int #n = 2;
rez = _foo(rez, #n);
ispisi  << rez << nr; // nr = prijelaz u novi red
```

e. Tipovi podataka

e.1. Cijeli (int) i prirodni (nat) brojevi

Predviđen je standardni rad s cijelim i prirodnim brojevima. **Imena počinju znakom #**. Jedine moguće greške su pridruživanje negativne vrijednosti prirodnom broju, i negativne potencije. Operatori opisani u f.1.

```
nat #a = -1;     // greška
int #b = 2^-3;   // greška
```

e.2. Formule

Formule se grade od propozicionalnih varijabli i drugih formula. **Imena formula počinju malim slovom**, a **prop. varijabli znakom $**. Može ih se deklarirati pomoću:

- prop. varijabli:

```
formula f = $p1;
```

- logičkih koinstanti:

```
formula g = T;
formula h = F;
```

- unarnih logičkih operatora:

```
formula e = []f;
```

- binarnih veznika:

```
formula i = (e | g);
```

e.3. Konačni modeli

Modeli su definirani u [VP20, pogl. 1.3]
Imena modela počinju velikim slovom, imena svijetova s `@`, a prop. varijable s `$`.

Primjer 1.5: Model naziva M sa svjetovima `svijet1, svijet2, svijet3` i prop. varijablama `pada_kisa, ulice_su_mokre, prolazi_cisterna` deklarira se kao

```
koristi M {
    @svijet1, @svijet2, @svijet3;
    $pada_kisa, $ulice_su_mokre, $prolazi_cisterna
};
```

◁

Tom naredbom su deklarirane i varijeble posebnog tipa — *svijet* i *prop. varijabla*. Svjetovi predstavljaju nosač modela, a definirane propozicionalne varijable mogu se shvatiti kao činjenice koje su relevantne za zaključivanja u definiranom modelu. Kako bi se uspostavila korelacija između svjetova i istinitosti, odnosno definirale relacije dostiživosti i forsiranja, potrebno je učitati model iz datoteke. Za upute vidjeti g.1.

f. Operatori

f.1. Aritmetički: `+, -, *, ^, ÷, %`

Operandi mogu biti konkretni brojevi i varijable aritmetičkih tipova e.1.

f.2. Relacijski

Navedeni u 1.3.

f.3. Modificiranje valuacije

U skladu s definicijom 1.4 [VP20] napravljeni su operatori *forsiranja*. Svaki od sljedećih ima svoj alias pored sebe. Moguće je s desne strane istovremeno navesti više varijabli u vitičastim zagradama kao što je prikazano u primjerima.

- `|=, ||-`
  Forsira istinitost propozicionalne varijable na nekom svijetu.

```
@svijet |= $pvar;
@svijet |= {$p1, $p2, $p3};
```

- `|~ , ||~`
  Ekvivalentno forsiranju negacije propozicionalne varijable u izrazu.

```
@svijet |~ $pvar;
@svijet |~ {$p1, $p2, $p3};
```

- `=|, -||`
  Ne postoji u teoriji, ali je koristan ako je potrebno opisivati vezu jedne prop. varijable i više svjetova.

```
$p -|| {@w1, @w2};
$p -|| @w3;
```

- `~|, ~||`
  Analogno prethodnom.

```
$p ~| {@w1, @w2};
$p ~| @w3;
```

f.4. Provjera istinitosti: `?`

Daje informaciju o **istinitosti formule** na nekom **svijetu** iz **modela** koji se trenutno **koristi** (tzv. istinitost na točkovnom modelu). Ako se ne nađe unutar `if` uvjeta, ispisuje se tvrdnja u tekstualnom obliku.

Primjer 1.6: Upotreba:

```
formula f = ($p -> []$q);
if(f ? @svijet1) ispisi << 1;
```

◁

f.5. Pridruživanje: `=`

Varijabli s lijeve strane pridružuje se vrijednost varijable ili literala s desne strane. Tipovi moraju odgovarati, inače dolazi do semantičke greške. Kod deklaracije varijabli mora biti pridružena **početna vrijednost**.

```
int n;      // ne valja
int n = 0;  // ok
```

f.6. Logički i modalni

Ponovo je vodilja sadržaj [VP20, pogl. 1.2].

U idućim primjerima su g i h neke prethodno deklarirane formule. Kod binarnih veznika su potrebne zagrade oko izraza, dok kod unarnih nisu.

- konjunkcija: `&`

```
formula f = (g & h);
```

- disjunkcija: `|`

```
formula f = (g | h);
```

- kondicional: `->`

```
formula f = (g -> h);
```

- bikondicional: `<->`

```
formula f = (g <-> h);
```

- negacija: `~`

```
formula f = ~g;
```

- box: `[]`

```
formula f = []g;
```

- diamond: `<>`

```
formula f = <>g;
```

g. Datoteke

g.1. Učitavanje modela

Unos iz datoteke se obavlja na sljedeći način:

```
// prije toga deklaracija
unesi << "relacijska_dat.mir" << 'val_datoteka.mir';
```

Obavezna je upotreba navodnika oko naziva datoteke. Mogu biti jednostruki ili dvostruki, ali se moraju poklapati. U relacijskoj datoteci mora prvo pisati **„rel"** (bitno je da su to prva 3 slova, smije pisati i npr. „relacija"), analogno za valuacijsku datoteku gdje mora pisati **„val"**.

U valuacijskoj su datoteci **svjetovi** u prvom stupcu, a **prop. varijable** u prvom retku. U relacijskoj su u prvom stupcu i prvom retku **svjetovi**.

Primjer 1.7: Primjer relacijske i valuacijske datoteke:

```
rel          svijet    world     el_mundo     za_warudo
svijet         1         0          0            1
world          0         0          1            0
el_mundo       0         1          0            1
za_warudo      0         1          0            1
```

```
valuacija    pada_kisa    ulice_su_mokre    prolazi_cisterna
svijet            1                1                 0
za_warudo         1                1                 1
el_mundo          0                1                 0
world             0                1                 1
```

◁

Oznaka za istinitost relacije odnosno forsiranja smije biti u raznim oblicima: bilo koji string kojem je prvi znak 'T', '1', 'Y', 'I', 'D' ili 'O' se interpretira kao istina, a za 'F', '0', 'N', 'L', 'N' i 'X' je neistina (znakovi ne ovise o velikim i malim slovima).

g.2. Spremanje modela

Kako bi podaci mogli biti ponovo učitani u kasnijem pokretanju (pogotovo zbog toga što se gore definiranim operatorima može promijeniti (ne)istinitost neke prop. varijable na svijetu), postoji ispis modela u *.mir* datoteku.

Primjer 1.8: Ako se model za pohranu zove `M`,

```
// zadnja koristi naredba mora biti
// koristi M{...};
spremi M;
```

kreira datoteke s nazivom *rel_datM.mir* i *val_datM.mir*. Ako želimo spremiti model koji prethodno nismo deklarirali, javlja se semantička greška. ◁

g.3. Skripte

Vidjeti 1.2.

h. Komentari

Jednolinijski komentari počinju s dvije kose crte.

```
// Ovo baš ništa ne znači.
```

i. Ispis u konzolu

Podržani tipovi varijabli, svjetovi, numerički literali i znak za prijelaz u novi red (nr) se mogu ispisivati pomoću ispiši « <ime_varijable>; što znatno olakšava testiranje ispravnosti programa.

```
formula a_1 = ($pada_kisa -> $ulice_su_mokre);
koristi M {
@svijet, @world, @za_warudo, @el_mundo;
$pada_kisa, $ulice_su_mokre, $prolazi_cisterna };
unesi << "relacijska_dat.mir" << 'val_datoteka.mir';

ispisi << M << nr;
ispisi << f << nr;
ispisi << @world; // ispisuje sljedbenike i p.var. koje forsira
// ispis:
// M: { @svijet @el_mundo @world @za_warudo; $prolazi_cisterna
   $ulice_su_mokre $pada_kisa }
// ($pada_kisa->$ulice_su_mokre)
// {@svijet; $pada_kisa}
```

2. Primjena

a. Iskazna logika

Za proizvoljnu formulu prop. logike je moguće provjeriti je li valjana. Potrebno je napraviti model s jednim svijetom i praznom relacijom. Sve prop. varijable koje se javljaju u formuli treba deklarirati kod deklaracije modela.

Primjer 2.1: Krnji model.

```
koristi M {@s; $p1, $p2, $p3};
```

◁

Primjer 2.2: Isječak za provjeru valjanosti formule logike sudova

```
// iteriranje po binarnim zapisima svih valuacija
// #m = broj prop.varijabli
// k5 = formula čija nas istinitost zanima
for (#i=0; #i<2^#m; #i++) {

    #temp = #i;
     ispisi  << #i;

    // kreiranje valuacije
    foreach $v {

        #bin = #temp % 2;

         ispisi  << #bin;
        if ( #bin == 1) @s |= $v;
        else @s |~ $v;

        #temp = #bin \ 2;

    }
    // provjera istinitosti
    if(k5 ? @s) continue;
    else {
        spremi M;   // kombinacija koja obara istinitost
        break;
    }
}
```

◁

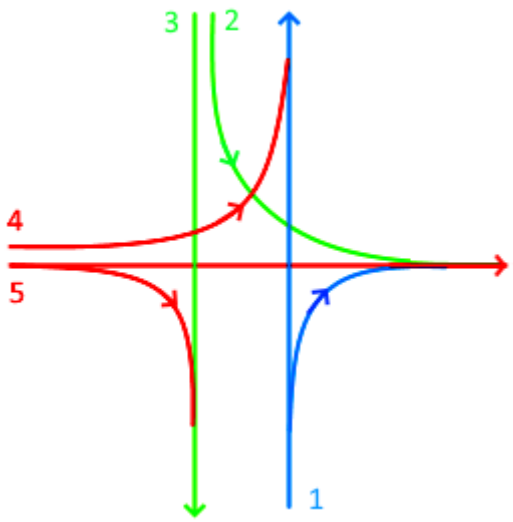
b. Istinitost formula

Primjer 2.3: Moguća implementacija provjere istinitosti modalne formule na modelu.

```
fun _globalnoIstinita(formula f) {
    formula neg = ~f;
    foreach @svijet {
        if (neg ? @svijet) vrati F;
    }
    vrati T;
}
// deklaracija modela
koristi M {@w1, @w2, @w3; $p1, $p2, $p3};
unesi << "rel.mir" << "val.mir";
formula rez = F;
// f neka formula već definirana formula
rez = _globalnoIstinita(f);
 ispisi  << rez << nr;
```

◁

b.1. Problem semafora



Slika 2.1.: Križanje

Na slici 2.1 je model jednostavnog križanja. Prijevod u *Kripkeov* okvir je relacija u datoteci

```
rel  t1  t2  t3  t4  t5
t1   0   1   0   1   1
t2   1   0   0   1   1
t3   0   0   0   1   1
t4   1   1   1   0   0
t5   1   1   1   0   0
```

koja predstavlja relaciju S *smetanja*. Elementi *nosača* su upravo tokovi na slici, označeni s t1 do t5. Dva toka su u relaciji ako se njihovi putevi križaju na prometno neprihvatljiv način. Cilj je sinkronizacija semafora tako da promet nesmetano teče. U prijedlogu rješenja koje želimo provjeriti su tri faze semaforskog ciklusa. U prvoj fazi je zeleno za tokove 1 i 3. U drugoj za 2 i 3, itd.

```
val  z1  z2  z3
t1   1   0   0
t2   0   1   0
t3   1   1   0
t4   0   0   1
t5   0   0   1
```

Prop. varijabla `$zi` je istinita na svijetu `@ti` akko je toku `i` upaljeno zeleno svjetlo. U prometu neće doći do kolizije ako je formula

```
// za svaki svijet mora vrijediti da ako je u istom ciklusu zeleno,
   ostalim svijetovima koji mu smetaju mora biti crveno
($z1 -> []~$z1) & ($z2 -> []~$z2) & ($z3 -> []~$z3)
```

istinita na modelu, odnosno istinita na svakom svijetu modela

```
koristi K { @t1, @t2, @t3, @t4, @t5; $z1, $z2, $z3 };
unesi << "krizanje_rel.mir" << "krizanje_val.mir";
```

Kod primjera je u datoteci *krizanje_prog.mir*. Pokretanjem skripte će se ispisati T jer je sinkronizacija ispravna. Promjenom sadržaja u valuacijskoj datoteci može se doći do lošeg rješenja za križanje. U tom slučaju je rezultat skripte `F`.

b.2. Broj ciklusa

Dizajnera može zanimati minimalan broj ciklusa potreban da na nekom raskrižju postoji mogućnost nesmetanog prometa. Može se koristiti funkcija

```
fun _dovoljnoCiklusa(formula f) {

    int #m = 0;
    int #n = 0;

    // broj svijetova
    foreach @svijet {
        #n = #n + 1;
    }
    // broj prop. varijabli
    foreach $pvar {
        #m = #m +1;
    }

    int #bin = 0;
    int #temp = 0;
```

```
    formula rez = T;

    // iteriranje po binarnim zapisima svih valuacija
    for (#i=0; #i<2^(m*n); #i++) {

        #temp = i;

        // kreiranje valuacije
        foreach $var {
            foreach @svijet {

                #bin = temp % 2;

                if ( #bin == 1) $pvar =| @svijet;
                else $pvar ~| @svijet;

                #temp = #bin / 2;
            }
        }
        // testiranje
        rez = _globalnoIstinita(f);
        if ( rez ? @svijet ) {
            vrati T;
            spremi M;   // zapisuje valuaciju u datoteku da vidimo kako
    sinkronizirati semafore
        }
    }
    vrati F;
}
```

U rješenju problema danom za primjer u b.1 su tri ciklusa. Kad bismo kreirali model s 2 prop. varijable bi funkcija vratila F.

c. Valjanost formula

```
fun _valjana(formula f) {

    int #m = 0;
    int #n = 0;

    // broj svijetova
    foreach @svijet {
        #n = #n + 1;
    }
    // broj prop. varijabli
    foreach $pvar {
        #m = #m +1;
```

```
13      }
14
15      int #bin = 0;
16      int #temp = 0;
17      formula rez = T;
18      formula neg = T;
19
20      // iteriranje po binarnim zapisima svih valuacija
21      for (#i=0; #i<2^(m*n); #i++) {
22
23          #temp = i;
24
25          // kreiranje valuacije
26          foreach $var {
27              foreach @svijet {
28
29                  #bin = temp % 2;
30
31                  if ( #bin == 1) $pvar =| @svijet;
32                  else $pvar ~| @svijet;
33
34                  #temp = #temp / 2;
35              }
36          }
37          // testiranje
38          rez = _globalnoIstinita(f);
39          neg = ~rez;
40          if ( neg ? @svijet ) vrati F;
41      }
42      vrati T;
43  }
```

d. Mogućnost daljnjeg razvoja

Uočimo kako ovim programskim jezikom možemo provjeriti valjanost formule na (konačnom) modelu. Postoji mnoštvo specifičnih formula modalne logike čija je valjanost na modelu ekvivalentna s nekim svojstvom njegove relacije dostiživosti. Primjerice, valjanost formule $(P \rightarrow \Box(\Diamond P))$ ekvivalentna je s činjenicom da je relacija dostiživosti simetrična i slično. Dakle, jednostavno bismo pozvali funkciju za provjeru valjanosti na toj formuli i dobili rezultat. Nadalje, u jeziku postoji ugrađena funkcija *optimiziraj* (zasad za korisnika neupotrebljiva) koja svaku formulu pretvara u njoj ekvivalentnu, ali samo s veznicima $\neg, \Box$ i $\rightarrow$. Korisnost te funkcije leži u provjeravanju ispravnosti dokaza u modalnoj logici jer se postojanjem samo 3 veznika složenost tog problema znatno smanjuje. Ideja je za svaku formulu u dokazu ispitati je li shema aksioma A1, A2, A3 (iz sistema RS) ili pak aksioma distributivnosti. Ako tome nije slučaj, onda je potrebno ispitati je li nastala primjenom nekog od pravila izvoda - *modus ponens* ili nužnosti. Već smo planirali daljnji razvoj ovog programskog jezika u svrhu rješavanja navedenog problema pa je implementirana funkcija *unos_dokaza* koja učitava dokaz iz datoteke kojeg je korisnik prethodno napisao. Ipak, kako bi se problem u potpunosti riješio potrebno je nešto više vremena, tako da, stay tuned...

Bibliografija

[VP20] Mladen Vuković i Tin Perkov. „Algebarska semantika modalne logike”. 2020.