SAFETY MODEL CHECKER ZA BULOVSKE PROGRAME

Josip Vujčić
jvujcic@gmail.com

16.6.2009

Sadržaj:

1. Uvod

Model checking je naziv za automatsku provjeru jednostavnih modela nekog složenijeg sustava. Najčešće se prave modeli hardvera ili softvera i zatim se uz pomoć model cheking alata provjerava da li ti modeli poštuju dane specifikacije. Da bi ostvarili algoritamsku provjeru modela, sam model i dane specifikacije se moraju formalno definirati. To se radi u jeziku matematičke logike. Tako se dani problem svodi na provjeravanje da li neka struktura (u jeziku logike) zadovoljava određenu logičku formulu.

Naš program će se baviti verifikacijom softvera. Verifikacijska svojstva možemo podijeliti u dvije grupe:

- ***Safety svojstva*** (ništa se "loše" ne dogodi"): ne postoje ulazni parametri koji bi doveli do greške u radu programa.
- ***Liveness svojstva*** (nešto "dobro" će se dogoditi): za svaki skup ulaznih parametara program će dati dobar rezultat.

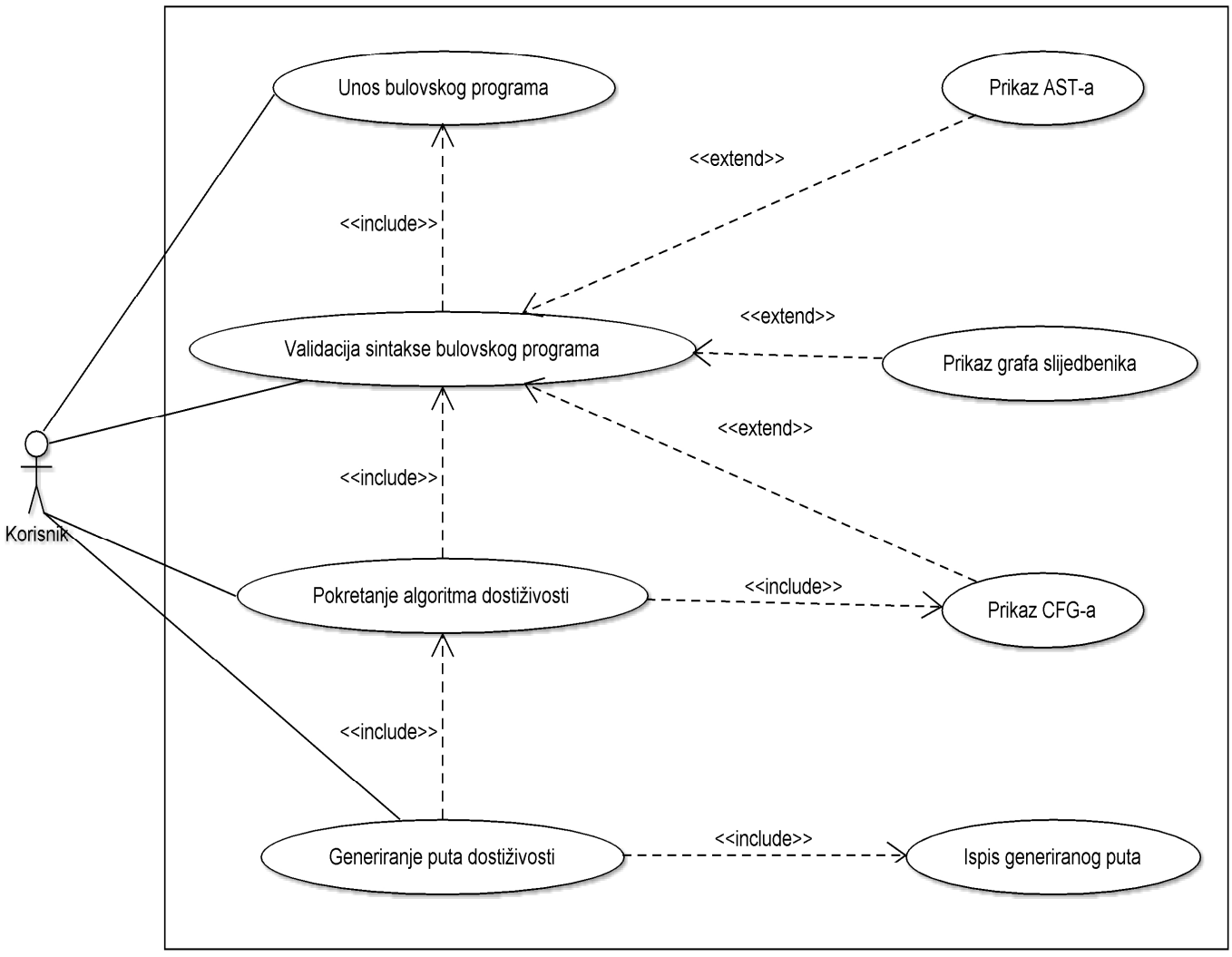
U praksi je lakše provjeriti safety svojstva jer se su ona ekvivalentna provjeravanju da li će se izvršiti određena linija koda. Na taj način radi i naš verifikacijski softver, tj. on provjerava da li je neka naredba u programu dostiživa ili ne. Najveći problem kod verifikacije nekog programa (koji je zapisan u nekom programskom jeziku) jest eksplozija stanja. Teoretski, nikad ne možemo provjeriti sve slučajeve ali možemo naći model koji dobro aproksimira zadani program i napraviti verifikaciju nad njim. Jedan od načina na koji se to može napraviti je da program (zapisan npr. u C programskom jeziku) aproksimiramo bulovskim programom. Bulovski program je jako pojednostavljeni C program. Sve varijable su tipa *bool* i upravo zbog toga ne postoji eksplozija stanja. Prednost bulovskih programa je što čuvaju slijed izvršavanja programa (jer se mogu opisat petlje, pozivi funkcija, rekurzije i grananja). Za potpun verifikacijski softver potreban je program koji svaki kod napisan u C programskom jeziku prebacuje u odgovarajući bulovski kod te program koji provjerava da li tako dobiven bulovski program poštuje dana svojstva. Program koji je izrađen (pod nazivom "Safety model checker za bulovske programe") se upravo bavi ovim drugim dijelom, tj. provjerava safety svojstva bulovskog programa.

U ostatku dokumenta će biti prezentiran dizajn programa i njegova upotreba. Sam algoritam korišten u analiziranju bulovskog programa ovdje nećemo opisivati. Opis i analiza algoritma se može pronaći u [1].

2. UML dijagrami

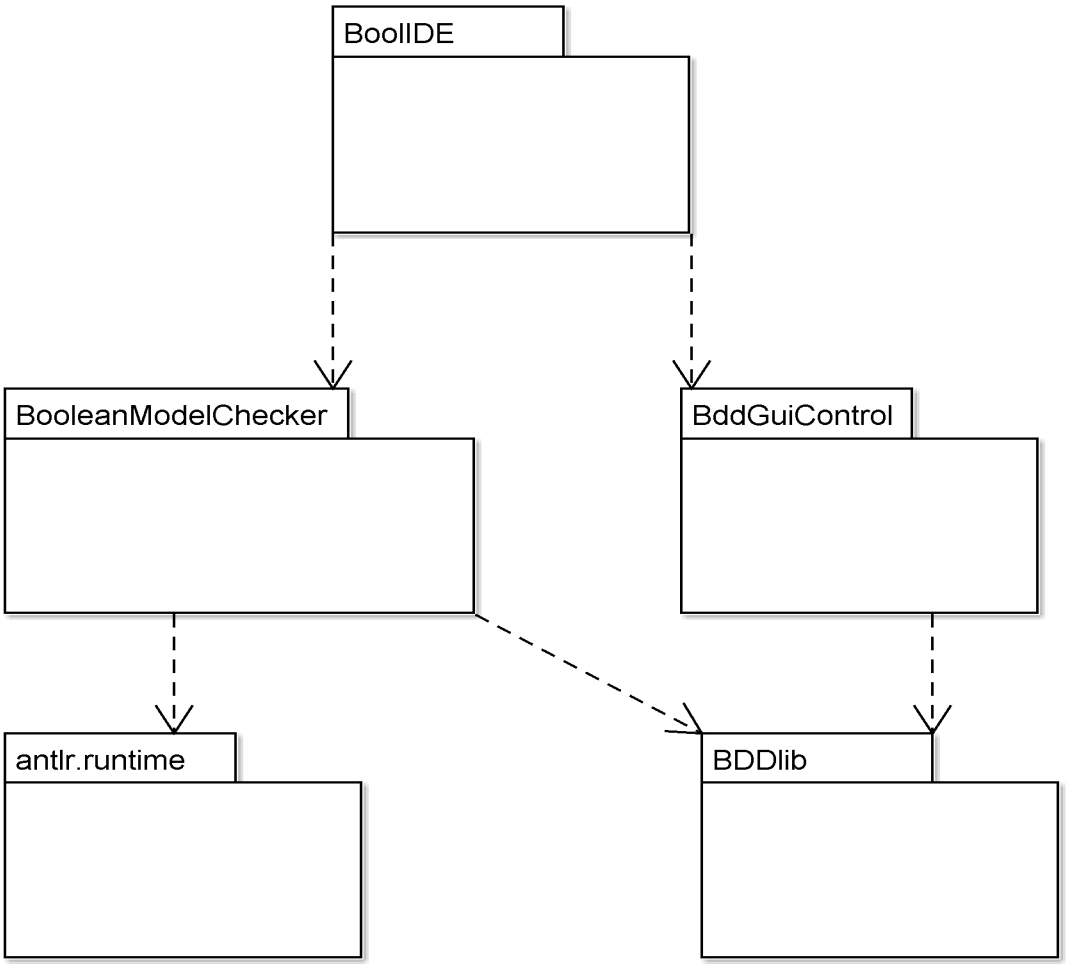
2.1. Use Case dijagram

Prikaz sustava sa stajališta korisnika dano je slijedećim dijagramom.



2.2. Package dijagram

Grafičko sučelje programa je odvojeno od algoritma. Također, glavni dijelovi programa (sintaktička analiza bulovskog koda i izgradnja AST-a, manipuliranje BDD-ovima, algoritam za verifikaciju) su napravljeni kao zasebne biblioteke. Relacije između biblioteka su dane slijedećim dijagramom:

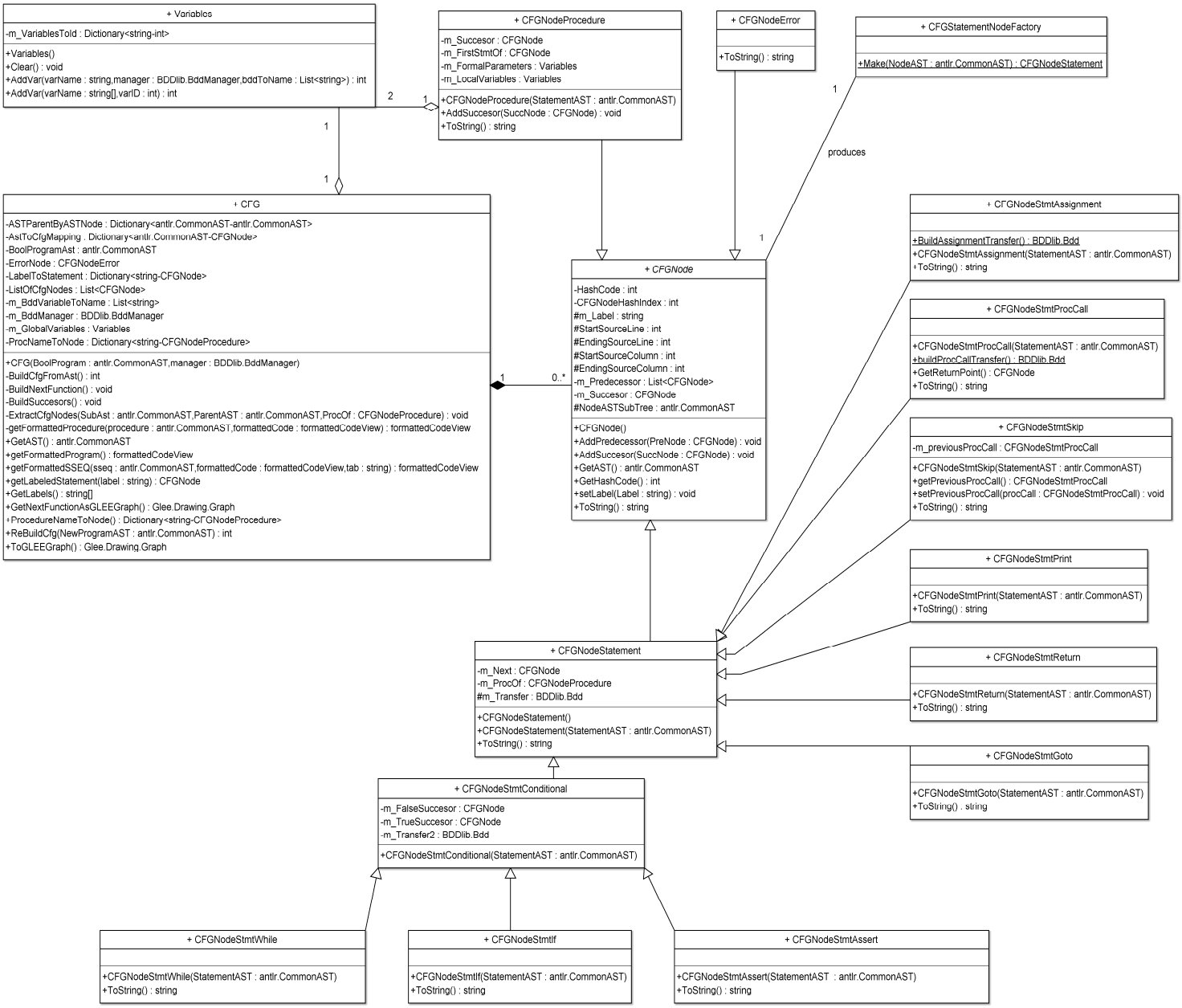


Sada ćemo dati kratki opis svakog paketa:

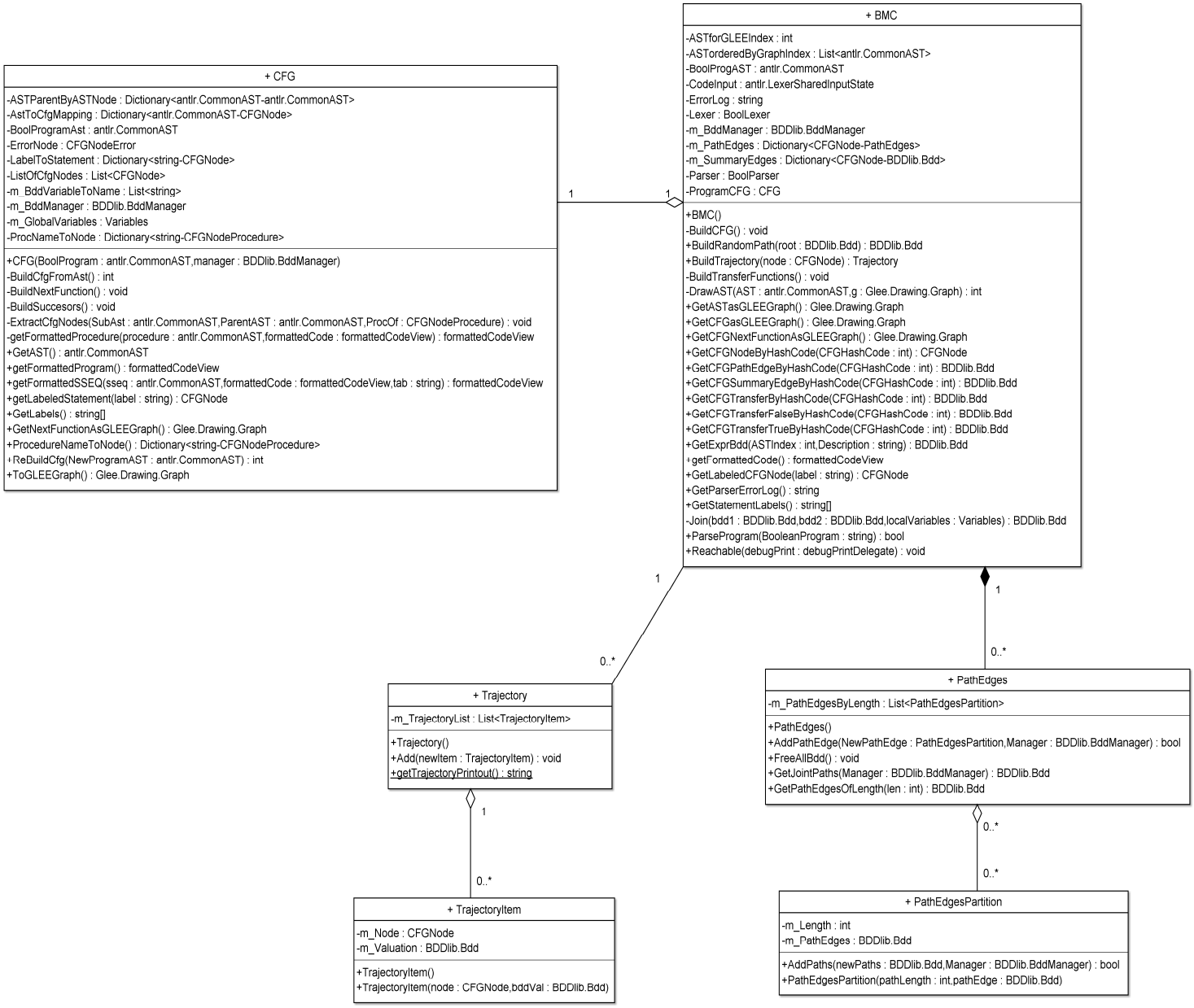
- **BoolIDE**: Grafičko sučelje programa. Ne sadrži nikakve algoritme za analizu bulovskog programa.
- **BooleanModelChecker**: (DLL) Sadrži CFG klasu i osnovne algoritme za verifikaciju safety svojstava bulovskih programa.
- **antlr**.**runtime**: (DLL) Biblioteka koja služi za sintaktičku analizu bulovskog programa i izgradnju pripadajućeg AST-a.
- **BDDlib**: (DLL) Služi za kreiranje BDD-ova koji se intenzivno koriste u osnovnom algoritmu te također sadrži algoritme za njihovu manipulaciju.
- **BddGuiControl**: (DLL) Biblioteka za grafički prikaz BDD-ova.

2.3. Class dijagrami

Sada ćemo ukratko prikazati osnovne klase potrebne za izvođenje algoritma. Za lakše shvaćanje klasa ćemo dati kratki prikaz samog algoritma. Prvo se uneseni bulovski program pomoću antlr.runtime biblioteke sintaktički analizira i kreira se AST reprezentacija samog programa. AST klasu nećemo prezentirati jer je ona kreirana vanjskim alatom (ANTLR). Pomoću izgrađenog AST-a se izgradi CFG bulovskog programa. CFG možemo shvatiti kao apstraktnu reprezentaciju izvođenja programa. Sam algoritam se izvodi tako da se šećemo po CFG-u i postepeno kreiramo sve moguće valuacije programa (njih kompaktno spremamo u obliku BDD-a). Nakon što smo izgradili BDD-ove, ponovno šetnjom po CFG-u možemo odrediti trajektoriju(put) do određene naredbe u bulovskom programu (ako ona postoji). Slijedeći dijagram prikazuje klase potrebne za izgradnju CFG:



Slijedeći dijagram prikazuje klase koje služe pri izvođenju samog algoritma:

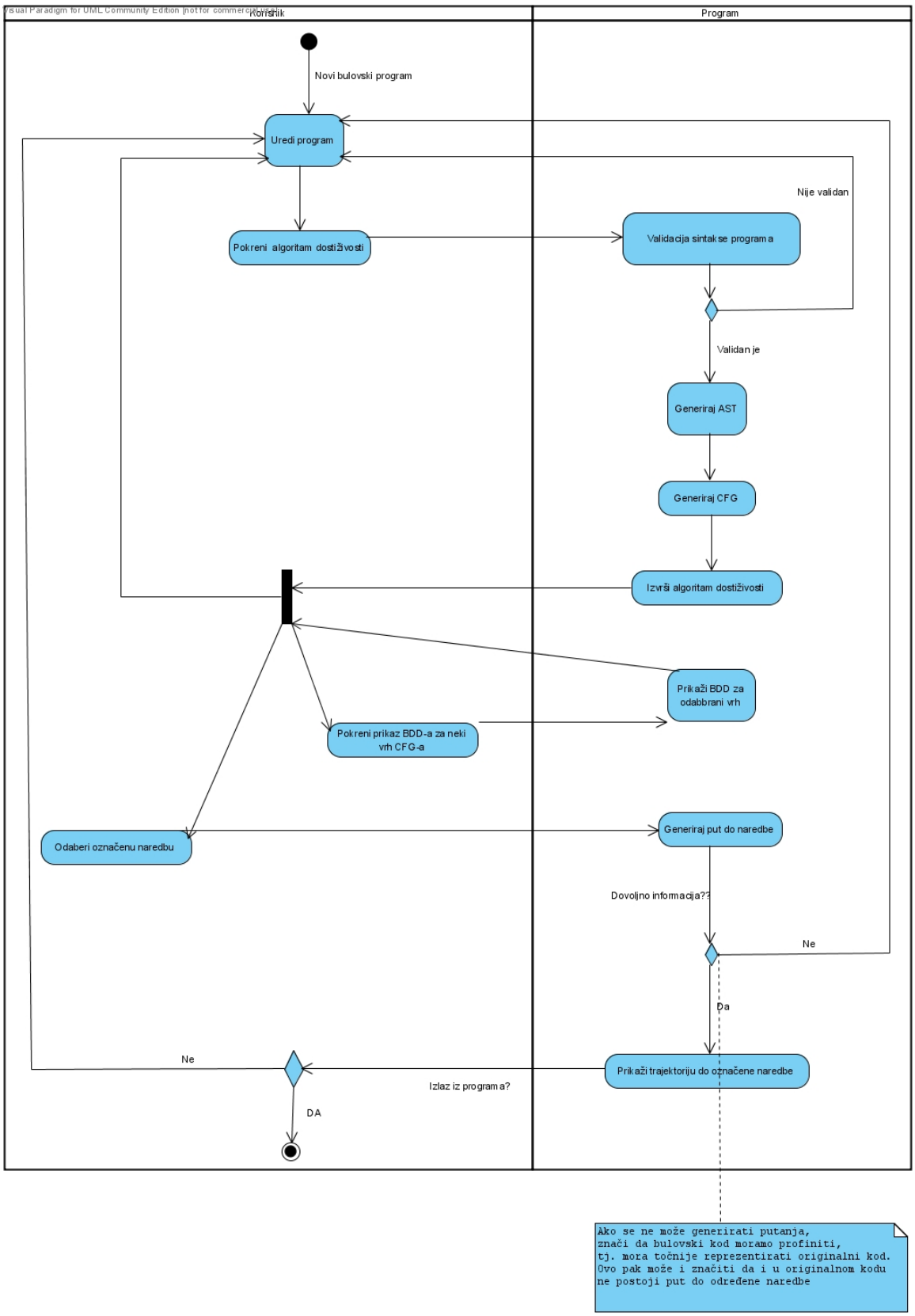


Kratki opis najvažnijih klasa:

- **BMC:** Klasa koja enkapsulira algoritam za verifikaciju safety svojstava bulovskog programa.
- **CFG:** Klasa koja enkapsulira Control Flow Graph bulovskog programa. Također sadrži dodatne funkcije i objekte koje su potrebne za izvođenje samog algoritma (npr. BDD-ove).
- **CFGNode:** Apstraktna klasa koja enkapsulira čvor CFG-a. Sve klase izvedene iz klase CFGNode su jasne same po sebi (svaka izvedena klasa opisuje određenu naredbu bulovske gramatike).
- **Variables:** Klasa koja sadrži imena varijabli bulovskog programa i također sadrži mapiranje tih varijabli na vrhove BDD-ova korištenih kroz algoritam.
- **PathEdgesPartition:** Klasa koja enkapsulira valuaciju varijabli u nekom čvoru CFG-a
- **PathEdges:** Klasa koja sadrži *PathEdgesPartition* objekte spremljene po duljini izvršavanja bulovskog programa do određenog čvora.
- **TrajectoryItem:** Klasa koja enkapsulira jedan korak izvođenja bulovskog programa.
- **Trajectory:** Klasa koja sadrži listu *TrajectoryItem* objekata. Služi za spremanje jednog mogućeg izvršavanja bulovskog programa do određene linije u bulovskom kodu.

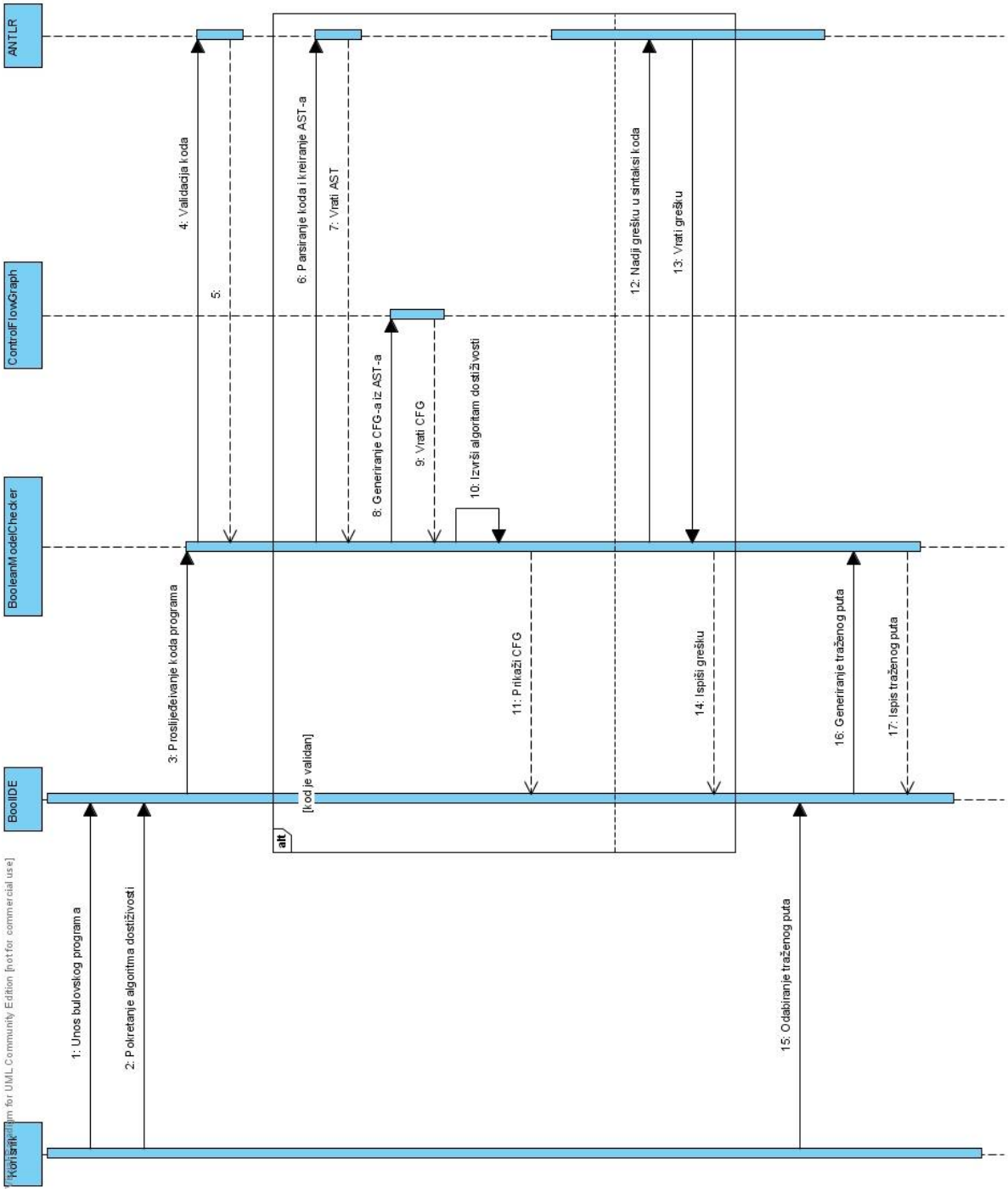
2.4. Activity dijagram

Na slijedećoj slici je prikazan activity dijagram. Dijagram je podijeljen na dva dijela, korisnika i program.


Visual Paradigm for UML Community Edition [not for commercial use]
Korisnik
Program
Novi bulovski program
Uredi program
Pokreni algoritam dostiživosti
Validacija sintakse programa
Nije validan
Validan je
Generiraj AST
Generiraj CFG
Izvrši algoritam dostiživosti
Prikaži BDD za odabbrani vrh
Pokreni prikaz BDD-a za neki vrh CFG-a
Odaberi označenu naredbu
Generiraj put do naredbe
Dovoljno informacija??
Ne
Da
Prikaži trajektoriju do označene naredbe
Ne
Izlaz iz programa?
DA
Ako se ne može generirati putanja,
znači da bulovski kod moramo profiniti,
tj. mora točnije reprezentirati originalni kod.
Ovo pak može i značiti da i u originalnom kodu
ne postoji put do određene naredbe

2.5. Sequence dijagram

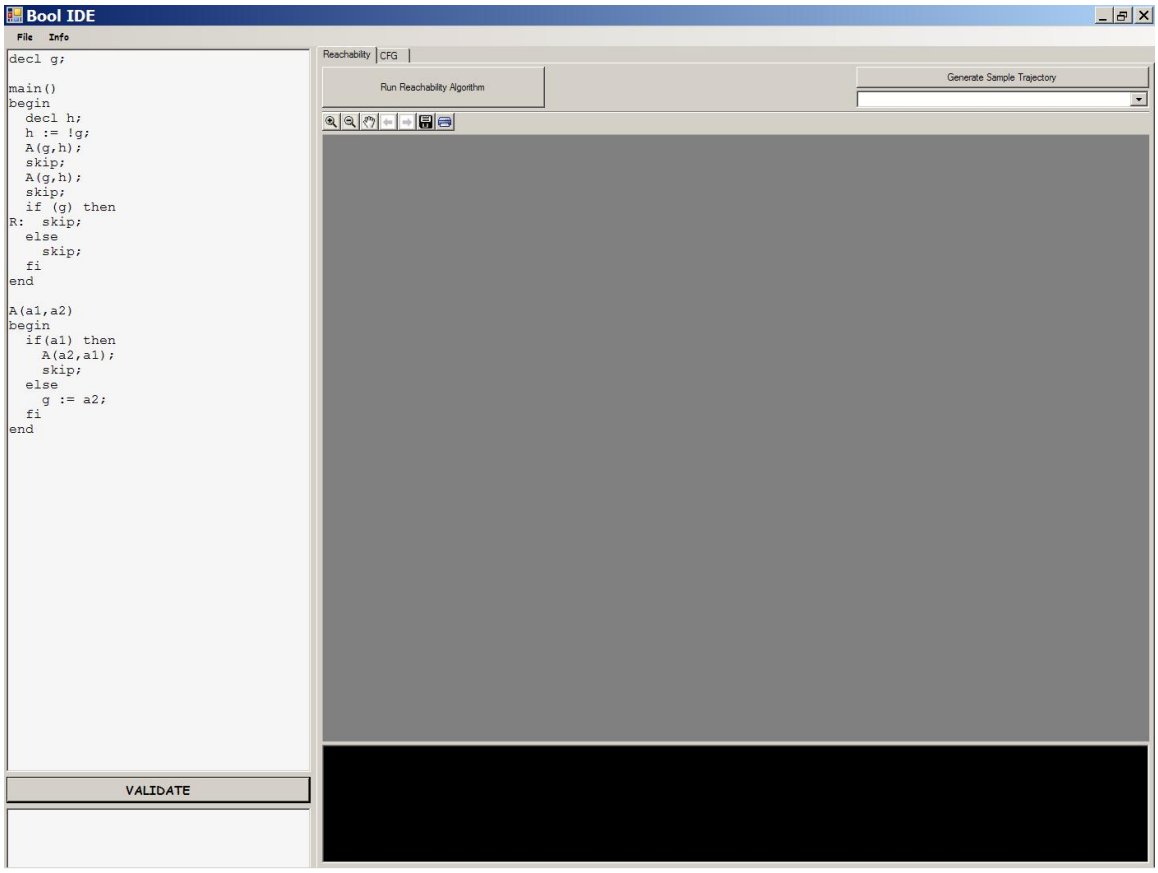
Idući dijagram koji prikazujemo je sequence dijagram. BoolIDE je grafičko sučelje programa, BooleanModelChecker sadrži algoritme dostiživosti, ControlFlowGraph služi za generiranje CFG-a bulovskog programa, te ANTLR služi za parsiranje bulovskog koda.



3. GUI

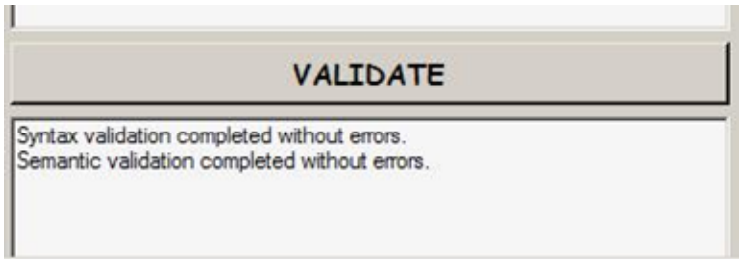
3.1. Glavni prozor

Na Slici 1 vidimo izgled glavnog prozora programa. Glavni prozor je podijeljen na nekoliko dijelova. S lijeve strane se nalazi uređivač teksta u koji se unosi kod bulovskog programa. Također, kod se može i učitati iz vanjske tekstualne datoteke pomoću izbornika *File* → *Open*. Ispod uređivača teksta se nalazi gumb *VALIDATE* koji provjerava sintaksu i semantiku koda. S desne strane na izbor imamo dva taba, *Reachibility* tab i *CFG* tab. Oba taba će biti pobliže objašnjeni u nastavku teksta.



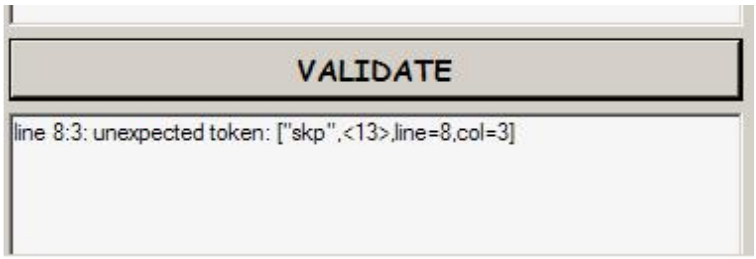
Slika 1. Glavni prozor

Pritiskom na gumb *VALIDATE* dobit ćemo tekstualnu poruku o ispravnosti našeg bulovskog koda. Ako je sve u redu sa sintaksom i semantikom dobit ćemo poruku kao na Slici 2.



Slika 2. Bulovski kod je valjan

Ako imamo grešku u kodu, npr. umjesto "skip" napišemo "skp" u osmoj liniji koda (pogledat Sliku 1) dobit ćemo slijedeću poruku:



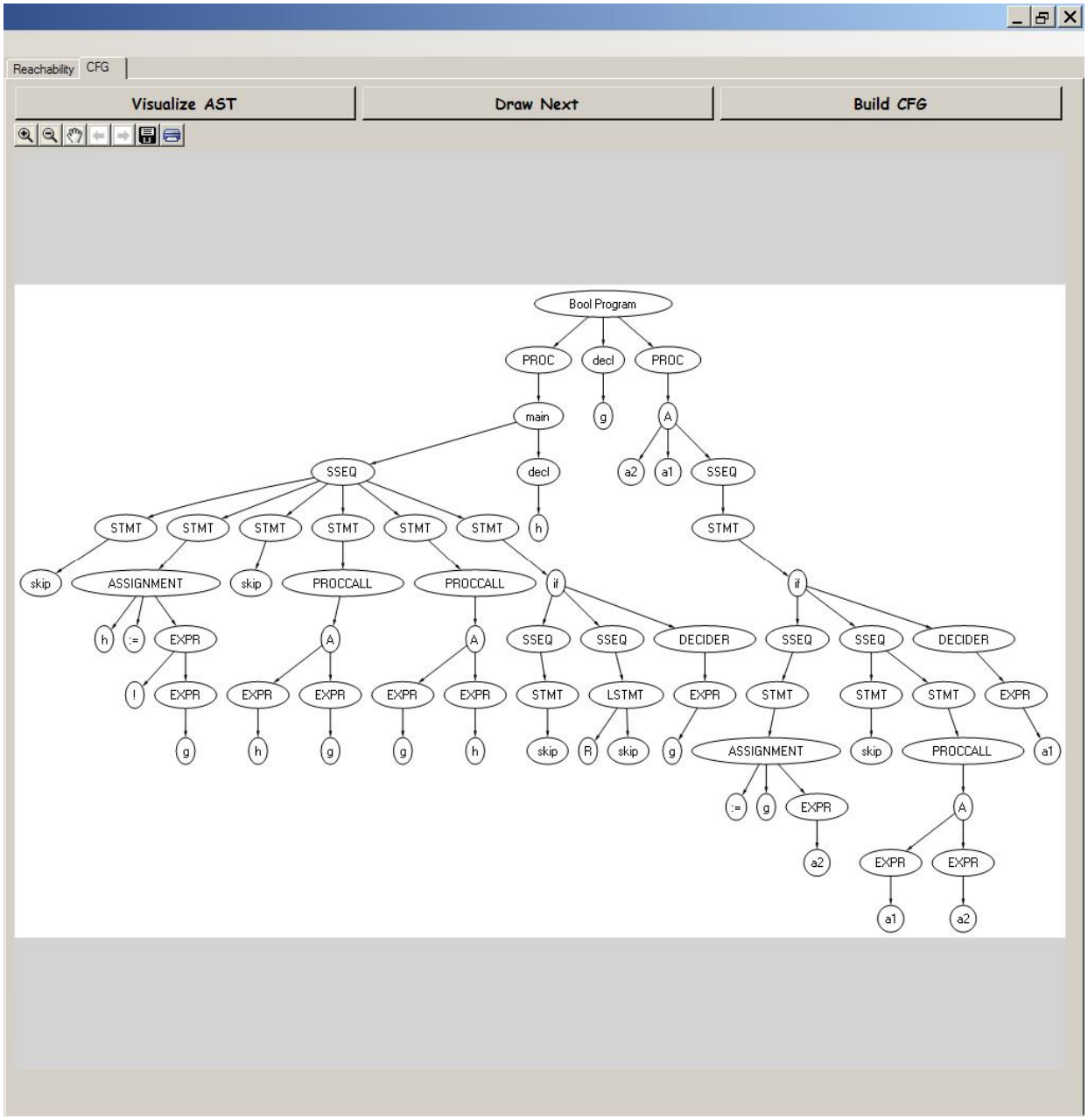
Slika 3. Sintaksna greška u bulovskom kodu

3.2. CFG tab

CFG tab je informativnog karaktera, te nije potreban za potpuno funkcioniranje programa. U *CFG* tabu možemo prikazati neke međukorake izvršavanja algoritma, kao što je izgradnja AST-a i CFG-a. *CFG* tab sadrži tri gumba:

- *Visualize AST*
- *Draw Next*
- *Build CFG*

Pritiskom na gumb *Visualize AST* prikazat će se AST našeg bulovkog koda. AST je stablo koje prikazuje strukturu programa i pogodno je za daljnju analizu programa. Na Slici 4 vidimo primjer AST-a bulovskog koda sa Slike 1.

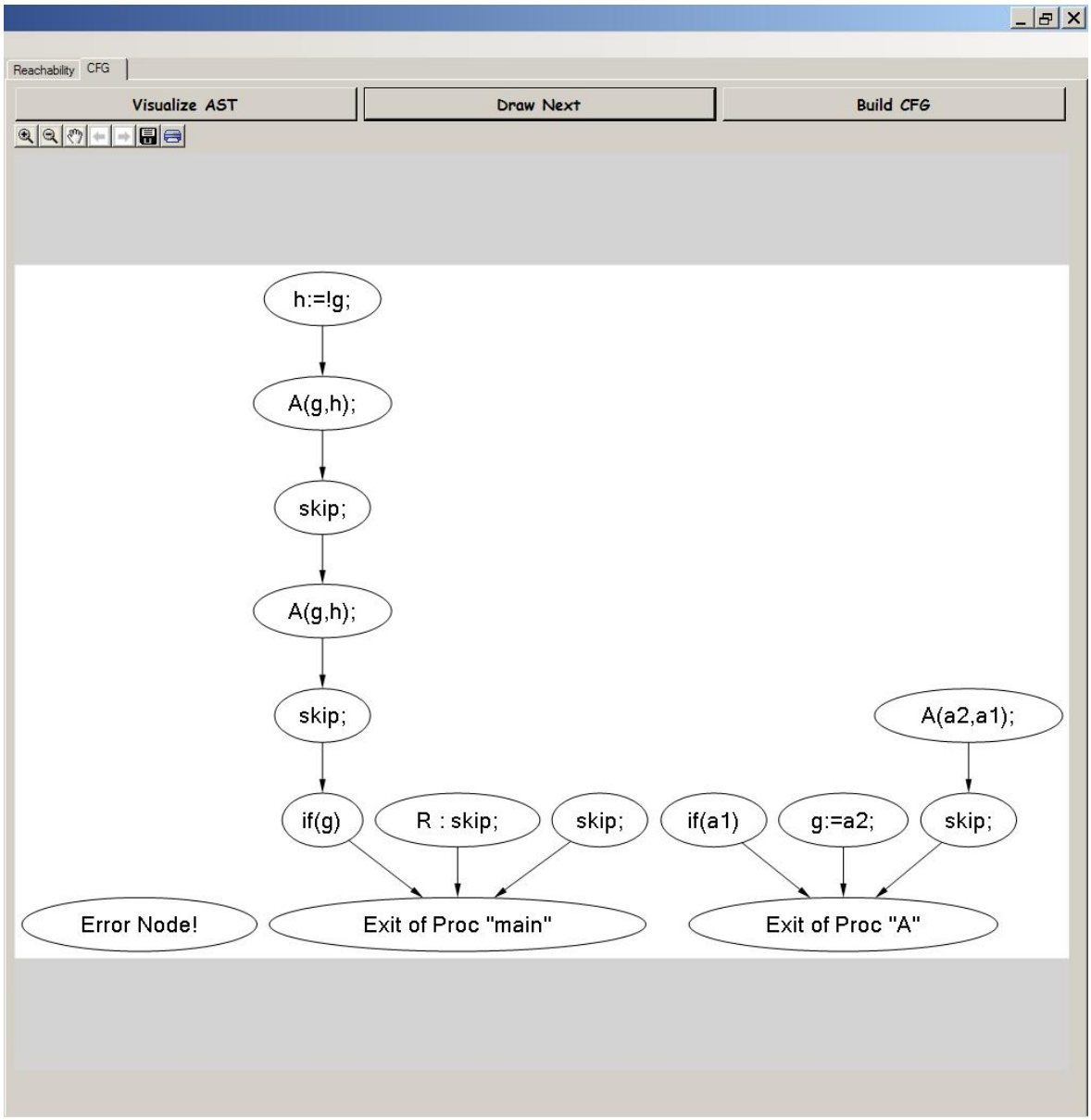


Slika 4. Gumb *Visualize AST*

Na grafu sa Slike 4 vidimo sintaktičke tipove našeg bulovskog programa. Sada ćemo dati njihovo kratko objašnjenje:

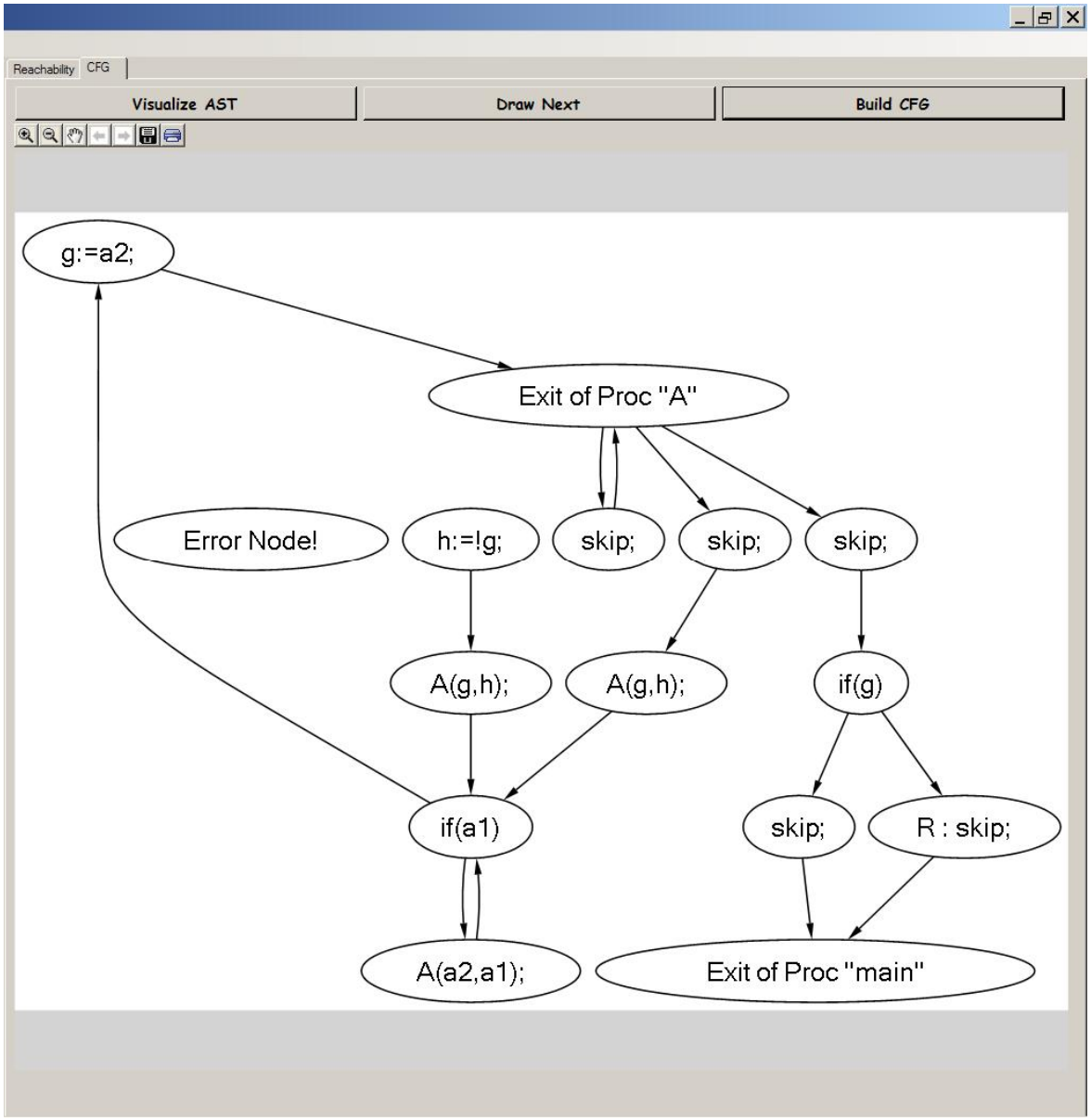
- ASSIGNMENT – naredba pridruživanja
- DECIDER – uvjetno grananje
- EXPR – izraz (pridruživanje varijabli ili formula nad skupom varijabli)
- STMT – naredba bulovskog programa
- LSTMT – naredba označena labelom
- SSEQ – niz naredbi
- PROC – procedura (podstablo su naredbe u proceduri)
- PROCCALL – poziv procedure

Pritiskom na gumb *Draw Next* iscrtat će se graf sljedbenika naredbi kao što je to prikazano na Slici 5. Svaka naredba ima jednog slijebenika (procedura se uzima kao jedna naredba) te graf sljedbenika prikazuje uređaj na skupu svih naredbi u bulovskom programu. Graf sljedbenika služi za izgradnju CFG-a.



Slika 5. Gumb *Draw Next*

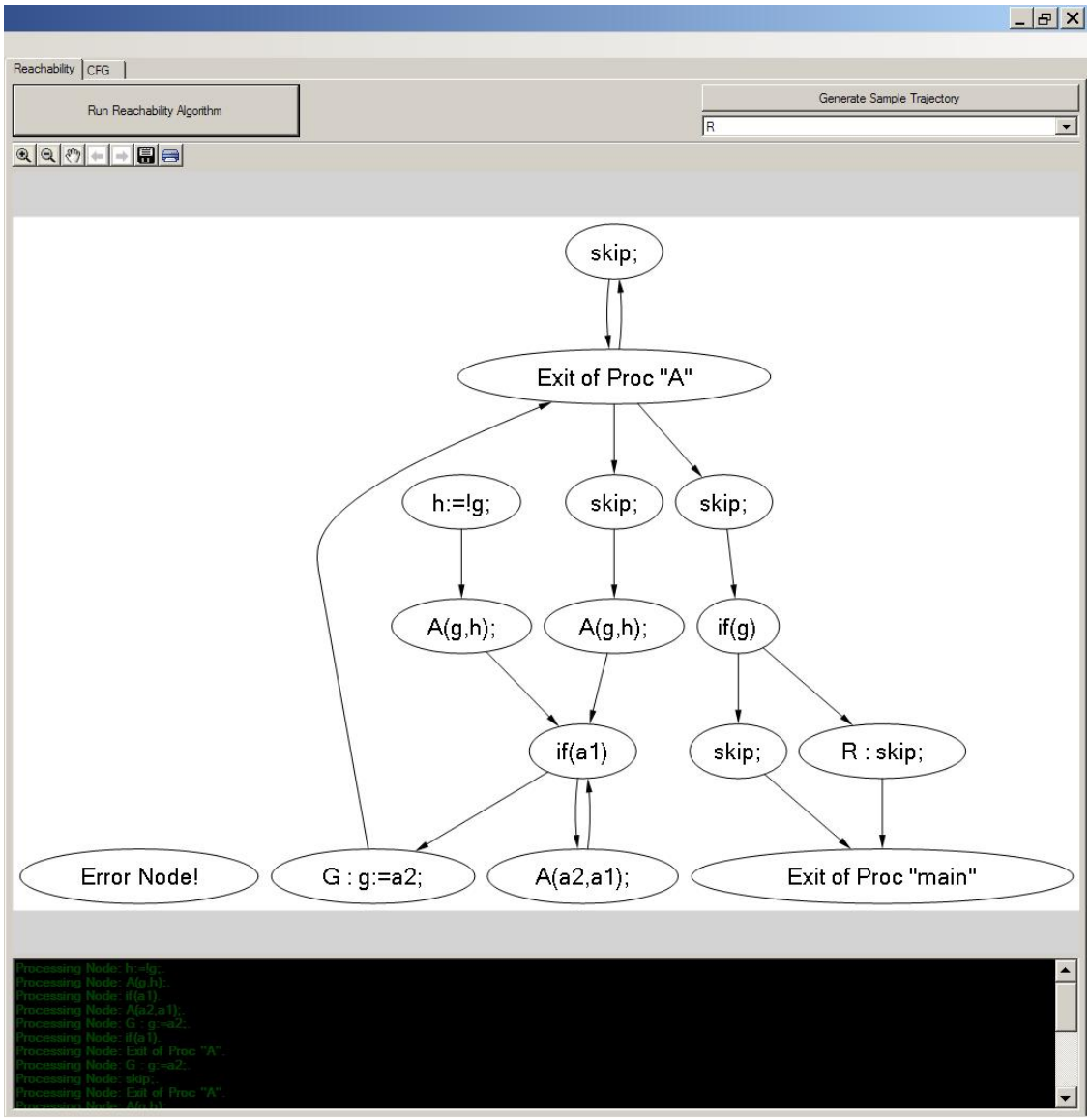
Zadnji gumb u CFG tabu je *Build CFG*. Pritiskom na njega izgradit će se CFG bulovskog programa te se prikazat na ekranu (Slika 6). Ovdje se ne izvršava nikakav algoritam dostiživosti, tj., možemo vidjeti strukturu samog bulovskog programa u obliku CFG-a bez pokretanja algoritma, te vizualno vidjeti moguća izvršavanja programa.



Slika 6. Gumb *Buld CFG*

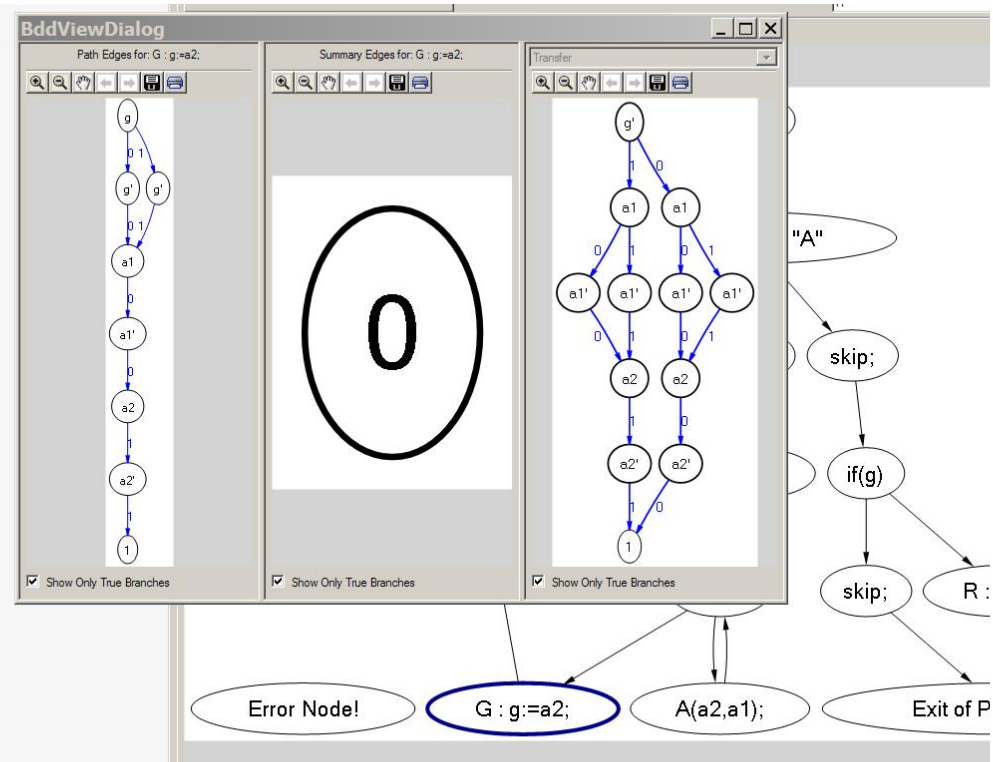
3.3. Reachibility tab

Reachibility tab je osnovni dio programa pomoću kojeg korisnik provjerava dostiživost određene naredbe u bulovskom kodu. Sadrži dva gumba, *Run Reachibility Algoritam* i *Generate Sample Trajectory*. Pritiskom na gumb *Run Reachibility Algoritam* pokreće se algoritam dostiživosti. Na ekranu se prikazuje CFG bulovksog programa, a na dnu ekrana se vidi trenutno stanje algoritma (ispisuje se čvor koji se obrađuje u tom trenutku). Primjer možemo vidjeti na Slici 7.

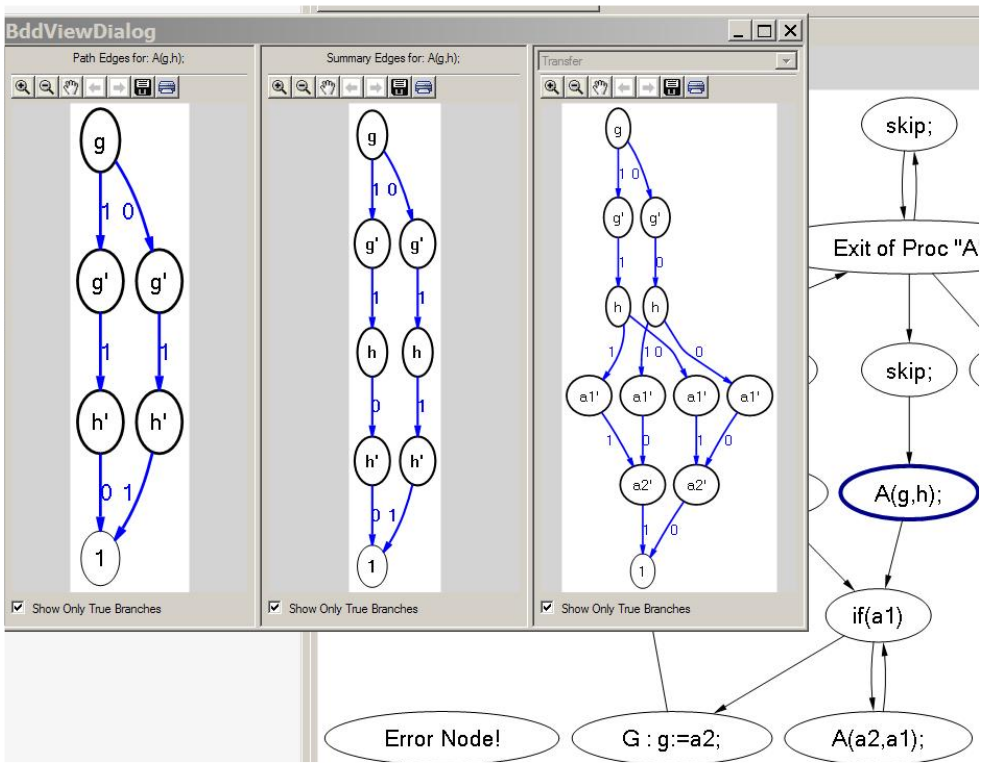


Slika 7. Izvršavanje algoritma dostiživosti

Nakon izvršavanja algoritma, korisnik ima nekoliko opcija, kao npr. pregled BDD-ova, generiranje puta dostiživosti, itd. Klikom na bilo koji čvor CFG-a može dobiti prikaz BDD-ova za dani čvor. Ova opcija je također informativnog karaktera ali može pomoći boljem razumijevanju dobivenih rezultata. Npr., klikom na čvor "G: g:=a2;" dobijemo prikaz BDD-a kao na Slici 8, te klikom na čvor "A(g,h);" (drugi poziv funkcije u kodu) dobijemo prikaz BDD-a kao na Slici 9.



Slika 8. Prikaz BDD-a za čvor "G: g:=a2;"

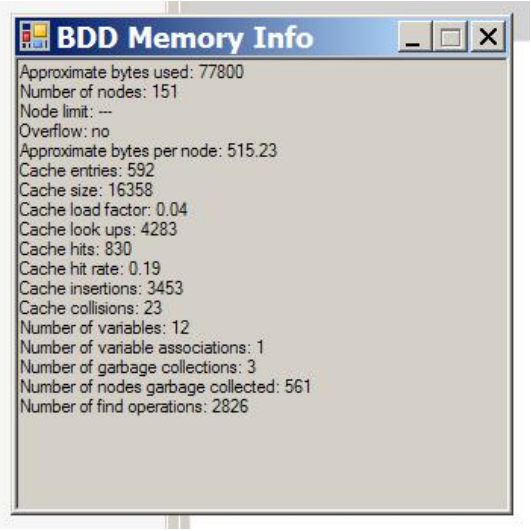


Slika 9. Prikaz BDD-a za čvor "A(g,h);"

Objasnimo na primjeru Slike 9 prikaz BDD-ova. Primijetimo da svaka varijabla dolazi u dva oblika, sa crticom i bez. Bez crtice označavamo stanje varijable prije izvršavanja određene naredbe, dok sa crticom označavamo vrijednost varijable nakon izvršavanja naredbe. Svaka naredba ima dva(tj. tri ako je poziv procedure) BDD-a:

- ***PathEdges*** – Opisuje moguća stanja varijabli koja dovode do izvršavanja izabrane naredbe. Na našem primjeru bez obzira da li je na početku g=0 ili g=1 naredba "A(g,h)" (drugi poziv u kodu) će se uvijek izvršiti. Vidimo da je g'=1 bez obzira na početnu vrijednost od g, tj. A(g,h) će se uvijek pozvati sa vrijednošću g=1.
- ***SummeryEdges*** (samo kod poziva procedure) – Opisuje ponašanje funkcije. Na prikazanom BDD-u se lako vidi što se dogodi sa varijablama g i h. Bez obzira na početnu vrijednost, varijabla g će po završetku funkcije poprimiti vrijednost 1, dok će varijabla h zadržati istu vrijednost.
- ***Transfer*** – Opisuje korak izvođenja naredbe. Kod poziva funkcije opisuje prijenos varijabli u unutarnje parametre te promjene nad globalnim varijablama, dok kod ostalih naredbi opisuje promjene globalnih i lokalnih varijabli.

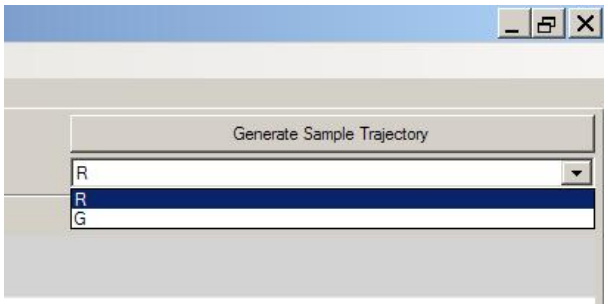
Pošto BDD-ovi intenzivno koriste memoriju, te veliki bulovski programi mogu zauzet poprilično veliku količinu memorije, korisniku je dana opcija *Info → Bdd Memory* koja prikazuje trenutno zauzeće memorije te još neke dodatne informacije vezane za BDD-ove (Slika 10).



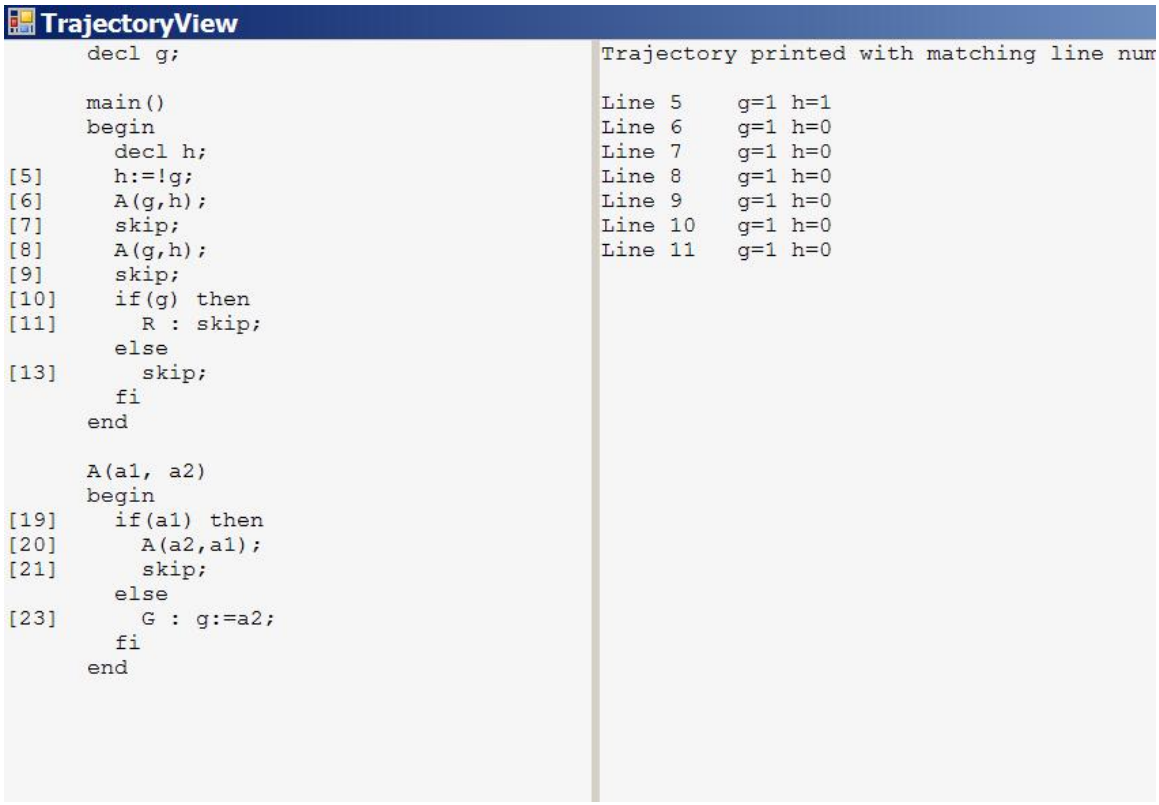
Slika 10. Prozor koji prikazuje informacije o korištenim BDD-ovima

3.4. TrajectoryView

Sada ćemo objasniti najvažniji dio programa. Nakon izvršenja algoritma dostiživosti, u desnom gornjem uglu, ispod gumba *Generate Sample Trajectory* na izbor će nam biti ponuđene labele naredbi bulovskog koda. Nakon biranja labele (odgovara određenoj liniji u bulovskom kodu) kao što je prikazano na Slici 11, klikom na gumb *Generate Sample Trajectory* prikazat će se ekran sa putanjom do izabrane naredbe (ako postoji) (Slika 12).



Slika 11. Izbornik za labele naredbi



Slika 12. Trajektorija do određene naredbe

Na desnoj strani *TrajectoryView*-a možemo lako pratiti evaluaciju varijabli do određene naredbe. Npr., vidimo da je linija 11 dostiživa ako su početne vrijednosti varijabli g=1 i h=1. Treba napomenuti da implementirani algoritam uvijek daje najkraću putanju.

Upravo generiranje puta do određene naredbe je najveća snaga Boolean Model Checkera. Korisnik na taj način može pratiti program i naći grešku. Treba napomenuti da je bulovski program samo apstrakcija originalnog proceduralnog programa (napisanog npr. u C-u) te da profinjavanjem bulovskog programa (što se također treba automatizirat) se lakše nalaze greške u kodu.

Najčešća upotreba programa je generiranje puta do čvora greške (Error Node), kojeg sami označimo. Znači, ako postoji izvršavanje programa koji dovodi do greške, algoritam će ga naći.

4. Dodatne informacije

Boolean Model Checker treba uzeti kao polovicu programa za automatsku verifikaciju safety svojstava. Druga polovica je automatsko mapiranje koda nekog proceduralnog programa u bulovski kod koji prepoznaje Boolean Model Checker. To je jedna od mogućih nadogradnji programa, ali također takve programe korisnici mogu razviti neovisno o Boolean Model Checkeru. Gramatika bulovskog programa je definirana u formatu parser generatora ANTLR. Gramatika je kontekstno slobodno, tipa LL(2) i dana je na kraju ovog poglavlja.

Boolean Model Checker je implementiran u programskom jeziku C# pomoću razvojnog alata Visual Studio 2005. Od vanjskih programa korišten je parser ANTLR te biblioteka CMUbdd za manipuliranje BDD-ovima. Biblioteka GLEE je korištena za iscrtavanje grafova. UML dijagrami koji se koriste kroz dokumentaciju su rađeni pomoću alata ArgoUML i Visual Paradigm for UML 7.0.

Za izovđenje programa potrebno je imati operativni sustav Microsoft Windows XP ili Microsoft Windows Vista s instaliranim .NET framework 2.0. Brzina izvođenja programa u potpunosti ovisi o brzini procesora i količini memorije (što su veći bulovski programi to se koristi više memorije), te je za velike bulovske programe preporučljivo imati najnoviju generaciju računala. Program ne podržava višenitnost te broj jezgri ne utječe na brzinu izvođenja.

```
Options { language = "CSharp"; }
/** Gramatika za sintaksu Bool programskoh jezika koristenogu Bebop-u prema
definiciji danoj u http://research.microsoft.com/slam/papers/bebop.pdf , strana 5*/

class BoolParser extends Parser;
options
{
      k=2;
      buildAST=true;
}
tokens
{
  PROC;  SSEQ;  STMT;  LSTMT;  ASSIGNMENT;  DECIDER;  PROCCALL;  EXPR;
}

prog     :  (decl)* (proc)*;

decl     :  "decl"^ (ID (COMMA! ID)*) SEMI!;

proc     :  ID^ LPAREN! (ID (COMMA! ID)*)? RPAREN! "begin"! (decl)* sseq "end"!
        {#proc = #([PROC,"PROC"],#proc);};

sseq     :  (lstmt)+ {#sseq = #([SSEQ,"SSEQ"],#sseq);};

lstmt     :  stmt {#lstmt = #([STMT,"STMT"],#lstmt);}
    |  ID COLON! stmt {#lstmt = #([LSTMT,"LSTMT"],#lstmt);};

stmt     :  "skip" SEMI!
    |  "print"^ LPAREN! (expr (COMMA! expr)*) RPAREN! SEMI!
    |  "goto"^ ID SEMI!
    |  "return"^ SEMI!
    |  (ID (COMMA! ID)*) ASSIGN (expr (COMMA! expr)*) SEMI!
        {#stmt = #([ASSIGNMENT,"ASSIGNMENT"],#stmt);}
    |  "if"^ LPAREN! decider RPAREN! "then"! sseq "else"! sseq "fi"!
    |  "while"^ LPAREN! decider RPAREN! "do"! sseq "od"!
    |  "assert"^ LPAREN! decider RPAREN! SEMI!
    |  ID^ LPAREN! (expr (COMMA! expr)*)? RPAREN! SEMI!
        {#stmt = #([PROCCALL,"PROCCALL"],#stmt);}
;

decider   :  (QMARK  | expr) {#decider = #([DECIDER,"DECIDER"],#decider);};

expr   :  (EMARK expr | LPAREN! expr RPAREN! | ID | CONST)
(options{greedy=true;}:BINOP expr)?
        {#expr = #([EXPR,"EXPR"],#expr);};

class BoolLexer extends Lexer;
options {
    k=2;
}

ASSIGN  :  ":=";
COMMA    :  ',';
SEMI    :  ';';
COLON    :  ':';
LPAREN  :  '(';
RPAREN  :  ')';
QMARK    :  '?';
EMARK    :  '!' ('=' {$setType(BINOP);})?;

ID    :  ('a'..'z' | 'A'..'Z' | '_')('a'..'z' | 'A'..'Z' | '0'..'9' | '_')*
| ('{' ~('{' | '}') '}');

BINOP   :  '|' | '&' | '^' | '=' | "=>";

CONST   :  '0' | '1';

WS    :  (' '
    |  '\t'
    |  '\n'{ newline(); }
    |  '\r')
    { _ttype = Token.SKIP; }
    ;
```

5. Rječnik pojmova

ANTLR: Another Tool for Language Recognition, alat za generiranje koda parsera iz zadane gramatike.

AST: Abstract Syntax Tree, stablo koje apstraktno reprezentira struktura koda.

BDD: Binary Decision Diagram, binarno stablo koje kompaktnu zapisuje bulovske formule u obliku grafa.

BDDlib: Programska biblioteka za izgradnju i baratanje bdd-ovima u software-u.

Bulovski program: Posebna vrsta programa sintakse slične C-u sa sličnim kontrolnim strukturama, uz bitnu razliku da su sve varijable bulovskog tipa. Nema praktičnu primjenu već služi za pogodnu apstrakciju programa napisanih u drugim programskim jezicima.

CFG: Control Flow Graph, usmjereni graf koji reprezentira moguće tokove izvršavanja programa.

Liveness svojstva: Svojstva modela koja izražavaju da će se dogoditi nešto dobro.

Model Checking: Naziv za automatsku provjeru jednostavnih modela nekog složenijeg sustava

Parser: Program koji intepretira niz tokena i prevodi u višu reprezentaciju koda (najčešće u AST)

Safety svojstva: Svojstva modela koja izražavaju da se neće dogoditi nešto loše.

UML: Unified Modeling Language, standarizirani jezik za modeliranje korišten u softverskom inžinjerstvu

Literatura

[1] Bebop: A Symbolic Model Checker for Boolean Programs, T. Ball, S. K. Rajamani, *SPIN 2000 Workshop on Model Checking of Software*, LNCS 1885, August/September 2000, pp. 113-130.

[2] Bebop: A Path-sensitive Interprocedural Dataflow Engine, T. Ball, S. K. Rajamani, *PASTE 2001*