Software model checking

Matko Botinčan

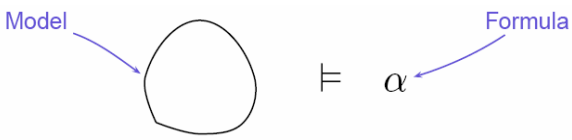
Seminar za teorijsko računarstvo
30.11.2006.

Slideovi vezani uz SLAM preuzeti iz PLDI'03 tutoriala:
Software Model Checking with SLAM (T. Ball, S. K. Rajamani)

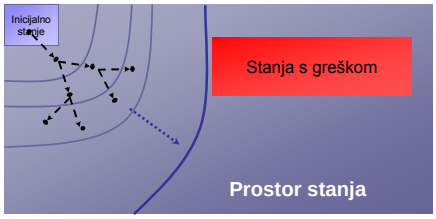
Zašto "model checking"?

- Dva značenja:

1. "model checking"



2. algoritmi za verifikaciju sustava
   tipično: eksploracija prostora stanja
   (== iscrpljujuće testiranje)



Zašto "model checking" softvera?

- "A defect that costs $1 to fix on the programmer's desktop costs $100 to fix once it is incorporated into a complete program and many thousands of dollars if it is identified only after the software has been deployed in the field."

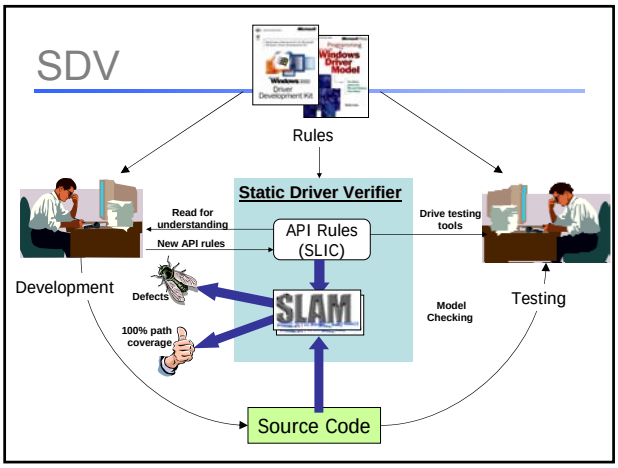
Building a Better Bug Trap – The Economist 06/2003

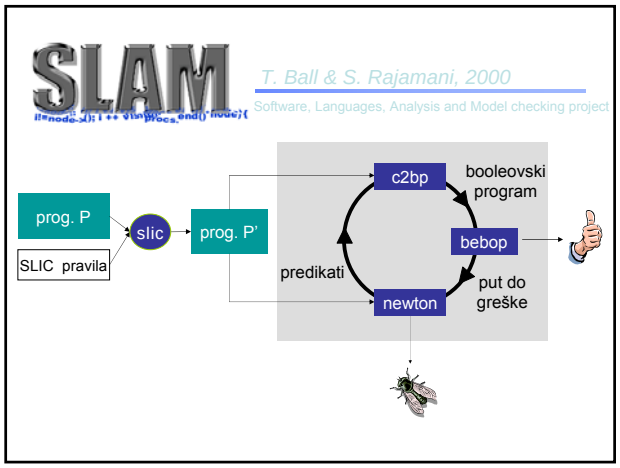
Real-life motivacija

SDV

- Windows Driver Foundation (WDF)
  - WDF Driver Verification Tools
    - PREfast
    - **Static Driver Verifier (SDV)**
      - "Finding Driver Bugs at Compile-Time"

http://www.microsoft.com/whdc/devtools/tools/SDV.mspx





Primjer 1

1. iteracija: c2bp → bebop → newton

```
void main () {
  int x = 0;
  x++;
  assert (x==1);
}
```

apstrakcija

```
main () {
  x==1 = F;
  x==1 = {F,T};
  assert (x==1);
}
```

assert (F) je dostiživ u Booleovskom programu, ali ne i u C programu.

x==1 ... booleovska varijabla

2. iteracija: c2bp → bebop → kraj

```
void main () {
  int x = 0;
  x++;
  assert (x==1);
}
```

apstrakcija

```
main () {
  x==1, x==0 = F, T;
  x==1, x==0 = T, F;
  assert (x==1);
}
```

assert (F) NIJE dostiživ u BP,

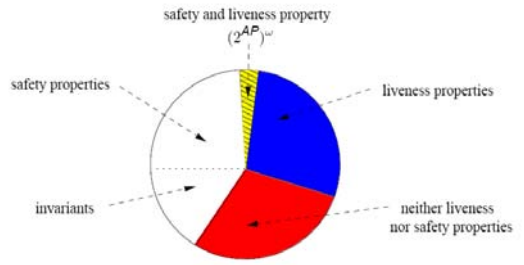
→ nije dostiživ niti u C programu

x==1 , x==0 ... booleovske varijable

Podsjetnik: safety vs. liveness

- Safety svojstva – nikada se neće dogoditi nešto "loše"
  - parcijalna korektnost: program nikada neće dati pogrešan rezultat
- Liveness svojstva – nešto "dobro" će se sigurno jednom dogoditi (ali ne znamo kada)
  - terminacija – program će dati rezultat

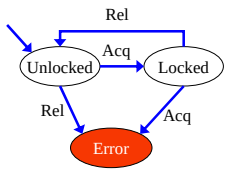
Kompletna slika



Primjer 2

- Safety svojstvo koje želimo provjeriti SLAM-om:
  - "Driver ne smije pozvati `KeReleaseSpinLock()` ukoliko prethodno nije pozvao `KeAcquireSpinLock()`"
- Primjer liveness svojstva koje ne možemo provjeriti SLAM-om (pa ni SDV-om):
  - "Ukoliko driver pozove `KeAcquireSpinLock()` onda jednom mora pozvati i `KeReleaseSpinLock()`"

Automat



SLIC kod

```
state {
  enum {Locked,Unlocked}
    s = Unlocked;
}

KeAcquireSpinLock.entry {
  if (s==Locked) abort;
  else s = Locked;
}

KeReleaseSpinLock.entry {
  if (s==Unlocked) abort;
  else s = Unlocked;
}
```

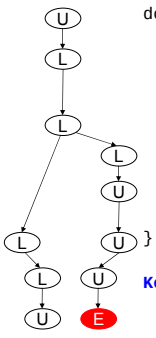
Primjer 2

Zadovoljava li ovaj programski kod željeno svojstvo?

```
do {
  KeAcquireSpinLock();

  nPacketsOld = nPackets;

  if(request){
      request = request->Next;
      KeReleaseSpinLock();
      nPackets++;
  }
} while (nPackets != nPacketsOld);

KeReleaseSpinLock();
```

Primjer 2

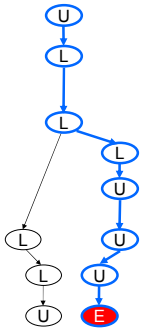
Model checking boolovskog programa (bebop)




```
do {
  KeAcquireSpinLock();

  if(*){

      KeReleaseSpinLock();

  }
} while (*);

KeReleaseSpinLock();
```

Primjer 2

Da li je pronađeni put koji vodi do greške dopustiv u C programu? (newton)

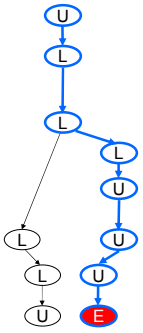



```
do {
  KeAcquireSpinLock();

  nPacketsOld = nPackets;

  if(request){
      request = request->Next;
      KeReleaseSpinLock();
      nPackets++;
  }
} while (nPackets != nPacketsOld);

KeReleaseSpinLock();
```

Primjer 2

b : (nPacketsOld == nPackets)

Dodavanje novog predikata u booleovski program (c2bp)

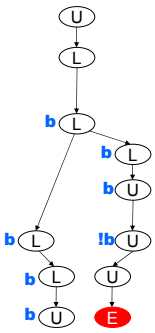



```
do {
  KeAcquireSpinLock();

  nPacketsOld = nPackets; b = true;

  if(request){
      request = request->Next;
      KeReleaseSpinLock();
      nPackets++; b = b ? false : *;
  }
} while (nPackets != nPacketsOld);  !b

KeReleaseSpinLock();
```

Primjer 2

b : (nPacketsOld == nPackets)

Model checking profinjenog booleovskog programa (bebop)

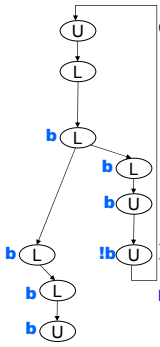



```
do {
  KeAcquireSpinLock();

  b = true;

  if(*){

      KeReleaseSpinLock();
      b = b ? false : *;
  }
} while ( !b );

KeReleaseSpinLock();
```

Primjer 2

b : (nPacketsOld == nPackets)

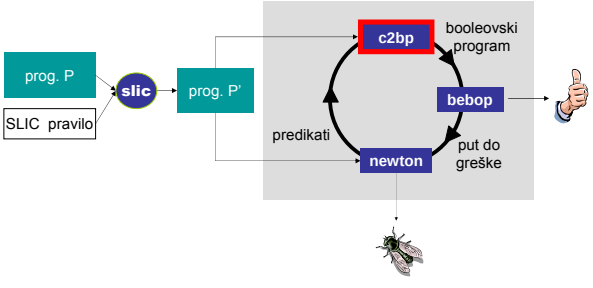
Model checking profinjenog booleovskog programa (bebop)




```
do {
  KeAcquireSpinLock();

  b = true;

  if(*){

      KeReleaseSpinLock();
      b = b ? false : *;
  }
} while ( !b );

KeReleaseSpinLock();
```

SLAM – pogled iznutra

SLAM



c2bp: Apstrakcija predikata za C programe

Ulaz
- P : C program
- F = {$e_1,\ldots,e_n$}
  - e_i ... predikati (C izrazi) s booleovskom vrijednošću
  - svaki e_i reprezentira skup stanja za koje je e_i istinit

Izlaz: *booleovski program* B(P,F):
- ista control-flow struktura kao P
- booleovske varijable {$b_1,\ldots,b_n$} odgovararju predikatima {$e_1,\ldots,e_n$}
- svako svojstvo istinito za B(P,F) istinito je i za P

c2bp algoritam

- Provodi modularnu apstrakciju
  - svaka procedura apstraktira se izolirano
  - -> sve varijable u predikatima su ili samo lokalne ili samo globalne
- Unutar pojedine procedure, svaka naredba apstraktira se izolirano
  - nema control-flow analize
  - nema određivanja invarijanti petlji

  (najbrutalniji mogući brute-force:)

C-

Types	τ	::=	void | bool | int | ref τ
Expressions	e	::=	c | x | e_1 op e_2 | &x | *x
LExpression	l	::=	x | *x
Declaration	d	::=	τ $x_1,x_2,\ldots,x_n$
Statements	s	::=	skip | goto $L_1,L_2\ldots L_n$ | L: s
		|	assume(e)
		|	l = e
		|	l = f ($e_1,e_2,\ldots,e_n$)
		|	return x
		|	s_1 ; s_2 ;... ; s_n
Procedures	p	::=	τ f (x_1: τ_1,x_2 : τ_2,...,x_n : τ_n)
Program	g	::=	d_1 d_2 ... d_n p_1 p_2 ... p_n

C--

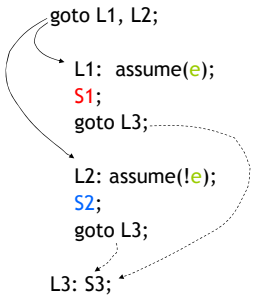
Types	τ	::=	void | bool | int
Expressions	e	::=	c | x | e_1 op e_2
LExpression	l	::=	x
Declaration	d	::=	τ $x_1,x_2,\ldots,x_n$
Statements	s	::=	skip | goto $L_1,L_2\ldots L_n$ | L: s
		|	assume(e)
		|	l = e
		|	f ($e_1,e_2,\ldots,e_n$)
		|	return
		|	s_1 ; s_2 ;... ; s_n
Procedures	p	::=	f (x_1: τ_1,x_2 : τ_2,...,x_n : τ_n)
Program	g	::=	d_1 d_2 ... d_n p_1 p_2 ... p_n

BP

Types	τ	::=	void | bool
Expressions	e	::=	c | x | e_1 op e_2
LExpression	l	::=	x
Declaration	d	::=	τ $x_1,x_2,\ldots,x_n$
Statements	s	::=	skip | goto $L_1,L_2\ldots L_n$ | L: s
		|	assume(e)
		|	l = e
		|	f ($e_1,e_2,\ldots,e_n$)
		|	return
		|	s_1 ; s_2 ;... ; s_n
Procedures	p	::=	f (x_1: τ_1,x_2 : τ_2,...,x_n : τ_n)
Program	g	::=	d_1 d_2 ... d_n p_1 p_2 ... p_n

if -- syntactic sugar

```
if (e) {
    S1;
} else {
    S2;
}
S3;
```



C primjer

```
int g;

main(int x, int y){

  cmp(x, y);

  if (!g) {
     if (x != y)
        assert(0);
  }
}
```

```
void cmp (int a , int b) {
  if (a == b)
       g = 0;
  else
       g = 1;
}
```

C-- primjer

```
int g;

main(int x, int y){

  cmp(x, y);

  assume(!g);
  assume(x != y)
  assert(0);
}
```

```
void cmp(int a , int b) {
  goto L1, L2;

  L1: assume(a==b);
       g = 0;
       return;

  L2: assume(a!=b);
       g = 1;
       return;
}
```

```
int g;

main(int x, int y){

  cmp(x, y);

  assume(!g);
  assume(x != y)
  assert(0);
}
```

```
void cmp (int a , int b) {
  goto L1, L2

  L1: assume(a==b);
       g = 0;
       return;

  L2: assume(a!=b);
       g = 1;
       return;
}
```

Predikati: {x==y}
{g==0}
{a==b}

```
int g;

main(int x, int y){

  cmp(x, y);

  assume(!g);
  assume(x != y)
  assert(0);
}

decl {g==0} ;

main( {x==y} ) {

}
```

```
void cmp (int a , int b) {
  goto L1, L2

  L1: assume(a==b);
       g = 0;
       return;

  L2: assume(a!=b);
       g = 1;
       return;
}

void cmp ( {a==b} ) {

}
```

Predikati: {x==y}
{g==0}
{a==b}

```
int g;

main(int x, int y){

  cmp(x, y);

  assume(!g);
  assume(x != y)
  assert(0);
}

decl {g==0} ;

main( {x==y} ) {

  cmp( {x==y} );

  assume( ![g==0] );
  assume( ![x==y] );
  assert(0);
}
```

```
void cmp (int a , int b) {
  goto L1, L2

  L1: assume(a==b);
       g = 0;
       return;

  L2: assume(a!=b);
       g = 1;
       return;
}

void cmp ( {a==b} ) {
  goto L1, L2;

  L1: assume( {a==b} );
       {g==0} = T;
       return;

  L2: assume( ![a==b} );
       {g==0} = F;
       return;
}
```

Osnovni problem

- Kako odrediti koje booleovske vrijednosti pridružiti predikatima u pojedinom stanju?
  -> određuje se najslabiji preduvjet (weakest precondition)
- WP(s,e) == najslabiji preduvjet od e obzirom na s

Primjer: WP obzirom na pridruživanje

- Neka je dan izraz y=y+1 i F={ y<4, y<5 }:
  - {y<4}, {y<5} = {y<3}, {y<4}
- WP(x=e,Q) = Q [e/x]
- WP(y=y+1, y<5) =
  (y<5) [y+1 / y] =
  (y+1<5) =
  (y<4)

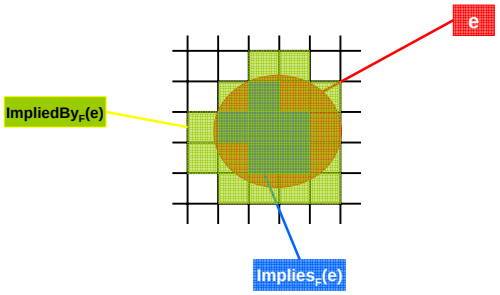
WP problem

- WP(s, e_i) nije uvijek izraziv pomoću { e_1,...,e_n }
- Npr:
  - F = { x==0, x==1, x<5 }
  - WP(x=x+1 , x<5) = x<4
  - "Najbolji" mogući: x==0 || x==1

Apstrakcija izraza pomoću F

- F = { e_1,...,e_n }
- $Implies_F(e)$
  - *najbolja* booleovska funkcija nad F *koja implicira* e
- $ImpliedBy_F(e)$
  - *najbolja* booleovska funkcija nad F *implicirana s* e
  - $ImpliedBy_F(e)$ = !$Implies_F$(!e)

$Implies_F(e)$ i $ImpliedBy_F(e)$



Računanje $Implies_F(e)$

- *minterm* $m = d_1 \wedge \ ... \wedge d_n$
  - gdje je $d_i = e_i$ or $d_i = !e_i$
- $Implies_F(e)$
  - disjunkcija svih minterama koji impliciraju e
- Naïvni pristup:
  1. generiraj svih 2^n mogućih minterama
  2. za svaki minterm m, upotrijebi dokazivač teorema za provjeru *valjanosti* svake od implikacija m⇒e
- Moguće brojne optimizacije...

Apstrakcija pridruživanja

- ako je $Implies_F$(WP(s, e_i)) istinito prije s, tada:
  - e_i je istinito nakon s
- ako je $Implies_F$(WP(s, $!e_i$)) istinito prije s, tada:
  - e_i is lažno nakon s

{e_i} = $Implies_F$(WP(s, e_i)) ? true :
$Implies_F$(WP(s, $!e_i$)) ? false
: *;

Primjer

Izraz: y = y+1;

Predikat: {x==y}

Najslabiji preduvjet:
WP(y=y+1, x==y) = x==y+1

$Implies_F$(x==y+1) = false
$Implies_F$(x!=y+1) = x==y

Apstrakcija u booleovskom programu:
{x==y} = {x==y} ? false : *;

Apstrakcija od assumes

- WP(assume(e) , Q) = e⇒Q
- apstrakcija od assume(e):
  assume($ImpliedBy_F$(e))
- Primjer:
  F = {x==2, x<5}
  assume(x < 2) se apstraktira u:
  assume({x<5} && !{x==2})

Ostale važne apstrakcije

- Apstrakcija poziva funkcija – straightforward, ali tehnički kompliciranо
- Apstrakcija izraza s pointerima
  - problem predstavlja aliasing -> Das-ov unifikacijski algoritam
  - relaksacija memorijskog modela: *(p+i) == *p

Verifikacija konkurentnih programa

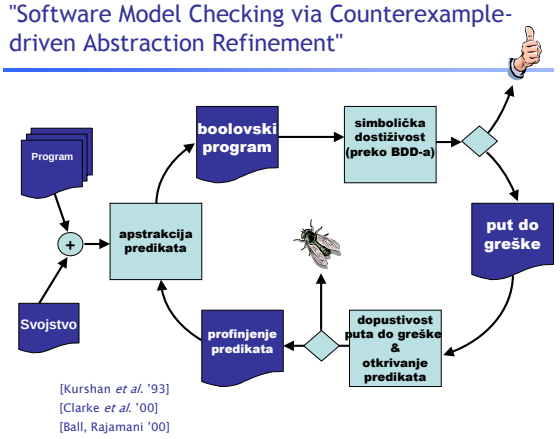
- Interleaving semantika -> eksplozija prostora stanja
  - nešto efikasniji pristupi: semantika parcijalnog uređaja, uvođenje ekvivalencija među stanjima, ...
- Skalabilnije rješenje – Kompozicionalna verifikacija:
  - Model checking + Assume-guarantee
- KISS ("Keep It Simple and Sequential"):
  - Flanagan, Freund, Qadeer ESOP'02
  - Implementacija koja koristi SLAM
  - i dalje samo safety svojstva :(

O izboru dokazivača teorema

- Korišteni automatski dokazivači teorema (ATP) za FO:
  - Simplify [HP/Compaq labs]
    (koriste ga skoro svi: BLAST, Spec#, ESC/Java2,...)
  - Vampyre [R. Majumdar]
    (NB: ovo nije Vampire [A. Voronkov])
    prednost: generira "upotrebljive" dokaze koje SLAM zna heuristički iskoristiti prilikom procesa apstrakcije

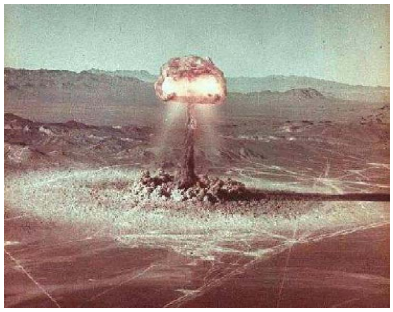
- Predikati u SLAM-u su iz FO_0
  -> zato je i moguća upotreba (efikasnih) ATP-ova
- Postoje nadogradnje za predikate iz nekih drugih (odlučivih) teorija:
  - Presburgerova aritmetika, polja, recordi, ...
- Ove apstrakcije su u načelu loše za pointerske programe
- Shape analysis – FO+TC
  - ne postoji ATP -> manualna konstrukcija predikata
- TO DO: odlučivi fragment separacijske logike
  - postoji translacija u FO -> upotreba FO ATP-a

"Software Model Checking via Counterexample-driven Abstraction Refinement"



[Kurshan *et al.* '93]
[Clarke *et al.* '00]
[Ball, Rajamani '00]

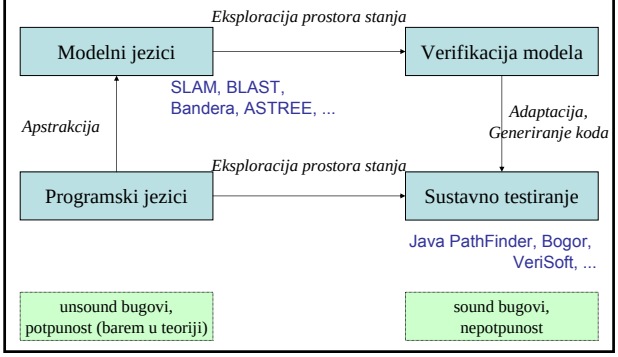
Glavna konkurencija



BLAST

Berkeley Lazy Abstraction Software Verification Tool

Software model checking svijet



Prijedlog projekta

- 1. faza:
  - Izrada model checkera za CLI programe
    - safety svojstva
    - lazy apstrakcija (kao u BLAST-u)
  - Koristiti efikasne vanjske module
    - dokazivači teorema
    - paketi za rad s BDD-ovima
    - ...
  - Tehnički najzahtjevnije: analiza i instrumentacija CIL koda
    - možebitna dobra primjena Phoenixa

Idući četvrtak

- Što imaju zajedničko?
  - Ramseyev teorem
  - Dobro utemeljene relacije
  - Apstrakcija predikata
  - Liveness svojstva
  - i :