Neograničena provera modela softvera kroz inkrementalno SAT rešavanje

Ivan Ristović

Uvod

- Ograničena provera modela softvera
- Inspiracija za neograničenu proveru modela softvera
- Nov način kodiranja stanja programa omogućava uklanjanje granice koja postoji kod ograničene provere modela softvera
- Kodiranje stanja programa u *DimSpec* formulu korišćenjem alata *LLUMC*
- Korišćenje inkrementalnih SAT rešavača za nalaženje modela

Opis rada neograničene provere modela

- Sekvencijalni C kod
- Prevođenje u *LLVM* međureprezentacija C koda
- Kodiranje stanja programa u SMT formule
- Stanje greške predstavlja ciljno stanje
- Prevođenje SMT formule u *DimSpec* formulu (SAT)
- Korišćenje inkrementalnih SAT rešavača za nalaženje modela ili algoritme za nalaženje invarijanti (*IC3*)
- Model se koristi kao dokaz neispravnosti programa

LLVM reprezentacija

- *LLVM* je open-source compiler framework projekat.
- Za neograničenu proveru modela se koristi *LLVM* međureprezentacija C koda koja se naziva *LLVM modul*.
- *LLVM* modul se može grupisati u nekoliko jedinica:
  - instrukcija
  - osnovni blok
  - funkcija

Prostor stanja programa

- Stanja programa se posmatraju kao stanja svakog osnovnog bloka zasebno. Slično je i za tranzicije između stanja.
- Jedno stanje se sastoji od promenljivih.
- Svaka promenljiva se kodira kao bit-vektor dužine n. Svi bitovi zasebno se posmatraju kao promenljive $x_1, \ldots, x_n$.
- Uvodimo dva specijalna stanja:
  - *ok* stanje - iz ovog stanja ne mogu nastati greške
  - *error* stanje

Prostor stanja programa

- Ako je $|B|$ broj osnovnih blokova u programu, onda je $\lceil \log_2 |B| + 2 \rceil$ bitova potrebno za enkodiranje jednog osnovnog bloka.
- Promenljivu koja kodira trenutni blok označavamo sa *curr*.
- S obzirom da vrednosti promenljivih mogu da zavise od vrednosti u prethodnom bloku, uvodimo promenljivu *prev* koja kodira prethodni blok.

DimSpec formula

- Svako stanje $t_0, \ldots, t_k$ sadrži vrednosti promenljivih u tekućem osnovnom bloku.
- *DimSpec* formula $\mathcal{F}$ se sastoji od četiri SMT formule: $\mathcal{I}$, $\mathcal{U}$, $\mathcal{G}$ i $\mathcal{T}$.
- $\mathcal{I}$ predstavlja skup inicijalnih klauza, tj. onih klauza koje stanje t_0 zadovoljava.
- $\mathcal{G}$ predstavlja skup završnih klauza, tj. onih koje stanje t_k zadovoljava.
- $\mathcal{U}$ predstavlja skup klauza koje su zadovoljene stanjima t_i.
- $\mathcal{T}$ predstavlja skup klauza koje su zadovoljene parovima stanja t_i, t_{i+1}.

DimSpec formula

- Testiranje da li je završno stanje dostižno iz početnog u k koraka, je ekvivalentno ispitivanju zadovoljivosti formule F_k:

$$F_k = \mathcal{I}(0) \wedge \left(\bigwedge_{i=0}^{k-1} (\mathcal{U}(i) \wedge \mathcal{T}(i, i+1)) \right) \wedge \mathcal{U}(k) \wedge \mathcal{G}(k)$$

- Jedan od načina da se nađe najmanji broj koraka za koje se završno stanje dostiže iz polaznog je da se rešavaju formule $F_1, F_2, \ldots$ sve dok se ne nađe zadovoljiva formula.
- Efikasan način da se ovo implementira je korišćenjem *Incremental SAT-Solving-a*.

Prevođenje SMT u SAT

- Enkodiranje opisano iznad nam kao rezultat daje četiri SMT formule
- *bit-blasting*
- *LLUMC* automatski vrši ovu transformaciju

Inkrementalno SAT rešavanje

- Inkrementalno dodavanje klauza tokom rešavanja problema zadovoljivosti
- Funkcije *add(C)* i *solve(A)*, gde je C klauza a A skup literala koji se nazivaju pretpostavke (engl. *assumptions*)
- Klauze se dodaju korišćenjem funkcije *add* i njihova konjunkcija se rešava pod predpostavkom da su vrednosti svih literala iz A tačne, koristeći *solve(A)*
- Proces inkrementalnog SAT rešavanja za *DimSpec* formule:

$$
\begin{array}{lcl}
step(0) & : & \texttt{add}(\mathcal{I}(0) \wedge (a_0 \vee \mathcal{G}(0)) \wedge \mathcal{U}(0)) \\
 & & \texttt{solve}(assumptions = \neg a_0) \\
step(k) & : & \texttt{add}(\mathcal{T}(k-1,k) \wedge (a_k \vee \mathcal{G}(k)) \wedge \mathcal{U}(k)) \\
 & & \texttt{solve}(assumptions = \neg a_k)
\end{array}
$$

Ograničenja Unbounded Software Model Checking-a

- Šta je uopste greška u softveru?
- Nemoguće je pokriti sve moguće greške.
- Stoga se u nastavku ograničavamo na *prekoračenja*.
- Koristićemo funkcije `assume` i `assert`.
- Program se ponaša po specifikaciji ukoliko svi pozivi funkcije `assert` vrate `true` pod uslovom da su svi uslovi iz poziva `assume` zadovoljeni.

Definicija (Greška programa za alat LLUMC)

Neka je p program. Greška u programu p postoji ukoliko su svi pozivi funkcije **assume** *pre poziva funkcije* **assert** *ili mogućeg prekoračenja vratili* **true** *i važi jedno od navedenog:*

1. *Funckija* **assert** *je vratila* **false**.
2. *Desilo se prekoračenje prilikom izvršavanja aritmetičke operacije.*

Primer

```
1  int main(void)
2  {
3      unsigned int x = 4294967295-101;
4      while (x >= 10) {
5          x += 2;
6      }
7      __VERIFIER_assert(x % 2);
8  }
```

Primer - LLVM međureprezentacija

```
1  define i32 @main() #0 {
2  %1 = alloca i32, align 4
3  %x = alloca i32, align 4
4  store i32 0, i32* %1
5  store i32 4294967194, i32* %x, align 4
6  br label %2
7
8  ; <label>:2                                     ; preds = %5, %0
9  %3 = load i32, i32* %x, align 4
10 %4 = icmp uge i32 %3, 10
11 br i1 %4, label %5, label %8
12 12
13 ; <label>:5                                     ; preds = %2
14 %6 = load i32, i32* %x, align 4
15 %7 = add i32 %6, 2
16 store i32 %7, i32* %x, align 4
17 br label %2
18 18
19 ; <label>:8                                     ; preds = %2
20 %9 = load i32, i32* %x, align 4
21 %10 = urem i32 %9, 2
22 call void @__VERIFIER_assert(i32 %10)
23 %11 = load i32, i32* %1
24 t i32 %11
25 }
```

Primer - Optimizovana LLVM međureprezentacija

```
1  define i32 @main() #0
2  entry:
3  %tmp = icmp uge i32 4294967194, 10
4  br i1 %tmp, label %bb1, label %return
5
6  bb1:                                      ; preds = %entry, %bb1
7  %x.0 = phi i32 [ %tmp2, %bb1 ], [ 4294967194, %entry ]
8  %tmp2 = add i32 %x.0, 2
9  %tmp3 = icmp uge i32 %tmp2, 10
10 br i1 %tmp3, label %bb1, label %return
11
12 return:                                   ; preds = %bb1, %entry
13 %x.1 = phi i32 [ 4294967194, %entry ], [ %tmp2, %bb1 ]
14 %tmp4 = urem i32 %x.1, 2
15 %tmp.i = icmp ne i32 %tmp4, 0
16 br i1 %tmp.i, label %__VERIFIER_assert.exit, label %bb1.i
17
18 bb1.i:                                    ; preds = %return
19 call void bitcast (void (...)* @__VERIFIER_error to void ()*)() #3
20 unreachable
21
22 __VERIFIER_assert.exit:                   ; preds = %return
23 ret i32 0
```

Primer - Kodiranje stanja

$$entry \rightarrow 1$$

$$bb1.lr.ph \rightarrow 2$$

$$bb1 \rightarrow 3$$

$$bb.return\_crit\_edge \rightarrow 4$$

$$return \rightarrow 5$$

$$bb.1 \rightarrow 6$$

$$\_\_VERIFIER\_assert\_exist \rightarrow 7$$

$$ok \rightarrow 8$$

$$error \rightarrow 9$$

Primer - Kodiranje skupova $\mathcal{I}, \mathcal{G}, \mathcal{U}$

$$\mathcal{I} = \{curr = 1 \wedge prev = 1\}$$

$$\mathcal{G} = \{curr = 9\}$$

$$\mathcal{U} = \{curr <= 9 \wedge prev <= 9\}$$

Primer - Kodiranje skupa $\mathcal{T}$

$$
\begin{aligned}
\mathcal{T} = \{ & \\
& (curr = 1 \wedge prev = 1 \Rightarrow curr' = 9 \wedge prev' = 9 \wedge tmp2' = tmp2) \\
\wedge \quad & (curr = 2 \Rightarrow curr = 3 \wedge prev' = 2 \wedge tmp2' = tmp2) \\
\wedge \quad & (curr = 3 \wedge 10 \leq (2 + ite(prev = 2, 10, tmp2)) \\
& \Rightarrow curr' = 3 \wedge prev' = 3 \wedge tmp2' = (2 + ite(prev = 2, 10, tmp2))) \\
\wedge \quad & (curr = 3 \wedge 10 > (2 + ite(prev = 2, 10, tmp2)) \\
& \Rightarrow curr' = 4 \wedge prev' = 3 \wedge tmp2' = (2 + ite(prev = 2, 10, tmp2))) \\
\wedge \quad & (curr = 4 \Rightarrow curr' = 5 \wedge prev' = 4 \wedge tmp2' = tmp2) \\
\wedge \quad & (curr = 5 \wedge 0 \neq bvmod(ite(prev = 4, tmp2, 10), 2) \\
& \Rightarrow curr' = 7 \wedge prev' = 5 \wedge tmp2' = tmp2) \\
\wedge \quad & (curr = 5 \wedge 0 = bvmod(ite(prev = 4, tmp2, 10), 2) \\
& \Rightarrow curr' = 6 \wedge prev' = 5 \wedge tmp2' = tmp2) \\
\wedge \quad & (curr = 6 \Rightarrow curr' = 9 \wedge prev' = 6 \wedge tmp2' = tmp2) \\
\wedge \quad & (curr = 7 \Rightarrow curr' = 8 \wedge prev' = 7 \wedge tmp2' = tmp2) \\
\wedge \quad & (curr = 8 \Rightarrow curr' = 8 \wedge prev' = 8 \wedge tmp2' = tmp2) \\
\wedge \quad & (curr = 9 \Rightarrow curr' = 9 \wedge prev' = 7 \wedge tmp2' = tmp2) \\
& \}
\end{aligned}
$$

Pitanja

???