



Angelic Verification

Precise Verification Modulo Unknowns

Jan Tužil

19. března 2018





Motivace

Detaily



Uzavřený program

Otevřený program - plané poplachy

```

1 // inconsistency
2 procedure Bar(x:int) {
3   if (x != NULL) { gs := 1; }
4   else { gs := 2; }
5   // possible BUG or dead code
6   assert x != NULL;
7   m[x] := 5;
8 }
9 // internal bug
10 procedure Baz(y:int) {
11   assert y != NULL; //DEFINITE BUG
12   m[y] := 4;
13 }
14 // entry point
15 procedure Foo(z:int) {
16   call Bar(z); //block + relax
17   call Baz(NULL); // internal bug
18   call FooBar(); // external calls
19 }

```

```

20 // globals
21 var gs:int, m:[int]int;
22
23 // external call
24 procedure FooBar() {
25   var x, w, z:int;
26   call z := Lib1();
27   assert z != NULL;
28   m[z] := NULL;
29   call x := Lib2();
30   assert x != NULL;
31   w := m[x];
32   assert w != NULL;
33   m[w] := 4;
34 }
35 // library
36 procedure Lib1() returns (r:int);
37 procedure Lib2() returns (r:int);

```



Některé stopy programu

```
1 push; // call bar
2 gs := 2;
3 assert x != null;
4 fail;
```

```
1 int gs;
2 void bar(int *x) {
3     if (x != nullptr) gs = 1;
4     else gs = 2;
5     *x = 5;
6 }
7
8 void baz(int *y) {
9     *y = 4;
10 }
11
12 void foo(int *z) {
13     bar(z);
14     baz(nullptr);
15 }
16
17 int ** Lib1();
18 int ** Lib2();
19 void FooBar() {
```

Cíl

- cíl: prioritizace důležitějších alarmů
- metoda: angelická verifikace (i.e. abduktivní inference)

Definice (Problém angelické verifikace)

Pro daný assert, existuje rozumná specifikace nad neznámými hodnotami taková, že daný assert platí?



Přijatelná specifikace

Chceme, aby přijatelná specifikace byla:

- stručná
- shovívavá

Shovívavá vstupní podmínka

Definice (Shovívavá vstupní podmínka)

Formule ϕ je shovívavá vstupní podmínka programu $P_{A, \hat{A}}$, značíme $\text{Permissive}(P_{A, \hat{A}}, \Phi)$, pokud pro každý andělský assert $s \in \hat{A}$ platí: pokud $\phi \models P_{\emptyset, \{s\}}$, pak $\text{true} \models P_{\emptyset, \{s\}}$.

Shovívavá vstupní podmínka

Definice (Shovívavá vstupní podmínka)

Formule ϕ je shovívavá vstupní podmínka programu $P_{A, \hat{A}}$, značíme $\text{Permissive}(P_{A, \hat{A}}, \Phi)$, pokud pro každý andělský assert $s \in \hat{A}$ platí: pokud $\phi \models P_{\emptyset, \{s\}}$, pak $\text{true} \models P_{\emptyset, \{s\}}$.

Jak to říci jinak?

Shovívavá vstupní podmínka

Definice (Shovívavá vstupní podmínka)

Formule ϕ je shovívavá vstupní podmínka programu $P_{A, \hat{A}}$, značíme $\text{Permissive}(P_{A, \hat{A}}, \Phi)$, pokud pro každý andělský assert $s \in \hat{A}$ platí: pokud $\phi \models P_{\emptyset, \{s\}}$, pak $\text{true} \models P_{\emptyset, \{s\}}$.

Jak to říci jinak?

Jak vypadají shovívavé vstupní podmínky programu, který obsahuje andělský

Shovívavá vstupní podmínka

Definice (Shovívavá vstupní podmínka)

Formule ϕ je shovívavá vstupní podmínka programu $P_{A, \hat{A}}$, značíme $\text{Permissive}(P_{A, \hat{A}}, \Phi)$, pokud pro každý andělský assert $s \in \hat{A}$ platí: pokud $\phi \models P_{\emptyset, \{s\}}$, pak $\text{true} \models P_{\emptyset, \{s\}}$.

Jak to říci jinak?

Jak vypadají shovívavé vstupní podmínky programu, který obsahuje andělský

- assert false na začátku programu?

Shovívavá vstupní podmínka

Definice (Shovívavá vstupní podmínka)

Formule ϕ je shovívavá vstupní podmínka programu $P_{A, \hat{A}}$, značíme $\text{Permissive}(P_{A, \hat{A}}, \Phi)$, pokud pro každý andělský assert $s \in \hat{A}$ platí: pokud $\phi \models P_{\emptyset, \{s\}}$, pak $\text{true} \models P_{\emptyset, \{s\}}$.

Jak to říci jinak?

Jak vypadají shovívavé vstupní podmínky programu, který obsahuje andělský

- assert false na začátku programu?
- assert false na konci každého bloku?

Shovívavá vstupní podmínka

Definice (Shovívavá vstupní podmínka)

Formule ϕ je shovívavá vstupní podmínka programu $P_{A, \hat{A}}$, značíme $\text{Permissive}(P_{A, \hat{A}}, \Phi)$, pokud pro každý andělský assert $s \in \hat{A}$ platí: pokud $\phi \models P_{\emptyset, \{s\}}$, pak $\text{true} \models P_{\emptyset, \{s\}}$.

Jak to říci jinak?

Jak vypadají shovívavé vstupní podmínky programu, který obsahuje andělský

- assert false na začátku programu?
- assert false na konci každého bloku?
- assert $x \neq v$ někde?

Andělská korektnost

Definice

Mějme program P obsahující sadu běžných assertů A a sadu andělských assertů \hat{A} , spolu se slovníkem formulí Vocab . Říkáme, že P je andělsky korektní za předpokladu (Vocab, \hat{A}) , pokud existuje formule $\phi \in \text{Vocab}$, která je shovívavou vstupní podmínkou programu P , a přitom $\phi \models P_{A, \emptyset}$.



AngelicVerifier

AngelicVerifier

```
1:  $E \leftarrow \emptyset$ 
2:  $\mathcal{A}_1 \leftarrow \mathcal{A}$ 
3: loop
4:    $\tau \leftarrow \text{Verify}(P_{\mathcal{A}_1, \emptyset}, E)$  /*  $E \models P$  */
5:   if  $\tau = \text{NO\_TRACE}$  then
6:     return  $(E, \mathcal{A}_1)$ 
7:   end if
8:    $\phi \leftarrow \text{ExplainError}(P, \tau, \text{Vocab})$ 
9:    $E_1 \leftarrow E \cup \{\phi\}$ 
10:  if  $\neg \text{Permissive}(P_{\emptyset, \mathcal{A}}, E_1)$  then
11:    Let  $a$  be the failing assert in  $\tau$ 
12:     $\mathcal{A}_1 \leftarrow \mathcal{A}_1 \setminus \{a\}$  /* Report  $a$  */
13:  else
14:     $E \leftarrow E_1$ 
15:  end if
16: end loop
```

AngelicVerifier

```

1:  $E \leftarrow \emptyset$ 
2:  $\mathcal{A}_1 \leftarrow \mathcal{A}$ 
3: loop
4:    $\tau \leftarrow \text{Verify}(P_{\mathcal{A}_1, \emptyset}, E)$  /*  $E \models P$  */
5:   if  $\tau = \text{NO\_TRACE}$  then
6:     return  $(E, \mathcal{A}_1)$ 
7:   end if
8:    $\phi \leftarrow \text{ExplainError}(P, \tau, \text{Vocab})$ 
9:    $E_1 \leftarrow E \cup \{\phi\}$ 
10:  if  $\neg \text{Permissive}(P_{\emptyset, \mathcal{A}}, E_1)$  then
11:    Let  $a$  be the failing assert in  $\tau$ 
12:     $\mathcal{A}_1 \leftarrow \mathcal{A}_1 \setminus \{a\}$  /* Report  $a$  */
13:  else
14:     $E \leftarrow E_1$ 
15:  end if
16: end loop

```

- Vstupy: program P s obyčejnými asserty A a andělskými asserty \hat{A} .

AngelicVerifier

```

1:  $E \leftarrow \emptyset$ 
2:  $\mathcal{A}_1 \leftarrow \mathcal{A}$ 
3: loop
4:    $\tau \leftarrow \text{Verify}(P_{\mathcal{A}_1, \emptyset}, E) \text{ /* } E \models P \text{ */}$ 
5:   if  $\tau = \text{NO\_TRACE}$  then
6:     return  $(E, \mathcal{A}_1)$ 
7:   end if
8:    $\phi \leftarrow \text{ExplainError}(P, \tau, \text{Vocab})$ 
9:    $E_1 \leftarrow E \cup \{\phi\}$ 
10:  if  $\neg \text{Permissive}(P_{\emptyset, \mathcal{A}}, E_1)$  then
11:    Let  $a$  be the failing assert in  $\tau$ 
12:     $\mathcal{A}_1 \leftarrow \mathcal{A}_1 \setminus \{a\}$  /* Report  $a$  */
13:  else
14:     $E \leftarrow E_1$ 
15:  end if
16: end loop

```

- Vstupy: program P s obyčejnými asserty A a andělskými asserty \hat{A} .
- Výstupy: shovívavá specifikace E a množina platících obyčejných assertů $A_1 \subseteq A$.

AngelicVerifier

```

1:  $E \leftarrow \emptyset$ 
2:  $\mathcal{A}_1 \leftarrow \mathcal{A}$ 
3: loop
4:    $\tau \leftarrow \text{Verify}(P_{\mathcal{A}_1, \emptyset}, E)$  /*  $E \models P$  */
5:   if  $\tau = \text{NO\_TRACE}$  then
6:     return  $(E, \mathcal{A}_1)$ 
7:   end if
8:    $\phi \leftarrow \text{ExplainError}(P, \tau, \text{Vocab})$ 
9:    $E_1 \leftarrow E \cup \{\phi\}$ 
10:  if  $\neg \text{Permissive}(P_{\emptyset, \mathcal{A}}, E_1)$  then
11:    Let  $a$  be the failing assert in  $\tau$ 
12:     $\mathcal{A}_1 \leftarrow \mathcal{A}_1 \setminus \{a\}$  /* Report  $a$  */
13:  else
14:     $E \leftarrow E_1$ 
15:  end if
16: end loop

```

- Vstupy: program P s obyčejnými asserty A a andělskými asserty \hat{A} .
- Výstupy: shovívavá specifikace E a množina platících obyčejných assertů $A_1 \subseteq A$.
- ϕ - formule blokující chybový běh τ

ExplainError

```

1:  $\tau_1 \leftarrow \text{ControlSlice}(P, \tau)$ 
2:  $\phi_1 \leftarrow \text{wlp}(\tau_1, \text{true})$ 
3:  $\phi_2 \leftarrow \text{EliminateMapUpdates}(\phi_1)$ 
4:  $\text{atoms}_1 \leftarrow \text{FilterAtoms}(\text{GetAtoms}(\phi_2),$ 
    $\text{Vocab.Atoms})$ 
5: if  $\text{Vocab.Bool} = \text{MONOMIAL}$  then
6:    $S \leftarrow \{a \mid a \in \text{atoms}_1, \text{ and } a \models \phi_2\}$ 
7:   return  $S = \emptyset ? \text{false} : (\bigvee_{a \in S} a)$ 
8: else
9:   return  $\text{ProjectAtoms}(\phi_2, \text{atoms}_1)$ 
10: end if

```

Explanation

Příklad

```
24 procedure FooBar() {  
25     var x, w, z: int;  
26     call z := Lib1 ();  
27     assert z != NULL;  
28     m[z] := NULL;  
29     call x := Lib2 ();  
30     assert x != NULL;  
31     w := m[x];  
32     assert w != NULL;  
33     m[w] := 4;  
34 }
```

Příklad

```
24 procedure FooBar() {  
25     var x, w, z: int ;  
26     call z := Lib1 ();  
27     assert z != NULL;  
28     m[z] := NULL;  
29     call x := Lib2 ();  
30     assert x != NULL;  
31     w := m[x];  
32     assert w != NULL;  
33     m[w] := 4;  
34 }
```

Stopa τ :

```
1  z := x_1 ;  
2  m[z] := NULL ;  
3  x := x_2 ;  
4  w := m[x] ;  
5  assert w != NULL ;
```

- volání knihovních procedur nahrazena novými proměnnými



Příklad

Stopa τ :

```
1 z := x_1;  
2 m[z] := NULL;  
3 x := x_2;  
4 w := m[x];  
5 assert w != NULL;
```


Příklad

Stopa τ :

```
1 z := x_1;  
2 m[z] := NULL;  
3 x := x_2;  
4 w := m[x];  
5 assert w != NULL;
```

$wlp(\tau, true)$:

```
1 read(write(m, x_1, NULL), x_2)
```

Příklad

Stopa τ :

```
1 z := x_1;  
2 m[z] := NULL;  
3 x := x_2;  
4 w := m[x];  
5 assert w != NULL;
```

$wlp(\tau, true)$:

```
1 read(write(m, x_1, NULL), x_2)
```

EliminateMapUpdates:

```
1 x_2 == x_1 ? NULL : read(m, x_1)
```