FAKULTA
INFORMATIKY
Masarykova univerzita

Angelic Verification

Precise Verification Modulo Unknowns

Jan Tušil

20. března 2018

Motivace

Motivace

Detaily

FAKULTA
INFORMATIKY
Masarykova univerzita

Osnova

FAKULTA INFORMATIKY
Masarykova univerzita

Uzavřený program

FAKULTA INFORMATIKY
Masarykova univerzita

Otevřený program - plané poplachy

```
// inconsistency
procedure Bar(x:int) {
      if (x != NULL) { gs := 1; }
      else { gs := 2; }
      // possible BUG or dead code
      assert x != NULL;
      m[x] := 5;
}
// internal bug
procedure Baz(y:int) {
      assert y != NULL; //DEFINITE BUG
      m[y] := 4;
}
// entry point
procedure Foo(z:int) {
      call Bar(z);        // block + relax
      call Baz(NULL); // internal bug
      call FooBar();      // external calls
}
// globals
var gs: int, m:[int]int;

// external call
procedure FooBar() {
      var x, w, z: int;
      call z := Lib1();
      assert z != NULL;
      m[z] := NULL;
      call x := Lib2();
      assert x != NULL;
      w := m[x];
      assert w != NULL;
      m[w] := 4;
}
// library
procedure Lib1() returns (r: int);
procedure Lib2() returns (r: int);
```

FAKULTA INFORMATIKY
Masarykova univerzita

Některé stopy programu

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

FAKULTA INFORMATIKY
Masarykova univerzita

```
int gs;
void bar(int *x) {
  if (x != nullptr) gs = 1;
  else gs = 2;
  *x = 5;
}

void baz(int *y) {
  *y = 4;
}

void foo(int *z) {
  bar(z);
  baz(nullptr);
}

int ** Lib1();
int ** Lib2();
void FooBar() {
  *Lib1() = NULL;
  **Lib2() = 4;
}
```

FAKULTA INFORMATIKY
Masarykova univerzita

Cíl

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

Definice (Problém angelické verifikace)

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

Chceme, aby přijatelná specifikace byla:

- stručná
- shovívavá

FAKULTA INFORMATIKY
Masarykova univerzita

Osnova

FAKULTA INFORMATIKY
Masarykova univerzita

Jazyk

$$
\begin{array}{llll}
P & \in \mathit{Program} & ::= & \mathit{Block}^+ \\
BL & \in \mathit{Block} & ::= & \mathit{BlockId} : s;\ \textsf{goto}\ \mathit{BlockId}^* \\
s, t & \in \mathit{Stmt} & ::= & \textsf{skip} \mid \textsf{assert}\ \phi \mid \textsf{assume}\ \phi \mid \mathtt{x} := e \mid \textsf{havoc}\ \mathtt{x} \mid s; s \\
\mathtt{x}, \mathtt{y} & \in \mathit{Vars} & & \\
e & \in \mathit{Expr} & ::= & \mathtt{x} \mid f(e, \ldots, e) \\
\phi, \psi & \in \mathit{Formula} & ::= & \textsf{true} \mid \textsf{false} \mid p(e, \ldots, e) \mid \phi \wedge \phi \mid \forall x :\ \phi \mid \neg\phi
\end{array}
$$

Sémantika

FAKULTA INFORMATIKY
Masarykova univerzita

Sémantika

- stavy: $\Sigma = (Vars \to \mathcal{Z}) \cup \{Err\}$

FAKULTA
INFORMATIKY
Masarykova univerzita

Sémantika

- stavy: $\Sigma = (Vars \to \mathcal{Z}) \cup \{Err\}$
- běhy: $Traces = \mathcal{N}_0 \hookrightarrow \Sigma$

FAKULTA
INFORMATIKY
Masarykova univerzita

Sémantika

- stavy: $\Sigma = (Vars \to \mathcal{Z}) \cup \{Err\}$
- běhy: $Traces = \mathcal{N}_0 \hookrightarrow \Sigma$
- Sémantika $\mathcal{T} = Program \to 2^{Traces}$ dle očekávání

FAKULTA INFORMATIKY
Masarykova univerzita

Sémantika

- stavy: $\Sigma = (\mathit{Vars} \to \mathcal{Z}) \cup \{\mathit{Err}\}$
- běhy: $\mathit{Traces} = \mathcal{N}_0 \hookrightarrow \Sigma$
- Sémantika $\mathcal{T} = \mathit{Program} \to 2^{\mathit{Traces}}$ dle očekávání
- selhaný `assert(E)` - nekonečný chybový běh běh (Err^ω)

Sémantika

- stavy: $\Sigma = (\mathit{Vars} \to \mathcal{Z}) \cup \{\mathit{Err}\}$
- běhy: $\mathit{Traces} = \mathcal{N}_0 \hookrightarrow \Sigma$
- Sémantika $\mathcal{T} = \mathit{Program} \to 2^{\mathit{Traces}}$ dle očekávání
- selhaný `assert(E)` - nekonečný chybový běh běh (Err^ω)
- selhaný `assume(E)` - nekonečný nechybový běh.

Sémantika

- stavy: $\Sigma = (Vars \to \mathcal{Z}) \cup \{Err\}$
- běhy: $Traces = \mathcal{N}_0 \hookrightarrow \Sigma$
- Sémantika $\mathcal{T} = Program \to 2^{Traces}$ dle očekávání
- selhaný `assert(E)` - nekonečný chybový běh běh (Err^{ω})
- selhaný `assume(E)` - nekonečný nechybový běh.

Definice

Program P je korektní, píšeme $\vDash P$, pokud $\mathcal{T}(P)$ neobsahuje stop obsahující stav Err.

FAKULTA INFORMATIKY
Masarykova univerzita

Ukázka

Co počítá tento program?

```
start:
  sum := 0;
  i := n;
  goto cycle end

cycle:
  assume i > 0;
  sum *= i;
  i--;
  goto cycle end

end:
  assume i <= 0;
  goto
```

FAKULTA INFORMATIKY
Masarykova univerzita

Ukázka

Co počítá tento program?

```
start:
  sum := 0;
  i := n;
  goto cycle end

cycle:
  assume i > 0;
  sum *= i;
  i--;
  goto cycle end

end:
  assume i <= 0;
  goto
```

Řídící struktury: kód ve tvaru

```
if (E) { S } else { T }
```

lze přepsat jako

```
Start: goto Then, Else
Then: assume E; S; goto End
Else: assume !E; T; goto End
End: /* ... */
```

FAKULTA INFORMATIKY
Masarykova univerzita

Korektnost se vstupní podmínkou

Definice

Program $P \in$ Program je korektní se vstupní podmínkou $\phi \in$ Formula, píšeme $\phi \vDash P$, pokud je korektní program:

```
Start0 : assume phi; goto Start
```

FAKULTA INFORMATIKY
Masarykova univerzita

Korektnost se vstupní podmínkou

Definice

Program $P \in Program$ je korektní se vstupní podmínkou $\phi \in Formula$, píšeme $\phi \models P$, pokud je korektní program:

```
Start0 : assume phi; goto Start
```

Definice

Nechť A je množina obyčejných assertů v programu P, a nechť $\hat{A}$ je množina andělských assertů uživatelem přidaných do programu P. Výrazem P_{A_1, A_2} označujeme instrumentovanou verzi programu P , která má povolené pouze obyčejné asserty $A_1 \in A$ a andělské assert $A_2 \in \hat{A}$.

FAKULTA INFORMATIKY
Masarykova univerzita

Shovívavá vstupní podmínka

Definice (Shovívavá vstupní podmínka)

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

FAKULTA INFORMATIKY
Masarykova univerzita

Shovívavá vstupní podmínka

Definice (Shovívavá vstupní podmínka)

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

Jak to říci jinak?

FAKULTA INFORMATIKY
Masarykova univerzita

Shovívavá vstupní podmínka

Definice (Shovívavá vstupní podmínka)

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

Jak to říci jinak?
Jak vypadají shovívavé vstupní podmínky programu, který obsahuje andělský

FAKULTA INFORMATIKY
Masarykova univerzita

Shovívavá vstupní podmínka

Definice (Shovívavá vstupní podmínka)

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

Jak to říci jinak?
Jak vypadají shovívavé vstupní podmínky programu, který obsahuje andělský

- `assert false` na začátku programu?

FAKULTA INFORMATIKY
Masarykova univerzita

Shovívavá vstupní podmínka

Definice (Shovívavá vstupní podmínka)

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

Jak to říci jinak?
Jak vypadají shovívavé vstupní podmínky programu, který obsahuje andělský

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

FAKULTA INFORMATIKY
Masarykova univerzita

Shovívavá vstupní podmínka

Definice (Shovívavá vstupní podmínka)

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

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 != v` někde?

FAKULTA INFORMATIKY
Masarykova univerzita

Andělská korektnost

Definice

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

FAKULTA
INFORMATIKY
Masarykova univerzita

AngelicVerifier

FAKULTA INFORMATIKY
Masarykova univerzita

AngelicVerifier

$E \leftarrow \emptyset$
$\mathcal{A}_1 \leftarrow \mathcal{A}$
loop
    $\tau \leftarrow \mathit{Verify}(P_{\mathcal{A}_1,\emptyset}, E)$ /* $E \models P$ */
    if $\tau = \texttt{NO\_TRACE}$ **then**
        return $(E, \mathcal{A}_1)$
    end if
    $\phi \leftarrow \mathit{ExplainError}(P, \tau, \mathit{Vocab})$
    $E_1 \leftarrow E \cup \{\phi\}$
    if $\neg \mathit{Permissive}(P_{\emptyset,\hat{\mathcal{A}}}, E_1)$ **then**
        Let a be the failing assert in τ
        $\mathcal{A}_1 \leftarrow \mathcal{A}_1 \setminus \{a\}$ /* Report a */
    else
        $E \leftarrow E_1$
    end if
end loop

AngelicVerifier

$E \leftarrow \emptyset$
$\mathcal{A}_1 \leftarrow \mathcal{A}$
loop
    $\tau \leftarrow Verify(P_{\mathcal{A}_1,\emptyset}, E)$ /* $E \models P$ */
    if $\tau =$ NO_TRACE **then**
        return $(E, \mathcal{A}_1)$
    end if
    $\phi \leftarrow ExplainError(P, \tau, Vocab)$
    $E_1 \leftarrow E \cup \{\phi\}$
    if $\neg Permissive(P_{\emptyset,\hat{\mathcal{A}}}, E_1)$ **then**
        Let a be the failing assert in τ
        $\mathcal{A}_1 \leftarrow \mathcal{A}_1 \setminus \{a\}$ /* Report a */
    else
        $E \leftarrow E_1$
    end if
end loop

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

AngelicVerifier

$E \leftarrow \emptyset$
$\mathcal{A}_1 \leftarrow \mathcal{A}$
loop
    $\tau \leftarrow \mathit{Verify}(P_{\mathcal{A}_1,\emptyset}, E)$ /* $E \models P$ */
    if $\tau =$ NO_TRACE **then**
        return $(E, \mathcal{A}_1)$
    end if
    $\phi \leftarrow \mathit{ExplainError}(P, \tau, \mathit{Vocab})$
    $E_1 \leftarrow E \cup \{\phi\}$
    if $\neg \mathit{Permissive}(P_{\emptyset,\hat{\mathcal{A}}}, E_1)$ **then**
        Let a be the failing assert in τ
        $\mathcal{A}_1 \leftarrow \mathcal{A}_1 \setminus \{a\}$ /* Report a */
    else
        $E \leftarrow E_1$
    end if
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

$E \leftarrow \emptyset$
$\mathcal{A}_1 \leftarrow \mathcal{A}$
loop
  $\tau \leftarrow \mathit{Verify}(P_{\mathcal{A}_1,\emptyset}, E)$ /* $E \models P$ */
  if $\tau = \texttt{NO\_TRACE}$ **then**
    return $(E, \mathcal{A}_1)$
  end if
  $\phi \leftarrow \mathit{ExplainError}(P, \tau, \mathit{Vocab})$
  $E_1 \leftarrow E \cup \{\phi\}$
  if $\neg \mathit{Permissive}(P_{\emptyset,\hat{\mathcal{A}}}, E_1)$ **then**
    Let a be the failing assert in τ
    $\mathcal{A}_1 \leftarrow \mathcal{A}_1 \setminus \{a\}$ /* Report a */
  else
    $E \leftarrow E_1$
  end if
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

FAKULTA INFORMATIKY
Masarykova univerzita

ExplainError

$\tau_1 \leftarrow \mathit{ControlSlice}(P, \tau)$
$\phi_1 \leftarrow \mathit{wlp}(\tau_1, \mathsf{true})$
$\phi_2 \leftarrow \mathit{EliminateMapUpdates}(\phi_1)$
$\mathit{atoms_1} \leftarrow \mathit{FilterAtoms}\ (\mathit{GetAtoms}(\phi_2), \mathit{Vocab.Atoms})$
if $\mathit{Vocab.Bool} = \mathtt{MONOMIAL}$ **then**
    $S \leftarrow \{a \mid a \in \mathit{atoms_1},\ \mathit{and}\ a \models \phi_2\}$
    return $S = \emptyset\ ?\ \mathsf{false}\ :\ (\bigvee_{a \in S} a)$
else
    return $\mathit{ProjectAtoms}(\phi_2, \mathit{atoms_1})$
end if

Explanation

FAKULTA INFORMATIKY
Masarykova univerzita

Příklad

```
procedure FooBar() {
    var x, w, z : int ;
    call z := Lib1 ();
    assert z != NULL;
    m[z] := NULL;
    call x := Lib2 ();
    assert x != NULL;
    w := m[x];
    assert w != NULL;
    m[w] := 4;
}
```

Příklad

```
procedure FooBar() {
    var x, w, z: int;
    call z := Lib1();
    assert z != NULL;
    m[z] := NULL;
    call x := Lib2();
    assert x != NULL;
    w := m[x];
    assert w != NULL;
    m[w] := 4;
}
```

Stopa τ:

```
z := x_1;
m[z] := NULL;
x := x_2;
w := m[x];
assert w != NULL;
```

- volání knihovních procedur nahrazena novými proměnnými

FAKULTA INFORMATIKY
Masarykova univerzita

Příklad

Stopa τ:

```
z := x_1;
m[z] := NULL;
x := x_2;
w := m[x];
assert w != NULL;
```

FAKULTA INFORMATIKY
Masarykova univerzita

Příklad

Stopa τ:

```
z := x_1;
m[z] := NULL;
x := x_2;
w := m[x];
assert w != NULL;
```

$wlp(\tau, true)$:

```
w != NULL
read(m, x) != NULL
read(m, x2) != NULL
read(write(m, z, NULL), x2) != NULL
read(write(m, x_1, NULL), x2) != NULL
```

```
read(write(m, x_1, NULL), x_2) != NULL
```

FAKULTA INFORMATIKY
Masarykova univerzita

Příklad

Stopa τ:

```
z := x_1;
m[z] := NULL;
x := x_2;
w := m[x];
assert w != NULL;
```

$wlp(\tau, true)$:

```
w != NULL
read(m, x) != NULL
read(m, x2) != NULL
read(write(m, z, NULL), x2) != NULL
read(write(m, x_1, NULL), x2) != NULL
```

```
read(write(m, x_1, NULL), x_2) != NULL
```

EliminateMapUpdates:


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