FAKULTA
INFORMATIKY
Masarykova univerzita

Angelic Verification

Precise Verification Modulo Unknowns

Jan Tušil

23. března 2018

Příklad

```
var m:[int]int;
```

Příklad

```
var m:[int]int;

// entry point
procedure Foo(z:int) {
  call Baz(NULL);
}
```

Příklad

```
var m:[int]int;

// entry point
procedure Foo(z:int) {
  call Baz(NULL);
}

procedure Baz(y:int) {
  assert y != NULL;
  m[y] := 4;
}
```

Příklad

```
var m:[int]int;

// entry point
procedure Foo(z:int) {
  call Baz(NULL);
}

procedure Baz(y:int) {
  assert y != NULL; // 100% bug
  m[y] := 4;
}
```

Jiný příklad

```
var gs:int, m:[int]int;
```

Jiný příklad

```
var gs:int, m:[int]int;

// entry point
procedure Foo(z:int) {
  call Bar(z);
}
```

Jiný příklad

```
var gs:int, m:[int]int;

// entry point
procedure Foo(z:int) {
  call Bar(z);
}

procedure Bar(x:int) {
  if (x != NULL) { gs := 1; }
  else { gs := 2; }
  assert x != NULL;
  m[x] := 5;
}
```

Jiný příklad

```
var gs:int, m:[int]int;

// entry point
procedure Foo(z:int) {
  call Bar(z);
}

procedure Bar(x:int) {
  if (x != NULL) { gs := 1; }
  else { gs := 2; }
  assert x != NULL; // bug
  m[x] := 5;
}
```

Jiný příklad

```
var gs:int, m:[int]int;
// precondition: z != NULL
// entry point
procedure Foo(z:int) {
  call Bar(z);
}

procedure Bar(x:int) {
  if (x != NULL) { gs := 1; }
  else { gs := 2; }
  assert x != NULL; // bug
  m[x] := 5;
}
```

Jiný příklad

```
var gs:int, m:[int]int;
// precondition: z != NULL
// entry point
procedure Foo(z:int) {
  call Bar(z);
}

procedure Bar(x:int) {
  if (x != NULL) { gs := 1; }
  else { gs := 2; }
  assert x != NULL; // ok due to precondition
  m[x] := 5;
}
```

Jiný příklad

```
var gs:int, m:[int]int;
// precondition: z != NULL
// entry point
procedure Foo(z:int) {
  call Bar(z);
}

procedure Bar(x:int) {
  if (x != NULL) { gs := 1; // unreachable }
  else { gs := 2; }
  assert x != NULL; // ok due to precondition
  m[x] := 5;
}
```

Jiný příklad

```
var gs:int, m:[int]int;
// precondition: z != NULL
// entry point
procedure Foo(z:int) {
  call Bar(z);
}
// inconsistent
procedure Bar(x:int) {
  if (x != NULL) { gs := 1; // unreachable }
  else { gs := 2; }
  assert x != NULL; // ok due to precondition
  m[x] := 5;
}
```

Do třetice ...

```
var m:[int]int;
```

Do třetice ...

```
var m:[int]int;

// library
procedure Lib1() {
  returns (r:int);

procedure Lib2()
  returns (r:int);
```

Do třetice ...

```
var m:[int]int;

// library
procedure Lib1() {
  returns (r:int);

procedure Lib2()
  returns (r:int);


// entry point
procedure Foo(z:int) {
  call FooBar();
}
```

Do třetice ...

```
var m:[int]int;

// library
procedure Lib1() {
  returns (r:int);

procedure Lib2()
  returns (r:int);


// entry point
procedure Foo(z:int) {
  call FooBar();
}
```

```
procedure FooBar() {
  var x, w, z:int

}
```

Do třetice ...

```
var m:[int]int;

// library
procedure Lib1() {
  returns (r:int);

procedure Lib2()
  returns (r:int);


// entry point
procedure Foo(z:int) {
  call FooBar();
}
```

```
procedure FooBar() {
  var x, w, z:int
  call z := Lib1();

}
```

Do třetice ...

```
var m:[int]int;

// library
procedure Lib1() {
  returns (r:int);

procedure Lib2()
  returns (r:int);


// entry point
procedure Foo(z:int) {
  call FooBar();
}
```

```
procedure FooBar() {
  var x, w, z:int
  call z := Lib1();
  assert z != NULL;



}
```

Do třetice ...

```
var m:[int]int;

// library
procedure Lib1() {
  returns (r:int)
  ensures (r != NULL);
procedure Lib2()
  returns (r:int);


// entry point
procedure Foo(z:int) {
  call FooBar();
}
```

```
procedure FooBar() {
  var x, w, z:int
  call z := Lib1();
  assert z != NULL;



}
```

Do třetice ...

```
var m:[int]int;

// library
procedure Lib1() {
  returns (r:int)
  ensures (r != NULL);
procedure Lib2()
  returns (r:int);


// entry point
procedure Foo(z:int) {
  call FooBar();
}
```

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


}
```

Do třetice ...

```
var m:[int]int;

// library
procedure Lib1() {
  returns (r:int)
  ensures (r != NULL);
procedure Lib2()
  returns (r:int);


// entry point
procedure Foo(z:int) {
  call FooBar();
}
```

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



}
```

Do třetice ...

```
var m:[int]int;

// library
procedure Lib1() {
  returns (r:int)
  ensures (r != NULL);
procedure Lib2()
  returns (r:int);


// entry point
procedure Foo(z:int) {
  call FooBar();
}
```

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



}
```

Do třetice ...

```
var m:[int]int;

// library
procedure Lib1() {
  returns (r:int)
  ensures (r != NULL);
procedure Lib2()
  returns (r:int)
  ensures (r != NULL);

// entry point
procedure Foo(z:int) {
  call FooBar();
}
```

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



}
```

Do třetice ...

```
var m:[int]int;

// library
procedure Lib1() {
  returns (r:int)
  ensures (r != NULL);
procedure Lib2()
  returns (r:int)
  ensures (r != NULL);

// entry point
procedure Foo(z:int) {
  call FooBar();
}
```

```
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];


}
```

Do třetice ...

```
var m:[int]int;

// library
procedure Lib1() {
  returns (r:int)
  ensures (r != NULL);
procedure Lib2()
  returns (r:int)
  ensures (r != NULL);

// entry point
procedure Foo(z:int) {
  call FooBar();
}
```

```
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;

}
```

Do třetice ...

```
var m:[int]int;

// library
procedure Lib1() {
  returns (r:int)
  ensures (r != NULL);
procedure Lib2()
  returns (r:int)
  ensures (r != NULL);

// entry point
procedure Foo(z:int) {
  call FooBar();
}
```

```
allways Lib1() != Lib2();

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;

}
```

Do třetice ...

```
var m:[int]int;

// library
procedure Lib1() {
  returns (r:int)
  ensures (r != NULL);
procedure Lib2()
  returns (r:int)
  ensures (r != NULL);

// entry point
procedure Foo(z:int) {
  call FooBar();
}
```

```
allways Lib1() != Lib2();

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;
}
```

Možné defekty

Možné defekty

- definitivní chyby

Možné defekty

- definitivní chyby
- možné chyby / nekonzistence

Možné defekty

- definitivní chyby
- možné chyby / nekonzistence
- chyby, jimž lze zabrát vhodnou vstupní podmínkou

Cíl

Cíl

- cíl: prioritizace důležitějších alarmů

Cíl

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

Cíl

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

Definice (Problém angelické verifikace)

Existuje přijatelná specifikace knihovních funkcí a přijatelná vstupní podmínka programu taková, že žádný assert nemůže v žádném běhu selhat?

Cíl

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

Definice (Problém angelické verifikace)

Existuje přijatelná specifikace knihovních funkcí a přijatelná vstupní podmínka programu taková, že žádný assert nemůže v žádném běhu selhat?

Chceme, aby přijatelná specifikace byla:

Cíl

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

Definice (Problém angelické verifikace)

Existuje přijatelná specifikace knihovních funkcí a přijatelná vstupní podmínka programu taková, že žádný assert nemůže v žádném běhu selhat?

Chceme, aby přijatelná specifikace byla:

- stručná

Cíl

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

Definice (Problém angelické verifikace)

Existuje přijatelná specifikace knihovních funkcí a přijatelná vstupní podmínka programu taková, že žádný assert nemůže v žádném běhu selhat?

Chceme, aby přijatelná specifikace byla:

- stručná
- shovívavá

Stručná vstupní podmínka

```
var m:[int]int;


procedure Foo() {
  var sum := 0;
  for var i = 1; i < 100; i++ {
    var idx = m[i];
    assert idx != NULL;
    sum := sum + m[idx];
  }
}
```

Stručná vstupní podmínka

```
var m:[int]int;
// precondition:
// m[1] != NULL && m[2] != NULL && ...
procedure Foo() {
  var sum := 0;
  for var i = 1; i < 100; i++ {
    var idx = m[i];
    assert idx != NULL;
    sum := sum + m[idx];
  }
}
```

Shovívavá vstupní podmínka

```
procedure square_root(m:int)
returns(r:int) {
  r := 3;
}
```

Shovívavá vstupní podmínka

```
//  precondition: m == 9
procedure square_root(m:int)
returns(r:int) {
  r := 3;
}
```

Jak poznat shovívavé vstupní podmínky?

Shovívavá vstupní podmínka

```
//  precondition: m != m
procedure square_root(m:int)
returns(r:int) {
  r := 3;
}
```

Jak poznat shovívavé vstupní podmínky?

- je splnitelná

Shovívavá vstupní podmínka

```
procedure square_root(m:int)
returns(r:int) {
  r := 3;
}
```

Jak poznat shovívavé vstupní podmínky?

- je splnitelná
- nevytvoří mrtvý kód

Shovívavá vstupní podmínka

```
procedure square_root(m:int)
returns(r:int) {
  r := 3;
}
```

Jak poznat shovívavé vstupní podmínky?

- je splnitelná
- nevytvoří mrtvý kód
- uživatel poradí

Příklad

```
// precondition: true
procedure Foo(b:Bool, m1,m2: Mutex) {




}
```

Příklad

```
// precondition: true
procedure Foo(b:Bool, m1,m2: Mutex) {

  if(b) {
    assert !locked(m1); lock(m1);
  } else {
    assert locked(m2); unlock(m2);
  }

}
```

Příklad

```
// precondition: !locked(m1) && locked(m2)
procedure Foo(b:Bool, m1,m2: Mutex) {

  if(b) {
    assert !locked(m1); lock(m1);
  } else {
    assert locked(m2); unlock(m2);
  }

}
```

Příklad

```
// precondition: !locked(m1) && locked(m2)
procedure Foo(b:Bool, m1,m2: Mutex) {
  assert m1 != m2
  if(b) {
    assert !locked(m1); lock(m1);
  } else {
    assert locked(m2); unlock(m2);
  }

}
```

Příklad

```
// precondition: !locked(m1) && locked(m2)
procedure Foo(b:Bool, m1,m2: Mutex) {
  assert m1 != m2
  if(b) {
    assert !locked(m1); lock(m1);
  } else {
    assert locked(m2); unlock(m2);
  }

}
```

Tento program s vloženým andělským assertem není andělsky korektní vzhledem ke vstupní podmínce.

Další andělské asserty

Jaký je význam andělských assertů v tomto kódu?

```
procedure Foo(x,y:Int) {
  if(x > y) {
    // ...
    assert false
  } else {
    // ...
    assert false
  }
}
```

Další andělské asserty

Jaký je význam andělských assertů v tomto kódu?

```
procedure Foo(x,y:Int) {
  if(x > y) {
    // ...
    assert false
  } else {
    // ...
    assert false
  }
}
```

Vstupní podmínka nesmí způsobit nedosažitelnost konce obou větví.

Další andělské asserty

Jaký je význam andělských assertů v tomto kódu?

```
// precondition: x > y
procedure Foo(x,y:Int) {
  assert false
  // ...
}
```

Další andělské asserty

Jaký je význam andělských assertů v tomto kódu?

```
// precondition: x > y
procedure Foo(x,y:Int) {
  assert false
  // ...
}
```

Vstupní podmínka musí být splnitelná.

Andělská verifikace

Andělská verifikace

Vstup: program P
s obyčejnými asserty A
a andělskými assert B.

Andělská verifikace

Vstup: program P
s obyčejnými asserty A
a andělskými assert B.
Výstup: vstupní podmínka E
a platící asserty $A' \subseteq A$.

Andělská verifikace

Vstup: program P
s obyčejnými asserty A
a andělskými assert B.
Výstup: vstupní podmínka E
a platící asserty $A' \subseteq A$.

$E \leftarrow \emptyset$
$A' \leftarrow A$
loop
  $\tau \leftarrow \mathit{Verify}(P_{A',\emptyset}, E)$
  if $\tau = \texttt{NoTrace}$ **then return** (E, A')
  end if
  $\phi \leftarrow \text{ExplainError}(P, \tau)$
  if $\text{Permissive}(P_{\emptyset,B}, E \cup \{\phi\})$ **then**
    $E \leftarrow E \cup \{\phi\}$
  else
    Let a be the failing assert in τ.
    Remove a from A'
  end if
end loop

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;
}
```

$\tau \leftarrow \mathit{Verify}(P_{A',\emptyset}, E)$:

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;
}
```

$\tau \leftarrow \mathit{Verify}(P_{A',\emptyset}, E)$:

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;
}
```

$\tau \leftarrow \mathit{Verify}(P_{A',\emptyset}, E)$:

```
z := x_1;
```

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;
}
```

$\tau \leftarrow Verify(P_{A',\emptyset}, E)$:

```
z := x_1;
```

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;
}
```

$\tau \leftarrow Verify(P_{A',\emptyset}, E)$:

```
z := x_1;
m[z] := NULL;
```

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;
}
```

$\tau \leftarrow Verify(P_{A',\emptyset}, E)$:

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

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;
}
```

$\tau \leftarrow Verify(P_{A',\emptyset}, E)$:

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

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;
}
```

$\tau \leftarrow \mathit{Verify}(P_{A',\emptyset}, E)$:

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

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;
}
```

$\tau \leftarrow Verify(P_{A',\emptyset}, E)$:

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

Příklad

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

Příklad

$\phi \leftarrow \text{ExplainError}(P, \tau)$

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

Příklad

$\phi \leftarrow \text{ExplainError}(P, \tau)$
Nejslabší vstupní podmínka pro τ:

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

Příklad

$\phi \leftarrow \text{ExplainError}(P, \tau)$

Nejslabší vstupní podmínka pro τ:

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

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

Příklad

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

$\phi \leftarrow \text{ExplainError}(P, \tau)$

Nejslabší vstupní podmínka pro τ:

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

Eliminace zápisů do pole:

Příklad

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

$\phi \leftarrow \text{ExplainError}(P, \tau)$

Nejslabší vstupní podmínka pro τ:

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

Eliminace zápisů do pole:

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

Příklad

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

$\phi \leftarrow \text{ExplainError}(P, \tau)$

Nejslabší vstupní podmínka pro τ:

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

Eliminace zápisů do pole:

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

Zjednodušení:

Příklad

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

$\phi \leftarrow \text{ExplainError}(P, \tau)$

Nejslabší vstupní podmínka pro τ:

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

Eliminace zápisů do pole:

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

Zjednodušení:

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

AngelicVerify (připomenutí)

$E \leftarrow \emptyset$
$A' \leftarrow A$
loop
  $\tau \leftarrow \mathit{Verify}(P_{A',\emptyset}, E)$
  if $\tau = \texttt{NoTrace}$ **then return** (E, A')
  end if
  $\phi \leftarrow \mathrm{ExplainError}(P, \tau)$
  if $\mathrm{Permissive}(P_{\emptyset,B}, E \cup \{\phi\})$ **then**
    $E \leftarrow E \cup \{\phi\}$
  else
    Let a be the failing assert in τ.
    Remove a from A'
  end if
end loop

ExplainError

$\tau' \leftarrow \mathit{ControlSlice}(P, \tau)$
$\phi_1 \leftarrow \mathit{wlp}(\tau')$
$\phi_2 \leftarrow \mathit{EliminateMapUpdates}(\phi_1)$
$\phi_3 \leftarrow \mathit{Simplify}(\phi_2)$

Shrnutí