Supporting Loop Proofs in KeY by using BLAST
Overview of Today's Talk

Introduction

Theoretical Background

Results

Justification

Remarks & Conclusion
Project Motivation

Fact 1) Proving loops in KeY is hard.

Fact 2) BLAST proves partial correctness automatically, sometimes.

Conclusion) KeY proofs (total correctness) can be supported by a BLAST plugin.
Toy Problem

precondition
true

program code
if (z != 5) {z = 5;} z = z - 1;

postcondition
z = 4
Theorem Proving by KeY

\[ \text{true} \]

\[
\text{if (z \neq 5) \{z = 5;\} } z = z - 1;
\]

\[ z = 4 \]
Theorem Proving by KeY

==>
{z := z_pre}
<{
  if (z != 5) z = 5;
  z = z - 1;
}> z = 4

case distinction
z_pre ≠ 5

  assignment
  z_pre := 5

  assignment
  z_pre := 5 - 1

  close by true
  4 = 4

case distinction
z_pre = 5

  assignment
  z_pre := 5 - 1

  close by true
  4 = 4
if (true)
{
    if (z != 5) {z = 5;}
    z = z - 1;
    if (!(z == 4)) {ERROR;}
}

true

if (z != 5) {z = 5;}
   z = z - 1;

z = 4
Model Checking by BLAST

Program

```c
if (z != 5)
{
  z = 5;
}
z = z - 1;
if (!(z == 4)
{
  ERROR;
}
```

Control Flow Automaton (CFA)

- **z = 5**
- **z != 5**
- **z := 5**
- **z := z - 1**
- **z = 4**
- **z != 4**
- **ERROR**
Model Checking by BLAST

Control Flow Automaton (CFA)

- $z = 5$
- $z \neq 5$
- $z := 5$
- $z := z - 1$
- $z = 4$
- $z \neq 4$
- ERROR

Abstract Reachability Tree (ART)

- true
- $z = 5$
- $z \neq 5$
- true
- $z := 5$
- true
- $z := z - 1$
- $z = 4$
- true
- $z \neq 4$
- ERROR
Abstract Reachability Tree (ART)

Path Formula

\[ z_0 \neq 5 \]
\[ z_0 \neq 5 \land z_1 = 5 \]
\[ z_0 \neq 5 \land z_1 = 5 \land z_2 = z_1 - 1 \]
\[ z_0 \neq 5 \land z_1 = 5 \land z_2 = z_1 - 1 \land z_2 \neq 4 \]
Model Checking by BLAST

Craig Interpolation Rules (Craig Interpolant $\psi$)

- $\varphi^-$ & $\varphi^+$ are unsatisfiable
- $\varphi^- \Rightarrow \psi$
- $\psi$ & $\varphi^+$ are unsatisfiable
- $\psi$ contains only symbols common to $\varphi^-$ and $\varphi^+$

Finding $\psi$ in the Path Formula

\[
\begin{align*}
\varphi^- & : z_0 ! = 5 \ & \& \ z_1 = 5 \ & \& \ z_2 = z_1 - 1 \\
\varphi^+ & : z_2 ! = 4
\end{align*}
\]

$\psi := z_2 = 4$
Model Checking by BLAST

Abstract Reachability Tree (ART)

- True
- z = 5
  - True
  - z ! = 5
    - True
    - z := 5
      - True
      - z := z - 1
        - z = 4
          - True
          - z ! = 4
            - True
            - ERROR

Refined ART (proof of partial correctness)

- True
- z = 5
  - True
  - z ! = 5
    - True
    - z := 5
      - True
      - z := z - 1
        - z = 4
          - True
          - z ! = 4
            - True
            - ERROR

Overview of Today's Talk

Introduction

Theoretical Background

Results

Justification

- Invariant Approach
- Hypothesis
- Descriptive Example

Remarks & Conclusion
Invariant Approach

- Invariant can be used for loop proofs in KeY, because of a specific taclet.
- Invariant must be somewhere in BLAST ARTs, because of the proof of partial correctness.
- Termination left as open topic to the user.
Problem Blueprint Hypothesis

- Precondition
- while (condition) { body }
- Postcondition
Descriptive Example

\[ z_1 > 0 \land z_2 \geq 0 \land z_3 > 0 \]

\[
\text{while} \ (z_1 > 0) \\
\{ \\
\quad \text{if} \ (z_2 > 0) \ \{z_1--; z_2--;\} \\
\quad \text{elseif} \ (z_3 > 0) \ \{z_3--;\} \\
\quad \text{else} \ \{z_1--;\} \\
\}
\]

\[ z_1 = 0 \land (z_3 > 0 \mid z_3 = 0) \]
Descriptive Example

precond

loopcond

z1<=0

postcond

z1=0 & (z3>0 | z3=0)

while (z1 > 0)
{
    if  (z2>0) {z1--;z2--;}
    elseif (z3>0) {z3--;}
    else          {z1--;}
}

z1>0 & z2>=0 & z3>0

z1=0 & z2>=0 & z3>0

z1<=0 & z3>0

z2=0 & z3>0

z2=0 & z3<=0

z1<=0 & z3>0

z1<=0 & z3>0

z1<=0 & z3>0
Descriptive Example

The invariant of a problem of the blueprint form is the expression

\[
\text{loopcond}_0 \mid \ldots \mid \text{loopcond}_i
\]

if we denote the annotations of the \(i\) loop-condition states by \(\text{loopcond}_i\).
Overview of Today's Talk

Introduction

Theoretical Background

- Paradigm Combination
- Invariant Justification

Results

Justification

Remarks & Conclusion
Path Formula and Symbolic Execution

**BLAST ART**

```
flowchart LR
  z=5  --> true
    z!=5  --> true
      z:=5  --> true
        z:=z-1  --> true
          z=4  --> z=4
            z!=4  --> z:=z-1
              z=4  --> true
                true  --> ERROR
```

**KeY Proof Tree**

```
==>
  {z := z_pre}
  <=
    if (z != 5) z = 5;
    z = z - 1;
  } z = 4

case distinction
  z_pre != 5
    assignment
      z_pre := 5
  case distinction
    z_pre = 5
      assignment
        z_pre := 5 - 1
        close by true
          4 = 4
      close by true
        4 = 4
```
Path Formula and Symbolic Execution

**Path Formula**

```
true

z ! = 5
```

```
true

z := 5
```

```
true

z := z - 1
```

```
z = 4
```

```
z_0 ! = 5 \& z_1 = 5 \& z_2 = z_1 - 1
```

```
 =>
z_2 = 4
```

**KeY Proof**

```
=>
{z := z_0}
<{
  if (z != 5)
  {
    z = 5;
  }
  z = z - 1;
}
```

```
z = 4
```

rule application: if_evaluation
Path Formula and Symbolic Execution

Path Formula

true

\[ z \neq 5 \]

true

\[ z = 5 \]

true

\[ z := z - 1 \]

\[ z = 4 \]

\[ z_0 \neq 5 \& z_1 = 5 \& z_2 = z_1 - 1 \]

\[ \Rightarrow \]

\[ z_2 = 4 \]

KeY Proof

\[ z_0 \neq 5 \]

\[ \Rightarrow \]

\{ \[ z := z_0 \] \}

\{ \[ z = 5; \]

\[ z = z - 1; \] \}

\[ z = 4 \]

rule application: assignment
Path Formula and Symbolic Execution

Path Formula

true

\[ z \neq 5 \]
true

\[ z = 5 \]
true

\[ z := z - 1 \]

\[ z = 4 \]

\[ z_0 \neq 5 \land z_1 = 5 \land z_2 = z_1 - 1 \]

\[ z_2 = 4 \]

KeY Proof

\[ z_0 \neq 5, z_1 = 5 \]

\[ \Rightarrow \]

\[ \{ z := z_1 \} \]

\[ <\{ z = z - 1; \} > \]

\[ z = 4 \]

rule application: assignment

auxiliary variable \( z_1 \) introduced
Path Formula and Symbolic Execution

Path Formula

true

true

true

z := 5

z := 5

z := z - 1

z = 4

z_0 \neq 5 \& z_1 = 5 \& z_2 = z_1 - 1

\Rightarrow

z_2 = 4

KeY Proof

z_0 \neq 5, z_1 = 5, z_2 = z_1 - 1

\Rightarrow

\{ z := z_2 \}

< \{ \} >

z = 4

rule application: empty_modality

auxiliary variable z_2 introduced
Path Formula and Symbolic Execution

### Path Formula

- `true`
  - `true`
    - `true`
      - `z := 5`
      - `z := z - 1`
      - `z = 4`

- `z_0 != 5 & z_1 = 5 & z_2 = z_1 - 1 => z_2 = 4`

### KeY Proof

- `z_0 != 5, z_1 = 5, z_2 = z_1 - 1 => z_2 = 4`

### Conclusion

A state annotation in an ART is true at the corresponding stage of a KeY proof.
Change of the ART Layout
Regrouping an ART into Zones

[Diagram showing a flowchart with nodes labeled as precond, loopcond, and postcond, with arrows indicating the flow between them.]

exit
Regrouping an ART into Zones

1) Invariant is initially valid.
2) Invariant remains valid.
3) Invariant is strong enough for the postcondition.
Regrouping an ART into Zones

1) Invariant is initially valid.

precondition => loopcond_0 | ... | loopcond_i
2) Invariant remains valid.

\[ \text{loopcond}_0 \mid \ldots \mid \text{loopcond}_i \]

\[ \implies \]

\[ \langle\{ \text{Loop-body} \}\rangle \ \text{loopcond}_0 \mid \ldots \mid \text{loopcond}_i \]
Regrouping an ART into Zones

3) Invariant is strong enough for the postcondition.

\[(\text{loopcond}_0 \mid \ldots \mid \text{loopcond}_i) \& \neg(\text{Loop-condition})\]

\[\Rightarrow\]

Postcondition
Overview of Today's Talk

Introduction

Theoretical Background

Results

Justification

Remarks & Conclusion

- Limitations
- Conclusion
Limitations

- Limitation to arithmetic problems
- Only simple while loops handled.
- Invariant is not optimal.
- BLAST does not support multiplication and division.
- No proof of termination.
Conclusions

- Model checking and theorem proving can be combined by invariants.
- Invariants can be found in ARTs.
- Between path formulas of model checkers and proof trees of theorem provers exists a connection.
Relation between ART State Annotations

given by craig: $p_1 \& \ldots \& p_N \& p_{N+1} \& \ldots \& p_M \Rightarrow \alpha_j$

not directly given: $\alpha_i \& p_{N+1} \& \ldots \& p_M \Rightarrow \alpha_j$