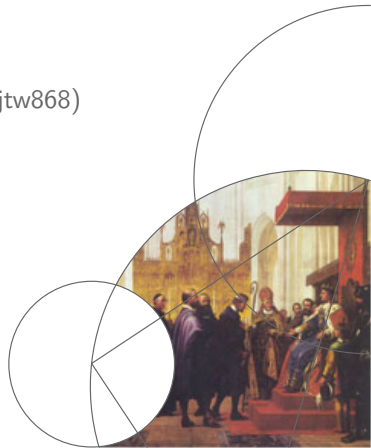# IFC: An application for dynamic evaluation and static verification of programs

Jacob Herbst (mwr148) & Matilde Broløs (jtw868)

Institute of Computer Science (DIKU)

Agenda

- Introduction
- Quantification in Interpreter
- Example programs
  - Div2
  - L
- Division and Modulo by 0
- Conclusion
- Questions

Introduction

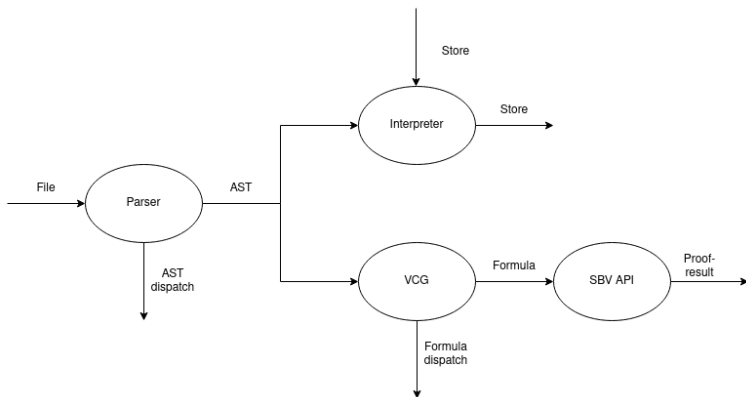- Static proving vs dynamic evaluation
- Hoare Triples

$$\{P\}s\{Q\}$$

- Verification Condition generation

$$\{WP(s, Q)\}s\{Q\}$$

Overview of Application🌮

Quantification in interpreter

- Poor choice of type
- Possibly do some of the same things as SMT solvers (however many SMT doesnt even allow quantification)
- Solution: All Quantifications evaluate to true.

Program DIV2

$$\vdash \{x = n \wedge n \geq 0\} \ \textbf{DIV2} \ \{2 \times q + x = n \wedge 0 \leq x \wedge x < 2\}$$

```
1 q := 0;
2 while x > 1 do
3   p := 1;
4   while 2 × p ≤ x do
5     q := q + p;
6     p := 2 × p;
7     x := x − p;
```

Invariant of inner loop:

$$2 \times q + x = n \wedge 0 \leq x \wedge 1 \leq p \wedge x + 2 \times (p - 1) = i \wedge 2 \leq i$$

Program DIV2

```
1  vars: [x]
2  requirements: {x ≥ 0}
3  <!=_=!>
4  👾n := x;
5  #{👾n ≥ 0};
6  q := 0;
7  while (x > 1) ?{2 × q + x = 👾n ∧ x ≥ 0 } !{x} {
8      p := 1;
9      while (2 × p ≤ x)
10     ?{2 × q + x = 👾n ∧ 0 ≤ x ∧ 1 ≤ p ∧ x + 2 × (p − 1) = i ∧ 2 ≤ i}
11     !{x} {
12         q := q + p;
13         p := 2 × p;
14         x := x − p;
15     };
16 };
17 #{2 × q + x = 👾n ∧ 0 ≤ x ∧ x < 2};
```

Program DIV2

```
1   vars: [x]
2   requirements: {x ≥ 0}
3   <!=_=!>
4   👻n := x;
5   #{👻n ≥ 0};
6   q := 0;
7   while (x > 1) ?{2 × q + x = 👻n ∧ x ≥ 0 } !{x} {
8       p := 1;
9       while (2 × p ≤ x)
10      ?{2 × q + x = 👻n ∧ 0 ≤ x ∧ 1 ≤ p ∧ x + 2 × (p − 1) = i ∧ 2 ≤ i}
11      !{x} {
12          q := q + p;
13          p := 2 × p;
14          x := x − p;
15      };
16  };
17  #{2 × q + x = 👻n ∧ 0 ≤ x ∧ x < 2};
```

Jacob Herbst (mwr148) & Matilde Brøløs (jtw868) (DIKU) — IFC: An application for dynamic evaluation and static verification of programs

Slide 8/29

Program DIV2

```
1  vars: [x]
2  requirements: {x ≥ 0}
3  <!=_=!>
4  👻n := x;
5  #{👻n ≥ 0};
6  q := 0;
7  while (x > 1) ?{2 × q + x = 👻n ∧ x ≥ 0 } !{x} {
8      i := x;
9      p := 1;
10     while (2 × p ≤ x)
11     ?{2 × q + x = 👻n ∧ 0 ≤ x ∧ 1 ≤ p ∧ x + 2 × (p − 1) = i ∧ 2 ≤ i}
12     !{x} {
13         q := q + p;
14         p := 2 × p;
15         x := x − p;
16     };
17 };
18 #{2 × q + x = 👻n ∧ 0 ≤ x ∧ x < 2};
```

Jacob Herbst (mwr148) & Matilde Brløs (jtw868) (DIKU) — IFC: An application for dynamic evaluation and static verification of programs

Slide 9/29

Program DIV2

- Loop-invariant of outer loop
- Introduction of i
- Prove total correctness

Program DIV2

- Loop-invariant of outer loop
- Introduction of i
- Prove total correctness

But what about L?

Program L

$$\vdash \{x \geq 0\} \ \mathbf{L} \ \{true\}$$

```
1  y := 0;
2  while x > 1 ∨ y > 0 do
3      if x > 0 then
4          x := x − 1;
5          y := y + y;
6      else
7          y := y − 1;
```

Program L

```
1  vars: [x]
2  requirements: {x ≥ 0}
3  <!=_=!>
4  y := 1;
5  while (x > 0 || y > 0) ?{true} !{?} {
6      if x > 0 {
7          x := x − 1;
8          y := y + y;
9      } else {
10         y := y − 1;
11     };
12 };
13 #{true};
```

Problem is that first x decrements while y increments, then only y decrements.

No way to express the variant.

Program L

With current methods we cannot prove termination, but dynamic evaluation terminates. How can we fix this?

Program L

With current methods we cannot prove termination, but dynamic evaluation terminates. How can we fix this?

Introducing tuples!

Program L

```
1  vars: [x]
2  requirements: {x ≥ 0}
3  <!=_=!>
4  y := 1;
5  while (x > 0 || y > 0) ?{true} !{x, y} {
6      if x > 0 {
7          x := x − 1;
8          y := y + y;
9      } else {
10          y := y − 1;
11      };
12  };
13  #{true};
```

What is the meaning of this?

Program L

$$\frac{\{I \wedge e \wedge v = \xi\} s \{I \wedge v \prec \xi\} \quad wf(\prec)}{\{I\} \text{ while } e \text{ invariant } I \text{ variant } v, \prec \text{ do } s \{I \wedge \neg e\}}$$

Jacob Herbst (mwr148) & Matilde BrKøs (jtw868) (DIKU) — IFC: An application for dynamic evaluation and static verification of programs

Slide 17/29

Program L

$$\frac{\{I \wedge e \wedge v = \xi\}s\{I \wedge v \prec \xi\} \quad wf(\prec)}{\{I\} \text{ while } e \text{ invariant } I \text{ variant } v, \prec \text{ do } s\{I \wedge \neg e\}}$$

$$\Downarrow$$

$$\frac{\{I \wedge e \wedge u = \xi_1 \wedge v = \xi_2\}s\{I \wedge (u \prec \xi_1 \vee v \prec \xi_2)\} \quad wf(\prec)}{\{I\} \text{ while } e \text{ invariant } I \text{ variant } u, v, \prec \text{ do } s\{I \wedge \neg e\}}$$

Program L

$$\frac{\{I \wedge e \wedge v = \xi\} s \{I \wedge v \prec \xi\} \quad wf(\prec)}{\{I\} \text{ while } e \text{ invariant } I \text{ variant } v, \prec \text{ do } s \{I \wedge \neg e\}}$$

$$\Downarrow$$

$$\frac{\{I \wedge e \wedge u = \xi_1 \wedge v = \xi_2\} s \{I \wedge (u \prec \xi_1 \vee v \prec \xi_2)\} \quad wf(\prec)}{\{I\} \text{ while } e \text{ invariant } I \text{ variant } u, v, \prec \text{ do } s \{I \wedge \neg e\}}$$

$$\Downarrow$$

$WP(\text{while } e \text{ invariant } I \text{ variant } u, v, \prec \text{ do } s, Q) =$

$I \wedge \forall x_1, ..., x_k, \xi_1, \xi_2$

$\qquad (((e \wedge I \wedge \xi_1 = u \wedge \xi_2 = v) \Rightarrow WP(s, I \wedge (u \prec \xi_1 \vee v \prec \xi_2)))$

$\qquad \wedge ((\neg e \wedge I) \Rightarrow Q))[w_i \leftarrow x_i]$

Jacob Herbst (mwr148) & Matilde Brߥ (jtw868) (DIKU) — IFC: An application for dynamic evaluation and static verification of programs

Slide 19/29

Program L

$$\frac{\{I \wedge e \wedge u = \xi_1 \wedge v = \xi_2\}s\{I \wedge (u \prec \xi_1 \vee v \prec \xi_2)\} \quad wf(\prec)}{\{I\} \text{ while } e \text{ invariant } I \text{ variant } u, v, \prec \text{ do } s\{I \wedge \neg e\}}$$

```
1  -- Updated AST ([] instead of Maybe)
2  While BExpr FOL [Variant] Stmt
```

```
1   -- Updated Parser
2   whileP :: Parser Stmt
3   whileP = do
4     rword "while"
5     c <- bExprP
6     invs <- sepBy1 (symbol "?" >> local (const True) (
          cbrackets impP)) (symbol ";")
7     let inv = foldr1 (./\.) invs
8     var <- option []
9            (symbol "!" >> cbrackets (sepBy1 aExprP (symbol ",")))
10    While c inv var <$> cbrackets seqP
```

Program L

$$\frac{\{I \wedge e \wedge u = \xi_1 \wedge v = \xi_2\}s\{I \wedge (u \prec \xi_1 \vee v \prec \xi_2)\} \quad wf(\prec)}{\{I\} \text{ while } e \text{ invariant } I \text{ variant } u,v,\prec \text{ do } s\{I \wedge \neg e\}}$$

```
1  wlp (While b inv vars s) q = do
2    (quants, vars', veq) <- foldrM go
3                  ([], ftrue, ftrue) vars
4    ...
5    where
6      ...
7      go :: Variant -> ([FOL -> FOL], FOL, FOL)
8                    -> WP ([FOL -> FOL], FOL, FOL)
9      go var (qs,rs,as) = do
10         (quant, rel, ass) <- resolveVar var
11         return (quant:qs, rel .\/. rs, ass ./\. as)
```

Jacob Herbst (mwr148) & Matilde Brøløs (jtw868) (DIKU) — IFC: An application for dynamic evaluation and static verification of programs

Slide 21/29

Program L

What about McCarthy 91?

Still not strong enough assertion language

Division and Modulo by 0

$$a \mathbin{\%} b = \begin{cases} \textit{false} & b = 0 \\ a \mathbin{\%} b & b \neq 0 \end{cases}$$

Jacob Herbst (mwr148) & Matilde Brpls (jtw868) (DIKU) — IFC: An application for dynamic evaluation and static verification of programs

Slide 23/29

Division and Modulo by 0

$$\forall y.(((b = 0) \Rightarrow \textit{false})$$
$$\wedge (b \neq 0 \Rightarrow y = a \% b \Rightarrow Q[x \leftarrow y]))$$

Division and Modulo by 0

$$\forall y.((\neg(b=0))$$
$$\wedge\,(b \neq 0 \Rightarrow y = a\,\%\,b \Rightarrow Q[x \leftarrow y]))$$

Division and Modulo by 0

$$\forall y.((\neg(b = 0))$$
$$\wedge (b \neq 0 \Rightarrow y = a \% b \Rightarrow Q[x \leftarrow y]))$$

```
1  vars: [a]
2  requirements: {}
3  <!=_=!>
4  🐛b := (−15) % (25 + a − 16);
```

Division and Modulo by 0

$$\forall y.((\neg (b = 0))$$
$$\wedge (b \neq 0 \Rightarrow y = a \% b \Rightarrow Q[x \leftarrow y]))$$

```
1 vars: [a]
2 requirements: {}
3 <!=_=!>
4 ☠b := (−15) % (25 + a − 16);
```

```
1 ∀a.
2    (25 + a − 16 ≠ 0 ∧
3    (25 + a − 16 ≠ 0 ⇒ ∀☠b. (☠b = (−15) % (25 + a − 16) ⇒ true)))
```

Cannot be verified (Counter example: $a = -9$)

Conclusion

- Dynamically evaluate programs
- Generate formulas
- Statically prove total correctness of certain programs, depending on:
    - The expressiveness of assertions (include more modulo theories)
    - The axiomatic system (additional axioms)
- Possibilities for extensions

Questions?

Jacob Herbst (mwr148) & Matilde Brוløs (jtw868) (DIKU) — IFC: An application for dynamic evaluation and static verification of programs

Slide 29/29