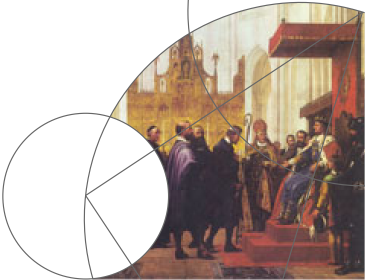
IFC: An application for dynamic evaluation and static verification of programs

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

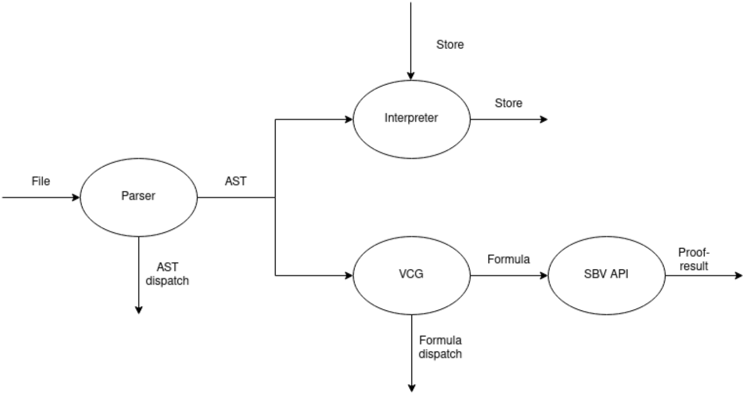
- Static proving vs dynamic evaluation
- Hoare Triples

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

- 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

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

```
q := 0;
while x > 1 do
  p := 1;
  while 2 × p ≤ x do
    q := q + p;
    p := 2 × p;
    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

```
vars: [x]
requirements: {x ≥ 0}
<!=_=!>
👻n := x;
#{👻n ≥ 0};
q := 0;
while (x > 1) ?{2 × q + x = 👻n ∧ x ≥ 0 } !{x} {
    p := 1;
    while (2 × p ≤ x)
    ?{2 × q + x = 👻n ∧ 0 ≤ x ∧ 1 ≤ p ∧ x + 2 × (p − 1) = i ∧ 2 ≤ i}
    !{x} {
        q := q + p;
        p := 2 × p;
        x := x − p;
    };
};
#{2 × q + x = 👻n ∧ 0 ≤ x ∧ x < 2};
```

Program DIV2

```
vars: [x]
requirements: {x ≥ 0}
<!=_=!>
👻n := x;
#{👻n ≥ 0};
q := 0;
while (x > 1) ?{2 × q + x = 👻n ∧ x ≥ 0 } !{x} {
    p := 1;
    while (2 × p ≤ x)
    ?{2 × q + x = 👻n ∧ 0 ≤ x ∧ 1 ≤ p ∧ x + 2 × (p − 1) = i ∧ 2 ≤ i}
    !{x} {
        q := q + p;
        p := 2 × p;
        x := x − p;
    };
};
#{2 × q + x = 👻n ∧ 0 ≤ x ∧ x < 2};
```

Program DIV2

```
vars: [x]
requirements: {x ≥ 0}
<!=_=!>
👻n := x;
#{👻n ≥ 0};
q := 0;
while (x > 1) ?{2 × q + x = 👻n ∧ x ≥ 0 } !{x} {
    i := x;
    p := 1;
    while (2 × p ≤ x)
    ?{2 × q + x = 👻n ∧ 0 ≤ x ∧ 1 ≤ p ∧ x + 2 × (p − 1) = i ∧ 2 ≤ i}
    !{x} {
        q := q + p;
        p := 2 × p;
        x := x − p;
    };
};
#{2 × q + x = 👻n ∧ 0 ≤ x ∧ x < 2};
```

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

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

```
y := 0;
while x > 1 ∨ y > 0 do
   if x > 0 then
      x := x − 1;
      y := y + y;
    else
      y := y − 1;
```

Program L

```
vars: [x]
requirements: {x ≥ 0}
<!=_=!>
y := 1;
while (x > 0 || y > 0) ?{true} !{?} {
    if x > 0 {
        x := x − 1;
        y := y + y;
    } else {
        y := y − 1;
    };
};
#{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

```
vars: [x]
requirements: {x ≥ 0}
<!=_=!>
y := 1;
while (x > 0 || y > 0) ?{true} !{x, y} {
    if x > 0 {
        x := x − 1;
        y := y + y;
    } else {
        y := y − 1;
    };
};
#{true};
```

What is the meaning of this?

Program L

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

Program L

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

$$\Downarrow$$

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

Program L

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

$$\Downarrow$$

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

$$\Downarrow$$

$$\begin{aligned}
&WP(\texttt{while}\ e\ \texttt{invariant}\ I\ \texttt{variant}\ u, v, \prec\ \texttt{do}\ s, Q) = \\
&I \land \forall x_1, \ldots, x_k, \xi_1, \xi_2 \\
&\qquad (((e \land I \land \xi_1 = u \land \xi_2 = v) \Rightarrow WP(s, I \land (u \prec \xi_1 \lor v \prec \xi_2))) \\
&\qquad\qquad \land ((\neg e \land I) \Rightarrow Q))[w_i \leftarrow x_i]
\end{aligned}$$

Program L

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

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

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

Program L

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

```
wlp (While b inv vars s) q = do
  (quants, vars', veq) <- foldrM go
            ([], ftrue, ftrue) vars
  ...
  where
    ...
    go :: Variant -> ([FOL -> FOL], FOL, FOL)
                -> WP ([FOL -> FOL], FOL, FOL)
    go var (qs,rs,as) = do
        (quant, rel, ass) <- resolveVar var
        return (quant:qs, rel .\/. rs, ass ./\. as)
```

Program L

What about McCarthy 91?

Still not strong enough assertion language

Division and Modulo by 0

$$a \,\%\, b = \begin{cases} \mathit{false} & b = 0 \\ a \,\%\, b & b \neq 0 \end{cases}$$

Division and Modulo by 0

$$\begin{aligned}\forall y.(((b = 0) \Rightarrow \mathit{false}) \\ \wedge (b \neq 0 \Rightarrow y = a \,\%\, b \Rightarrow Q[x \leftarrow y]))\end{aligned}$$

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

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

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

Division and Modulo by 0

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

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

```
∀a.
  (25 + a − 16 ≠ 0 ∧
  (25 + a − 16 ≠ 0 ⇒ ∀👻b. (👻b = (−15) % (25 + a − 16) ⇒ true)))
```

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

Conclusion

- Generate formulas
- Dynamically evaluate programs
- 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?