Completeness of Hoare Logics for While-Programs and Graph Programs

PLASMA Group Seminar

Chris Poskitt

cposkitt@cs.york.ac.uk

Department of Computer Science

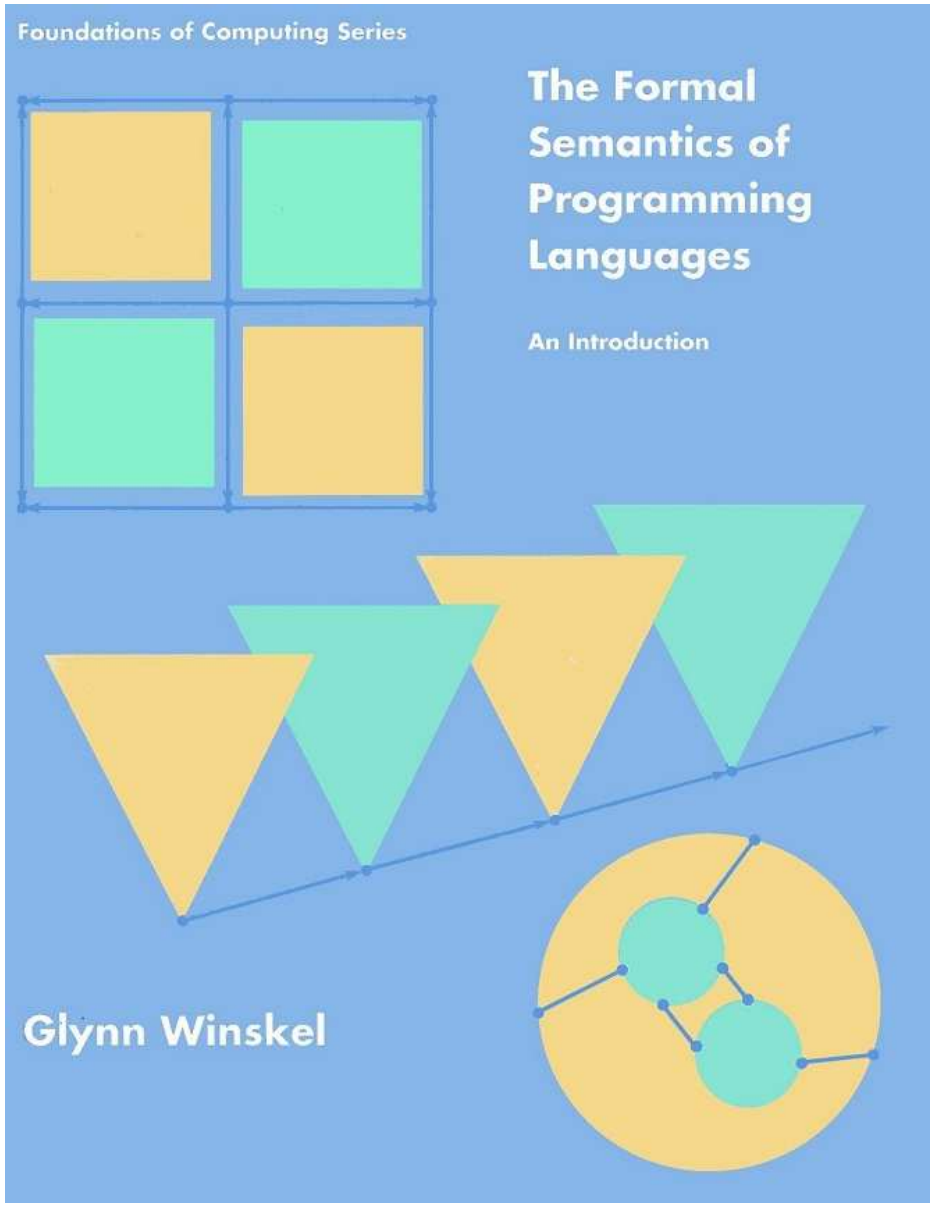
THE UNIVERSITY of York

Thursday 19th May 2011

Talk Contents

1. What is Hoare Logic?
2. Hoare Logic for While-Programs
3. Completeness of Hoare Logics
   - Relative Completeness
4. Proof of Relative Completeness
   - Expressiveness of Assertion Language
5. Completeness of Hoare Logic for Graph Programs
   - If we have time!

Acknowledgement



What is Hoare Logic?

Hoare Logic

Hoare logic is a formal proof system for reasoning about program correctness, with respect to preconditions and postconditions.

Provides axioms and inference rules for the constructs of a language.

- These capture the essential properties of the constructs, without getting too close to the underlying semantic definitions.

Hoare Triple

Central to it is the Hoare triple,

$$\{c\}\ P\ \{d\}$$

which makes the claim that:

- if program P is executed on a state satisfying precondition c, then any proper state resulting satisfies postcondition d.

Total correctness guarantees that a proper state will result.

- i.e. P terminates on c, and in the case of nondeterministic programs, does not fail.
- We consider only partial correctness in this talk (i.e. proper state not guaranteed).

Hoare Logic for While-Programs

Assertion Language **Assn**

Let **Assn** be our language of assertions (for stating preconditions and postconditions.)

We shall be informal and simply say that it contains:

- Program variables (or "locations").
- Integers.
- Arithmetical expressions (e.g. $a + b$, $c \times d$).
- Boolean expressions (e.g. $c \vee d$), arithmetic relations (e.g. $a < b$).
- Quantifiers (e.g. $\forall i.\ A$, $\exists i.\ A$).

Notation

Let $\sigma \in \Sigma$ denote a program state (a mapping from program variables to numbers), A denote an assertion in **Assn**, and I denote an interpretation. Then

$$\sigma \models^{I} A$$

denotes that σ satisfies A under interpretation I.

Notation

Let $\sigma \in \Sigma$ denote a program state (a mapping from program variables to numbers), A denote an assertion in **Assn**, and I denote an interpretation. Then

$$\sigma \models^I A$$

denotes that σ satisfies A under interpretation I.

- An interpretation I is a mapping from integer variables to integers.
- Let $\Sigma_\perp = \Sigma \cup \{\perp\}$ (where $\perp$ represents a nonterminating state).
- Let A be an assertion in **Assn** and I an interpretation. Then:

$$A^I = \{\sigma \in \Sigma_\perp \mid \sigma \models^I A\}$$

Syntax of While-Programs

While-Programs are built up from the following constructs. Again, we shall be informal (assume the usual semantics).

- Do-nothing: `skip`
- Assignment: $X := a$
- Sequential composition: $c_0; c_1$
- Conditional branching: `if` b `then` c_0 `else` c_1
- Guarded loop: `while` b `do` c

Hoare Rules for While-Programs

$$[\text{skip}]\ \frac{}{\{A\}\ \texttt{skip}\ \{A\}}$$

$$[\text{ass}]\ \frac{}{\{B[a/X]\}\ X := a\ \{B\}}$$

$$[\text{comp}]\ \frac{\{A\}\ c_0\ \{C\} \qquad \{C\}\ c_1\ \{B\}}{\{A\}\ c_0; c_1\ \{B\}}$$

Hoare Rules for While-Programs

$$[\text{if}]\ \frac{\{A \wedge b\}\ c_0\ \{B\} \qquad \{A \wedge \neg b\}\ c_1\ \{B\}}{\{A\}\ \texttt{if}\ b\ \texttt{then}\ c_0\ \texttt{else}\ c_1\ \{B\}}$$

$$[\text{while}]\ \frac{\{A \wedge b\}\ c\ \{A\}}{\{A\}\ \texttt{while}\ b\ \texttt{do}\ c\ \{A \wedge \neg b\}}$$

$$[\text{cons}]\ c \Rightarrow c'\ \frac{\{c'\}\ P\ \{d'\}}{\{c\}\ P\ \{d\}}\ d' \Rightarrow d$$

Proof Trees

A triple is provable if one can construct a proof tree with that triple as the root, with

- each leaf an instance of an axiom.
- the branches constructed by instances of inference rules.

Proof Trees

A triple is provable if one can construct a proof tree with that triple as the root, with

- each leaf an instance of an axiom.
- the branches constructed by instances of inference rules.

A simple example:

$$[\text{comp}]\ \frac{\text{SUBTREE } A \qquad [\text{ass}]\ \dfrac{}{\{x = 3 \wedge z = 5\}\ y := z\ \{x = 3 \wedge y = 5\}}}{\{y = 3 \wedge x = 5\}\ z := x; x := y; y := z\ \{x = 3 \wedge y = 5\}}$$

where SUBTREE A is:

$$\frac{\dfrac{}{\{y = 3 \wedge x = 5\}\ z := x\ \{y = 3 \wedge z = 5\}} \qquad \dfrac{}{\{y = 3 \wedge z = 5\}\ x := y\ \{x = 3 \wedge z = 5\}}}{\{y = 3 \wedge x = 5\}\ z := x;\ x := y\ \{x = 3 \wedge z = 5\}}$$

Completeness of Hoare Logics

Provable vs. Valid

If a triple $\{c\}\ P\ \{d\}$ is provable in a Hoare logic proof system, we write:

$$\vdash \{c\}\ P\ \{d\}$$

If such a triple is valid under all interpretations, regardless of the state we consider, we write:

$$\models \{c\}\ P\ \{d\}$$

Soundness

Soundness is an important property for Hoare logic systems to have.

If a system is sound, then any triple provable within it is valid.

More formally:

$$\text{if } \vdash \{c\}\ P\ \{d\} \text{ then } \models \{c\}\ P\ \{d\}$$

Completeness

Completeness would also be desirable.

If a system is complete, then any valid triple is provable within it.

More formally:

$$\text{if } \models \{c\}\ P\ \{d\} \text{ then } \vdash \{c\}\ P\ \{d\}$$

A Problem...

Consider the rule of consequence:

$$[\text{cons}]\ c \Rightarrow c' \ \frac{\{c'\}\ P\ \{d'\}}{\{c\}\ P\ \{d\}}\ d' \Rightarrow d$$

An instance of [cons] requires that we determine that certain assertions in **Assn** are valid.

Gödel's Incompleteness Theorem: logically impossible to have a complete and consistent set of axioms capable of proving all facts about natural numbers.

- Hence no complete and consistent set of axioms for establishing all valid statements in **Assn**.

Hoare Logic for While-Programs is Incomplete

Hoare logic for While-programs is incomplete due to the incompleteness of the assertion language.

A hopeless situation?! Maybe, but one does get the feeling that the axioms and rules are "adequate", or "as good as they can be".

Hoare Logic for While-Programs is Incomplete

Hoare logic for While-programs is incomplete due to the incompleteness of the assertion language.

A hopeless situation?! Maybe, but one does get the feeling that the axioms and rules are "adequate", or "as good as they can be".

Stephen Cook: separate the incompleteness of the assertion language from incompleteness due to inadequacies in the axioms and rules.

Relative Completeness

Take every valid assertion as an axiom of the system.

- Like having an oracle for the validity of assertions.

Now establish the relative completeness (or completeness in the sense of Cook) of the Hoare axioms and rules.

- i.e. relative to being able to draw from the set of valid assertions about arithmetic.

Relative completeness separates concerns about programs and reasoning about them from concerns about arithmetic and the incompleteness of any proof system for it.

Proof of Relative Completeness

Expressiveness of **Assn**

Suppose we want to show $\vdash \{A\}\ c_0; c_1\ \{B\}$. First, we must find an intermediate assertion C and show:

$$\vdash \{A\}\ c_0\ \{C\} \text{ and } \vdash \{C\}\ c_1\ \{B\}$$

How do we know that a suitable assertion C can be found? Is **Assn** expressive enough?

- A sufficient condition for expressiveness is that for every command c and postcondition B we can express their weakest precondition in **Assn**.
- Challenge: loops!

Expressiveness of **Assn**

The weakest precondition of B w.r.t. c is defined by:

$$wp^I(c, B) = \{\sigma \in \Sigma_\perp \mid [\![c]\!]\sigma \models^I B\}$$

i.e. all states from which the execution of c either diverges or ends in a final state satisfying B.

Suppose there is an assertion A_0 s.t. for all interpretations I, $A_0^I = wp^I(c, B)$. Then:

$$\models \{A\}\ c\ \{B\} \text{ iff } \models (A \Rightarrow A_0)$$

Expressiveness of **Assn**

The weakest precondition of B w.r.t. c is defined by:

$$wp^I(c, B) = \{\sigma \in \Sigma_\perp \mid [\![c]\!]\sigma \models^I B\}$$

i.e. all states from which the execution of c either diverges or ends in a final state satisfying B.

Suppose there is an assertion A_0 s.t. for all interpretations I, $A_0^I = wp^I(c, B)$. Then:

$$\models \{A\}\ c\ \{B\} \text{ iff } \models (A \Rightarrow A_0)$$

Assn is expressive iff for every c, B, there is an A_0 s.t. $A_0^I = wp^I(c, B)$ for any interpretation I.

Proof of Expressiveness

Theorem

Assn *is expressive.*

Proof: show by structural induction on commands c that for all assertions B there is an assertion $w(c, B)$ s.t.

$$wp^I(c, B) = w(c, B)^I$$

for all commands c. This amounts to showing that the following holds for all states σ:

$$\sigma \models^I w(c, B) \text{ iff } [\![c]\!]\sigma \models^I B$$

Proceed by case analysis of the language's constructs.

Proof of Expressiveness

Case $c \equiv X := a$.

In this case, let the assertion $w(X := a, B)$ be $B[a/X]$. For all states σ and interpretations I,

$$\begin{aligned} \sigma \in wp^I(X := a, B) \quad & \text{iff} \quad \sigma[[\![a]\!]\sigma/X] \models^I B \\ & \text{iff} \quad \sigma \models^I B[a/X] \\ & \text{iff} \quad \sigma \models^I w(X := a, B) \end{aligned}$$

Proof of Expressiveness

Case $c \equiv c_0; c_1$.

In this case, inductively define $w(c_0; c_1, B) \equiv w(c_0, w(c_1, B))$. For all states σ and interpretations I,

$$
\begin{aligned}
\sigma \in wp^I(c_0; c_1, B) \quad & \text{iff} \quad [\![c_0; c_1]\!]\sigma \models^I B \\
& \text{iff} \quad [\![c_1]\!]([\![c_0]\!]\sigma) \models^I B \\
& \text{iff} \quad [\![c_0]\!]\sigma \models^I w(c_1, B) \quad \text{by induction} \\
& \text{iff} \quad \sigma \models^I w(c_0, w(c_1, B)) \quad \text{by induction} \\
& \text{iff} \quad \sigma \models^I w(c_0; c_1, B)
\end{aligned}
$$

Proof of Expressiveness

Similar approaches taken for the other constructs.

Exception: while-loops! The most interesting for us as difficult to adapt for Graph Programs.

Proof of Expressiveness

Case $c \equiv \texttt{while}\ b\ \texttt{do}\ c_0$

We have that $\sigma \in wp^I(c, B)$ iff:

$$\begin{aligned}
&\forall k \forall\ \sigma_0, \ldots, \sigma_k \in \Sigma. \\
&(\sigma = \sigma_0 \wedge \\
&\forall i (0 \leq i < k).\ (\sigma_i \models^I b \wedge [\![c_0]\!]\sigma_i = \sigma_{i+1})) \\
\\
&\Rightarrow (\sigma_k \models^I b \vee B)
\end{aligned}$$

. . . but how to express this in **Assn**?

Proof of Expressiveness

Case $c \equiv \texttt{while}\ b\ \texttt{do}\ c_0$

We have that $\sigma \in wp^I(c, B)$ iff:

$$\begin{aligned}
&\forall k \forall\ \sigma_0, \ldots, \sigma_k \in \Sigma. \\
&(\sigma = \sigma_0 \wedge \\
&\forall i (0 \leq i < k).\ (\sigma_i \models^I b \wedge [\![c_0]\!]\sigma_i = \sigma_{i+1})) \\
\\
&\Rightarrow (\sigma_k \models^I b \vee B)
\end{aligned}$$

. . . but how to express this in **Assn**?

- Replace sequence of states with references to the values they contain at the program variables mentioned in c and B.

Proof of Expressiveness

Suppose an assertion A mentions only program variables from the vector $\bar{X} = X_1, \ldots, X_l$.

For a state σ, let $s_i = \sigma(X_i)$ for $1 \leq i \leq l$, and let $\bar{s} = s_1, \ldots, s_l$. Then

$$\sigma \models^I A \text{ iff } \models^I A[\bar{s}/\bar{X}]$$

for any interpretation I.

Proof of Expressiveness

Let $\bar{s}_i$ abbreviate $s_{i1}, \ldots, s_{il}$, a sequence of numbers. We rewrite the original statement.

Case $c \equiv \texttt{while}\ b\ \texttt{do}\ c_0$. We have that $\sigma \in wp^I(c, B)$ iff:

$$\begin{array}{l}
\forall k \forall\ \bar{s}_0, \ldots, \bar{s}_k. \\
(\sigma \models^I \bar{X} = \bar{s}_0 \wedge \\
\forall i (0 \leq i < k).\ (\models^I b[\bar{s}_i/\bar{X}] \wedge \models^I (w(c_0, \bar{X} = \bar{s}_{i+1}) \wedge \neg w(c_0, \mathrm{false}))[\bar{s}_i/\bar{X}]))
\end{array}$$

$$\Rightarrow \quad \models^I (b \vee B)[\bar{s}_k/\bar{X}]$$

Proof of Expressiveness

Remarkably, this statement can be expressed in **Assn** by encoding the sequences of values using Gödel's β predicate.

$$\beta(a, b, i, x) =_{def} x = a \bmod (1 + (1 + i) \times b)$$

For any sequence $a_0, \ldots, a_k$ of natural numbers, there are natural numbers n, m such that for all j, $0 \leq j \leq k$, and all x we have:

$$\beta(n, m, j, x) \text{ if and only if } x = a_j$$

i.e. x is the jth item of the sequence characterised by n and m.

NB: need to encode integers and naturals.

Proof of Expressiveness

For simplicity, assume only one program variable and hence $\bar{X} = X$.

Case $c \equiv \texttt{while}\ b\ \texttt{do}\ c_0$. We have that $\sigma \in wp^I(c, B)$ iff:

$$
\begin{aligned}
\sigma \models^I \quad & \forall k\ \forall m, n \geq 0 \\
& [\beta(n, m, 0, X) \wedge \\
& \forall i (0 \leq i < k).\ (\forall x. \beta(n, m, i, x) \Rightarrow b[x/X]) \wedge \\
& \forall x, y.\ (\beta(n, m, i, x) \wedge \beta(n, m, i+1, y) \Rightarrow \\
& \qquad (w(c_0, X = y) \wedge \neg w(c_0, \mathrm{false}))[x/X])] \\
& \Rightarrow (\beta(n, m, k, x) \Rightarrow (b \vee B)[x/X])
\end{aligned}
$$

We take the above as $w(c, B)$.

Proof of Relative Completeness

Since **Assn** is expressive, there is an assertion $w(c, B)$ for the weakest precondition of command c w.r.t. B. To prove relative completeness, we first must show that for all c, B,

$$\vdash \{w(c, B)\}\ c\ \{B\}$$

(We omit the proof, much of which overlaps with that for expressiveness of **Assn**. . . and since it is expressiveness that is causing us problems for Graph Programs!)

Then as we can apply the consequence rule, for any partial correctness triple $\{A\}\ c\ \{B\}$,

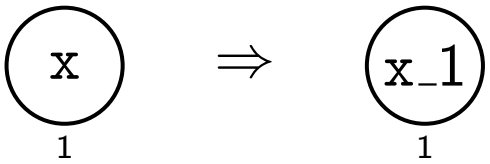
$$\models \{A\}\ c\ \{B\} \text{ implies } \vdash \{A\}\ c\ \{B\}$$

Completeness of Hoare Logic for Graph Programs

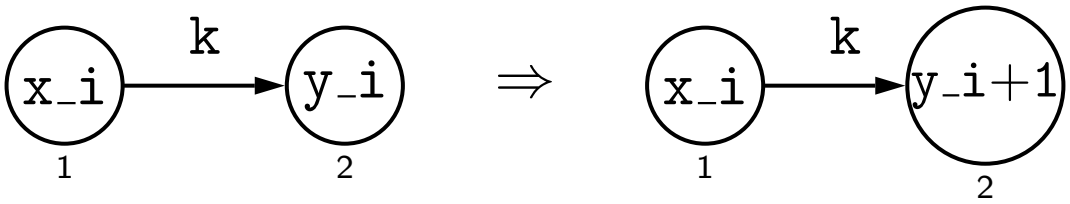
Graph Programs: Quick Reminder!

```
main = init!; inc!
```

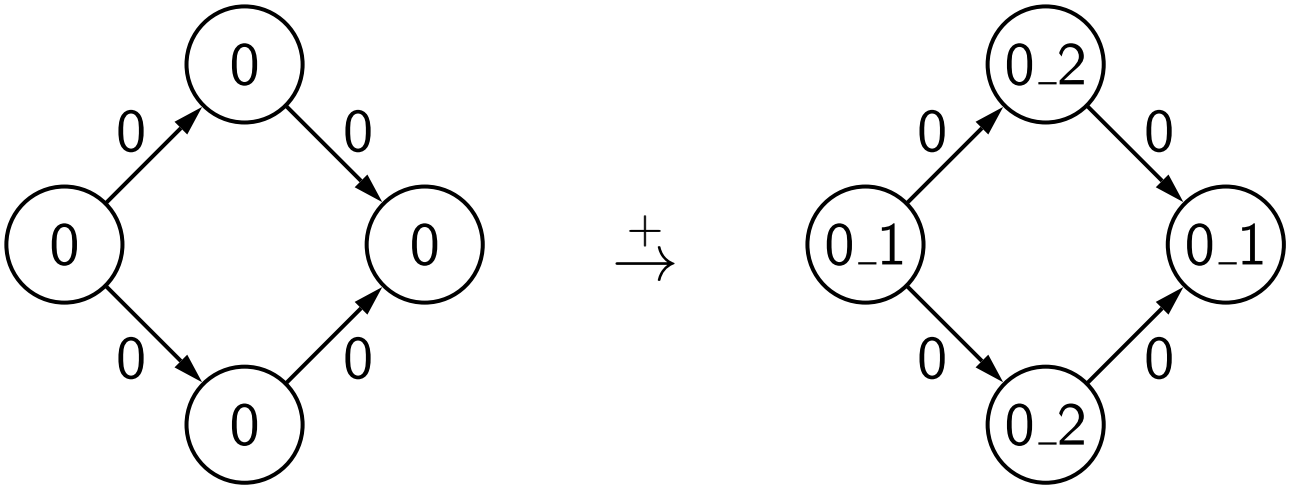
```
init(x: int)
```




```
inc(i, k, x, y: int)
```



Graph Programs: Quick Reminder!



$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{\mathrm{Pre}(\texttt{init}, e)\}\ \texttt{init}\ \{e\}}\,[\text{rule}]}{\{e\}\ \texttt{init}\ \{e\}}\,[\text{cons}]}{\{e\}\ \texttt{init!}\ \{e \wedge \neg\mathrm{App}(\{\texttt{init}\})\}}\,[!]}{\{c\}\ \texttt{init!}\ \{d\}}\,[\text{cons}] \qquad \dfrac{\dfrac{\dfrac{}{\{\mathrm{Pre}(\texttt{inc}, d)\}\ \texttt{inc}\ \{d\}}\,[\text{rule}]}{\{d\}\ \texttt{inc}\ \{d\}}\,[\text{cons}]}{\{d\}\ \texttt{inc!}\ \{d \wedge \neg\mathrm{App}(\{\texttt{inc}\})\}}\,[!]}{\{c\}\ \texttt{init!};\ \texttt{inc!}\ \{d \wedge \neg\mathrm{App}(\{\texttt{inc}\})\}}\,[\text{comp}]
$$

$$
\begin{aligned}
c &= \neg\exists(\ \textcircled{a}\ \mid \mathsf{type}(\mathtt{a}) \neq \mathsf{int}) \\
d &= \forall(\ \textcircled{a}\ , \exists(\ \textcircled{a}\ \mid \mathtt{a} = \mathtt{b_c} \wedge \mathsf{type}(\mathtt{b}, \mathtt{c}) = \mathsf{int})) \\
e &= \forall(\ \textcircled{a}^{1}, \exists(\ \textcircled{a}^{1} \mid \mathsf{type}(\mathtt{a}) = \mathsf{int}) \vee \exists(\ \textcircled{a}_{1} \mid \mathtt{a} = \mathtt{b_c} \wedge \mathsf{type}(\mathtt{b}, \mathtt{c}) = \mathsf{int})) \\
\neg\mathrm{App}(\{\texttt{init}\}) &= \neg\exists(\ \textcircled{x}_{1} \mid \mathsf{type}(\mathtt{x}) = \mathsf{int}) \\
\neg\mathrm{App}(\{\texttt{inc}\}) &= \neg\exists(\ \textcircled{x_i} \xrightarrow{\mathtt{k}} \textcircled{y_i} \mid \mathsf{type}(\mathtt{i}, \mathtt{k}, \mathtt{x}, \mathtt{y}) = \mathsf{int}) \\
\mathrm{Pre}(\texttt{init}, e) &= \forall(\ \textcircled{x}_{1}\ \textcircled{a}_{2} \mid \mathrm{type}(\mathtt{x}) = \mathrm{int}, \exists(\ \textcircled{x}_{1}\ \textcircled{a}_{2} \mid \mathsf{type}(\mathtt{a}) = \mathsf{int}) \\
&\quad \vee \exists(\ \textcircled{x}_{1}\ \textcircled{a}_{2} \mid \mathtt{a} = \mathtt{b_c} \wedge \mathsf{type}(\mathtt{b}, \mathtt{c}) = \mathsf{int})) \\
\mathrm{Pre}(\texttt{inc}, d) &= \forall(\ \textcircled{x_i}_{1} \xrightarrow{\mathtt{k}} \textcircled{y_i}_{2}\ \textcircled{a}_{3} \mid \mathsf{type}(\mathtt{i}, \mathtt{k}, \mathtt{x}, \mathtt{y}) = \mathsf{int}, \\
&\quad \exists(\ \textcircled{x_i}_{1} \xrightarrow{\mathtt{k}} \textcircled{y_i}_{2}\ \textcircled{a}_{3} \mid \mathtt{a} = \mathtt{b_c} \wedge \mathsf{type}(\mathtt{b}, \mathtt{c}) = \mathsf{int}))
\end{aligned}
$$

Completeness of Graph Programs

The problem we are facing is to prove the expressiveness of our assertion language.

- E-conditions demands the existence of graph morphisms between (instantiated) graphs of the assertion and the program state (i.e. graph).
- Though there should be a translation into something close to Courcelle's first-order logic over graphs.

Biggest challenge: expressing the weakest precondition of our analogue to the while-loop:

$$w(\mathcal{R}!, c)$$

A Starting Point!

Can we express the following as an E-condition?

We claim that $G \models \mathrm{wp}(\mathcal{R}!, c)$ iff:

$$\begin{aligned}
&\forall k \in \mathbb{N}.\ \forall G_0, \ldots, G_k \in \mathcal{G}(\mathcal{L}) \\
&(G = G_0 \wedge \\
&\forall i (0 \leq i < k).\ G \models \mathrm{App}(\mathcal{R}) \wedge G_i \Rightarrow_{\mathcal{R}} G_{i+1})) \\
\\
&\Rightarrow G_k \models \mathrm{App}(\mathcal{R}) \vee c)
\end{aligned}$$

Ideas and Problems

- We conjecture the existence of a transformation from first-order arithmetic to E-conditions. e.g. $\forall x.\ \exists y.\ x + y = 0$ could be expressed as:
- $\forall(\emptyset \mid \texttt{type(x)} = \texttt{int}, \exists(\emptyset \mid \texttt{x} + \texttt{y} = \texttt{0 and type(y)} = \texttt{int}))$

- Might be helpful, but how do we then encode information about arbitrary graphs in E-conditions (the graphs in E-conditions are fixed)?
- Is there a Gödel numbering for graphs? Can we encode the original statement entirely in first-order arithmetic?

Summary

Summary

- Discussed the notion of completeness for Hoare logic systems.
- Showed that Hoare logic for While-Programs is incomplete...
- ...but relatively complete, if we have an oracle for the validity of assertions.
- Discussed the proof of relative completeness, with a focus on showing that an assertion language is expressive.
- Encoded a statement about a sequence of states using arithmetic (i.e. Gödel's β predicate).
- Considered Hoare logic for Graph Programs, and highlighted the difficulty of proving that the assertion language is expressive.

That's all folks!

Questions? Ideas? :-)

cposkitt@cs.york.ac.uk

http://www.cs.york.ac.uk/~cposkitt/