Relational logic and its applications to verification.

How to verify several programs at once.

Mihir Mehta

Department of Computer Science
University of Texas at Austin
mihir@cs.utexas.edu

01 April 2015

Relational logic

- Relational logic started out as a means to prove program equivalence.
- Hoare quadruples - intended to express equivalence of two programs in a certain context.
- Benton's initial work - limited to structurally identical programs.
- Later, Zaks and Pnueli, cross products - again limited to structurally equivalent programs.
- Separately, self-composition - sound and complete but gives really hard verification conditions.
- This paper: serves to unite both into one that's actually machine-checkable.

- The program syntax is:

$$c \quad ::= \quad x := e \mid a[e] := e \mid \mathsf{skip} \mid \mathsf{assert}(b) \mid c; c \mid \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mid \mathsf{while}\ b\ \mathsf{do}\ c$$

- The program semantics are:

$$\frac{}{\langle \mathsf{assert}(b), \sigma\rangle \rightsquigarrow \langle \mathsf{skip}, \sigma\rangle}\ [\![b]\!]\sigma$$

$$\frac{\langle c_1, \sigma\rangle \rightsquigarrow \langle c_1', \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \rightsquigarrow \langle c_1'; c_2, \sigma'\rangle} \qquad \frac{\langle c_1, \sigma\rangle \rightsquigarrow \langle \mathsf{skip}, \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \rightsquigarrow \langle c_2, \sigma'\rangle}$$

$$\frac{}{\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle \rightsquigarrow \langle c; \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle}\ [\![b]\!]\sigma \qquad \frac{}{\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle \rightsquigarrow \langle \mathsf{skip}, \sigma\rangle}\ [\![\neg b]\!]\sigma$$

Fig. 2. Program semantics (excerpt)

$$\frac{}{c_1 \times c_2 \to c_1; c_2} \qquad \frac{c_1 \times c_2 \to c \qquad c_1' \times c_2' \to c'}{(c_1; c_1') \times (c_2; c_2') \to c; c'}$$

$$\frac{c_1 \times c_2 \to c}{(\mathsf{while}\ b_1\ \mathsf{do}\ c_1) \times (\mathsf{while}\ b_2\ \mathsf{do}\ c_2) \to \mathsf{assert}(b_1 \Leftrightarrow b_2); \mathsf{while}\ b_1\ \mathsf{do}\ (c; \mathsf{assert}(b_1 \Leftrightarrow b_2))}$$

$$\frac{c_1 \times c_2 \to c \qquad c_1' \times c_2' \to c'}{(\mathsf{if}\ b_1\ \mathsf{then}\ c_1\ \mathsf{else}\ c_1') \times (\mathsf{if}\ b_2\ \mathsf{then}\ c_2\ \mathsf{else}\ c_2') \to \mathsf{assert}(b_1 \Leftrightarrow b_2); \mathsf{if}\ b_1\ \mathsf{then}\ c\ \mathsf{else}\ c'}$$

$$\frac{c_1 \times c \to c_1' \qquad c_2 \times c \to c_2'}{(\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2) \times c \to \mathsf{if}\ b\ \mathsf{then}\ c_1'\ \mathsf{else}\ c_2'}$$

Fig. 3. Product construction rules

$$\frac{}{\vdash \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \succcurlyeq \mathsf{assert}(b); c_1} \qquad \frac{}{\vdash \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \succcurlyeq \mathsf{assert}(\neg b); c_2}$$

$$\frac{}{\vdash \mathsf{while}\ b\ \mathsf{do}\ c \succcurlyeq \mathsf{assert}(b); c; \mathsf{while}\ b\ \mathsf{do}\ c}$$

$$\frac{}{\vdash \mathsf{while}\ b\ \mathsf{do}\ c \succcurlyeq \mathsf{while}\ b \wedge b'\ \mathsf{do}\ c; \mathsf{while}\ b\ \mathsf{do}\ c} \qquad \frac{}{\vdash \mathsf{while}\ b\ \mathsf{do}\ c \succcurlyeq \mathsf{assert}(b); c; \mathsf{assert}(\neg b)}$$

$$\frac{\vdash c \succcurlyeq c'}{\vdash \mathsf{while}\ b\ \mathsf{do}\ c \succcurlyeq \mathsf{while}\ b\ \mathsf{do}\ c'} \qquad \frac{\vdash c_1 \succcurlyeq c_1' \qquad \vdash c_2 \succcurlyeq c_2'}{\vdash \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \succcurlyeq \mathsf{if}\ b\ \mathsf{then}\ c_1'\ \mathsf{else}\ c_2'}$$

$$\frac{\vdash c \succcurlyeq c' \qquad \vdash c' \succcurlyeq c''}{\vdash c \succcurlyeq c''} \qquad \frac{}{\vdash c \succcurlyeq c} \qquad \frac{\vdash c_1 \succcurlyeq c_1' \qquad \vdash c_2 \succcurlyeq c_2'}{\vdash c_1; c_2 \succcurlyeq c_1'; c_2'}$$

Fig. 4. Syntactic reduction rules

Our work I

- We'd like to work with proving properties between the input and output of several calls to the same function. $\forall x1, ..., xn.(Pre(x1, ..., xn) \Rightarrow Post(f(x1), ...f(xn)))$
- By making product programs, we should be able to prove properties such as symmetry, reflexivity, transitivity, determinism by expressing them as Hoare pre- and post-conditions and then passing them to a theorem prover (Z3).
- Evidently, we can't assume structural equivalence in general. For instance, conditionals may go down different branches, and loops may have different numbers of iterations.
- However, we should be able to benefit from the somewhat common structure.

Our work II

- At present, we're trying to implement a verifier of this kind, and also figure out judgement rules that might be useful for the kinds of things we're trying to prove.
- We're focussing on Java programs with comparators - those are the main motivating example.
- We're able to deal with loop-free programs, but will re-write to deal better with looping and recursing programs, incorporating the product construction.