Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Lars Hupel

WS 2017/18
30.01.2018

Semantics of Programming Languages

Exercise Sheet 14

Exercise 14.1 Inverse Analysis

Consider a simple sign analysis based on this abstract domain:

datatype *sign* = *None* | *Neg* | *Pos0* | *Any*

fun γ :: *"sign ⇒ val set"* **where**
"γ None = {}" |
"γ Neg = {i. i < 0}" |
"γ Pos0 = {i. i ≥ 0}" |
"γ Any = UNIV"

Define inverse analyses for "+" and "<" and prove the required correctness properties:

fun *inv_plus'* :: *"sign ⇒ sign ⇒ sign ⇒ sign * sign"*
lemma
"⟦ inv_plus' a a1 a2 = (a1',a2'); i1 ∈ γ a1; i2 ∈ γ a2; i1+i2 ∈ γ a ⟧
⟹ i1 ∈ γ a1' ∧ i2 ∈ γ a2' "
fun *inv_less'* :: *"bool ⇒ sign ⇒ sign ⇒ sign * sign"*
lemma
"⟦ inv_less' bv a1 a2 = (a1',a2'); i1 ∈ γ a1; i2 ∈ γ a2; (i1<i2) = bv ⟧
⟹ i1 ∈ γ a1' ∧ i2 ∈ γ a2'"

Exercise 14.2 Command Equivalence

Recall the notion of *command equivalence*:

$c_1 \sim c_2 \equiv (\forall s\ t.\ (c_1,\ s) \Rightarrow t \ \longleftrightarrow\ (c_2,\ s) \Rightarrow t)$

1. Define a function *is_SKIP* :: *com* ⇒ *bool* which holds on commands equivalent to *SKIP*. The function *is_SKIP* should be as precise as possible, but it should not analyse arithmetic or boolean expressions.

   Prove: *is_SKIP c* ⟹ $c \sim$ *SKIP*

2. The following command equivalence is wrong. Give a counterexample in the form of concrete instances for b_1, b_2, c_1, c_2, and a state s.

$$\begin{array}{l} \mathit{WHILE}\ b_1\ \mathit{DO}\ \mathit{IF}\ b_2\ \mathit{THEN}\ c_1\ \mathit{ELSE}\ c_2 \\ \sim\ \mathit{IF}\ b_2\ \mathit{THEN}\ (\mathit{WHILE}\ b_1\ \mathit{DO}\ c_1)\ \mathit{ELSE}\ (\mathit{WHILE}\ b_1\ \mathit{DO}\ c_2) \end{array} \quad (*)$$

3. Define a condition P on b_1, b_2, c_1, and c_2 such that the previous statement $(*)$ holds, i.e. $P\ b_1\ b_2\ c_1\ c_2 \Longrightarrow (*)$

   Your condition should be as precise as possible, but only using:

   - *lvars* :: *com* ⇒ *vname set* (all left variables, i.e. written variables),
   - *rvars* :: *com* ⇒ *vname set* (all right variables, i.e. all read variables),
   - *vars* :: *bexp* ⇒ *vname set* (all variables in a condition), and
   - boolean connectives and set operations

General homework instructions

The first homework is pen & paper (or keyboard & text file). You have the choice of uploading a text file or a PDF scan of hand-written notes to the submission system. Physical paper submissions are not accepted.

Homework 14.1 Palindromes

Submission until Tuesday, February 6, 2018, 10:00am.

A *palindrome* is a word which reads the same in forward and backward direction. We introduce an inductive predicate *palindrome* of type *'a list* ⇒ *bool*:

inductive *palindrome* **where**
  "palindrome []"
| *"palindrome [x]"*
| *"palindrome xs* $\Longrightarrow$ *palindrome* ([*x*] @ *xs* @ [*x*])*"*

1. Show *palindrome xs* $\Longrightarrow$ *rev xs* = *xs*.
2. Show *rev xs* = *xs* $\Longrightarrow$ *palindrome xs*.

You are allowed to use rule induction, structural induction, and the following induction rule:

$$\frac{P\ []\qquad \forall x.\ P\ [x]\qquad \forall x\ y\ xs.\ P\ xs \longrightarrow P\ ([x]\ @\ xs\ @\ [y])}{\forall xs.\ P\ xs}\ \text{IND}$$

Homework 14.2 Assertions

Submission until Tuesday, February 6, 2018, 10:00am.

We extend IMP with an assertion command *ASSERT bexp*. Intuitively, the execution gets stuck if the asserted expression evaluates to false, otherwise *ASSERT bexp* behaves like *SKIP*. Add the appropriate rule to the big-step semantics. Also add a rule to the Hoare calculus and adapt the proofs of correctness and completeness.

```
datatype
  com = SKIP
      | Assign vname aexp       ("_ ::= _" [1000, 61] 61)
      | Seq    com  com         ("_;;/ _" [60, 61] 60)
      | If     bexp com com     ("(IF _/ THEN _/ ELSE _)" [0, 0, 61] 61)
      | While  bexp com         ("(WHILE _/ DO _)" [0, 61] 61)
      | ASSERT bexp
```