Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Andrei Popescu, Peter Lammich

WS 2012/13
29. 01. 2013

Semantics of Programming Languages

Exercise Sheet 14

The following exercises are typical exam exercises. You are supposed to solve them on a sheet of paper, without using Isabelle/HOL.

Exercise 14.1 Inductive Predicates

Consider the following inductive predicate, which characterizes odd natural numbers.

```
inductive odd :: "nat ⇒ bool" where
  Suc_0: "odd (Suc 0)" |
  Suc_Suc: "odd n ⟹ odd (Suc (Suc n))"
```

Using the induction principle for the predicate *odd*, it can be proven that three times any odd number is also odd:

```
lemma "odd n ⟹ odd (n + n + n)"
proof (induct rule: odd.induct)
  case Suc_0 show ?case by (simp add: odd.intros)
next
  case (Suc_Suc n) thus ?case by (simp add: odd.intros)
qed
```

First, write down precisely what subgoals remain after performing induction. How many cases are there? Which assumptions are available, and what conclusion must be proved in each case? Next, describe how each case can be proved. Which simplification rules or introduction rules are used to prove each case?

Exercise 14.2 Collecting Semantics

Recall the datatype of annotated commands (type *'a acom*) and the collecting semantics (function *step* :: *state set* ⇒ *state set acom* ⇒ *state set acom*) from the lecture. We reproduce the definition of *step* here for easy reference. (Recall that *post c* simply returns the right-most annotation from command *c*.)

```
step S (SKIP {_}) = SKIP {S}
step S (x::=e {_}) = x::=e {{s'. ∃s∈S. s'=s(x:=aval e s)}}
step S (c_1; c_2) = step S c_1; step (post c_1) c_2
step S (IF b THEN {P_1} c_1 ELSE {P_2} c_2 {_}) =
    IF b THEN {{s∈S. bval b s}} step P_1 c_1
    ELSE {{s∈S. ¬ bval b s}} step P_2 c_2
    {post c_1 ∪ post c_2}
step S ({I} WHILE b DO {P} c {_}) =
    {S ∪ post c}
    WHILE b DO {{s:I. bval b s}} step P C
    {{s∈I. ¬ bval b s}}
```

In this exercise you must evaluate the collecting semantics on the example program below by repeatedly applying the *step* function.

```
c = (IF x < 0
        THEN {A_1}
            {A_2} WHILE 0 < y DO {A_3} (y := y + x {A_4}) {A_5}
        ELSE {A_6} SKIP {A_7}
    ) {A_8}
```

Let S be $\{\langle -2,3\rangle,\langle 1,2\rangle\}$, abbreviated $-2,3 \mid 1,2$. Calculate column $n{+}1$ in the table below by evaluating *step S c* with the annotations for c taken from column n. For conciseness, we use "$\langle i,\, j\rangle$" as notation for the state $<''x'':=i,\ ''y'':=j>$. We have filled in columns 0 and 1 to get you started; now compute and fill in the rest of the table.

	0	1	2	3	4	5	6	7	8	9	10
A_1	$\emptyset$	$-2,3$	$-2,3$	$-2,3$	$-2,3$	$-2,3$	$-2,3$	$-2,3$	$-2,3$	$-2,3$	$-2,3$
A_2	$\emptyset$	$\emptyset$	$-2,3$	$-2,3$	$-2,3$	$-2,3 \mid -2,1$	$-2,3 \mid -2,1$	$-2,3 \mid -2,1$	$-2,3 \mid -2,1 \mid -2,-1$	$-2,3 \mid -2,1 \mid -2,-1$	$-2,3 \mid -2,1 \mid -2,-1$
A_3	$\emptyset$	$\emptyset$	$\emptyset$	$-2,3$	$-2,3$	$-2,3$	$-2,3 \mid -2,1$	$-2,3 \mid -2,1$	$-2,3 \mid -2,1$	$-2,3 \mid -2,1$	$-2,3 \mid -2,1$
A_4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$-2,1$	$-2,1$	$-2,1$	$-2,1 \mid -2,-1$	$-2,1 \mid -2,-1$	$-2,1 \mid -2,-1$	$-2,1 \mid -2,-1$
A_5	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$-2,-1$	$-2,-1$
A_6	$\emptyset$	$1,2$	$1,2$	$1,2$	$1,2$	$1,2$	$1,2$	$1,2$	$1,2$	$1,2$	$1,2$
A_7	$\emptyset$	$\emptyset$	$1,2$	$1,2$	$1,2$	$1,2$	$1,2$	$1,2$	$1,2$	$1,2$	$1,2$
A_8	$\emptyset$	$\emptyset$	$\emptyset$	$1,2$	$1,2$	$1,2$	$1,2$	$1,2$	$1,2$	$1,2$	$1,2 \mid -2,-1$

Exercise 14.3 Substitution

type_synonym *vname = "char list"*

datatype *aexp = N int | V vname | Plus aexp aexp*

fun *subst*:: *"aexp ⇒ vname ⇒ aexp ⇒ aexp"* **where**
  "subst (N i) v a' = (N i)"
| *"subst (V v') v a' = (if v=v' then a' else (V v'))"*
| *"subst (Plus a1 a2) v a' = Plus (subst a1 v a') (subst a2 v a')"*

fun *occurs*:: *"aexp ⇒ vname ⇒ bool"* **where**
  "occurs (N _) _ ⟷ False"
| *"occurs (V v') v ⟷ v=v'"*
| *"occurs (Plus a1 a2) v ⟷ occurs a1 v ∨ occurs a2 v"*

lemma *lem1*: "$\neg$*occurs a v* $\Longrightarrow$ *subst a v a'* = *a*"
by (*induct a*) *auto*

lemma *lem2*: "*occurs* (*subst* (*V v*) *v* (*V v*)) *v*" **by** *simp*