Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Simon Wimmer

WS 2016/17
06. 12. 2016

Semantics of Programming Languages

Exercise Sheet 7

Exercise 7.1 Deskip

Define a recursive function

fun *deskip* :: "$com \Rightarrow com$"

that eliminates as many *SKIP*s as possible from a command. For example:

$deskip\ (SKIP;;\ WHILE\ b\ DO\ (x ::= a;;\ SKIP)) = WHILE\ b\ DO\ x ::= a$

Prove its correctness by induction on c:

lemma "$deskip\ c \sim c$"

Exercise 7.2 Small step pre-order

We define a pre-order $\preceq$ on programs that uses the small-step semantics. The relation $p \preceq p'$ shall hold if p' computes for any input the same output as p, and in at most the same number of steps.

The following relation is the n-steps reduction relation:

inductive
  nsteps :: "$com * state \Rightarrow nat \Rightarrow com * state \Rightarrow bool$"
  ("_ $\to$^_ _" [60,1000,60]999)
where
  zero_steps: "$cs \to$^$0\ cs$" |
  one_step: "$cs \to cs' \Longrightarrow cs' \to$^$n\ cs'' \Longrightarrow cs \to$^$(Suc\ n)\ cs''$"

Prove the following lemmas:

lemma *small_steps_n*: "$cs \to* cs' \Longrightarrow (\exists n.\ cs \to$^$n\ cs')$"
lemma *n_small_steps*: "$cs \to$^$n\ cs' \Longrightarrow cs \to* cs'$"
lemma *nsteps_trans*: "$cs \to$^$n1\ cs' \Longrightarrow cs' \to$^$n2\ cs'' \Longrightarrow cs \to$^$(n1+n2)\ cs''$"

The pre-order relation is defined as follows:

definition
  small_step_pre :: "$com \Rightarrow com \Rightarrow bool$" (**infix** "$\preceq$" 50) **where**

$``c \preceq c' \equiv (\forall\, s\ t\ n.\ (c,s) \rightarrow\hat{}n\ (SKIP,\ t)\ \longrightarrow (\exists\ n' \geq n.\ (c',\ s) \rightarrow\hat{}n'\ (SKIP,\ t)))"$

Prove the following lemma:

lemma *small_eqv_implies_big_eqv*:
  assumes $``c \preceq c'"$ $``c' \preceq c"$
  shows $``c \sim c'"$

Exercise 7.3 Compiler optimization

A common programming idiom is *IF b THEN c*, i.e., the else-branch consists of a single *SKIP* command.

1. Look at how the program *IF Less (V ''x'') (N 5) THEN ''y'' ::= N 3 ELSE SKIP* is compiled by *ccomp* and identify a possible compiler optimization.
2. Implement an optimized compiler (by modifying *ccomp*) which reduces the number of instructions for programs of the form *IF b THEN c*.
3. Extend the proof of *comp_bigstep* to your modified compiler.

Homework 7.1 Compiling *REPEAT*

Submission until Tuesday, December 2, 10:00am.

We extend *com* with a *REPEAT c UNTIL b* statement, adding the following rules to our big-step semantics:
RepeatTrue: $[\![(c,\ s_1) \Rightarrow s_2;\ bval\ b\ s_2]\!] \Longrightarrow (REPEAT\ c\ UNTIL\ b,\ s_1) \Rightarrow s_2$
RepeatFalse: $[\![(c,\ s_1) \Rightarrow s_2;\ \neg\ bval\ b\ s_2;\ (REPEAT\ c\ UNTIL\ b,\ s_2) \Rightarrow s_3]\!] \Longrightarrow (REPEAT\ c\ UNTIL\ b,\ s_1) \Rightarrow s_3$

Building on this, extend the compiler *ccomp* and its correctness theorem *ccomp_bigstep* to *REPEAT* loops. **Hint:** the recursion pattern of the big-step semantics and the compiler for *REPEAT* should match.

Download the files *Repeat_Big_Step.thy* and *Repeat_Compiler_Template.thy*. Finish the definition of *ccomp* and the proof of *ccomp_bigstep* in *Repeat_Compiler_Template.thy*, and submit this theory using as filename the schema *FirstnameLastname2.thy*.

Homework 7.2 Commuting sequences of commands

Submission until Tuesday, December 13, 10:00am.

Write a function that collects all variables that occur in a command. (Hint: You need to write such functions also for boolean and arithmetic expressions)

fun *vars* :: *"com ⇒ vname set"* **where**

Then show the following two lemmas:

lemma *aval_equiv*:
  "$(c, s) \Rightarrow t \Longrightarrow \mathit{varsa}\ a \cap \mathit{vars}\ c = \{\} \Longrightarrow \mathit{aval}\ a\ t = \mathit{aval}\ a\ s$"
lemma *bval_equiv*:
  "$(c, s) \Rightarrow t \Longrightarrow \mathit{varsb}\ b \cap \mathit{vars}\ c = \{\} \Longrightarrow \mathit{bval}\ b\ t = \mathit{bval}\ b\ s$"

Finally prove that a sequence of commands can be commuted if the commands do not share any common variables:

lemma *Seq_commute*:
  assumes "$\mathit{vars}\ c1 \cap \mathit{vars}\ c2 = \{\}$"
    shows "$c1;;c2 \sim c2;;c1$"
oops

One possible way to get there, is to prove the following auxiliary lemma first:

lemma *Seq_commute'*:
  assumes "$(c1, s) \Rightarrow s'$" "$(c2, s') \Rightarrow t$" "$\mathit{vars}\ c1 \cap \mathit{vars}\ c2 = \{\}$"
    shows "$(c2;;c1, s) \Rightarrow t$"

You only need to do the cases for while-loops and assignment. The latter may necessitate another helper lemma.

lemma *Seq_commute*:
  assumes "$\mathit{vars}\ c1 \cap \mathit{vars}\ c2 = \{\}$"
    shows "$c1;;c2 \sim c2;;c1$"

Homework 7.3 Algebra of Commands

Submission until Tuesday, December 13, 10:00am.

We define an extension of the language with parallel composition ($\|$).for which we consider the small-step equivalence
Your task will be to prove various algebraic laws for the small-step equivalence. The most helpful methods will be number induction and/or pair-based rule induction over the *nsteps* relation, using *nsteps_induct* (provided below).

datatype
  com =
— sequential part as before —
    | *Par com com*   (**infix** "$\|$" *59*)

inductive
  small_step :: "$com * state \Rightarrow com * state \Rightarrow bool$" (**infix** "$\rightarrow$" *55*)
where
— sequential part as before —
ParL: "$(c1,s) \rightarrow (c1',s') \Longrightarrow (c1 \parallel c2,s) \rightarrow (c1' \parallel c2,s')$" |
ParLSkip: "$(SKIP \parallel c,s) \rightarrow (c,s)$" |
ParR: "$(c2,s) \rightarrow (c2',s') \Longrightarrow (c1 \parallel c2,s) \rightarrow (c1 \parallel c2',s')$" |

ParRSkip: "$(c \parallel SKIP,s) \rightarrow (c,s)$"

lemmas *small_step_induct* = *small_step.induct*[*split_format*(*complete*)]
inductive
nsteps :: "*com* ∗ *state* ⇒ *nat* ⇒ *com* ∗ *state* ⇒ *bool*"
("_ →^_ _" [60,1000,60]999)
where
zero_steps[*simp*,*intro*]: "$cs \rightarrow\hat{}0\ cs$" |
one_step[*intro*]: "$cs \rightarrow cs' \Longrightarrow cs' \rightarrow\hat{}n\ cs'' \Longrightarrow cs \rightarrow\hat{}(Suc\ n)\ cs''$"

lemmas *nsteps_induct* = *nsteps.induct*[*split_format*(*complete*)]

definition
small_step_pre :: "*com* ⇒ *com* ⇒ *bool*" (**infix** "⪯" *50*) **where**
"$c \preceq c' \equiv (\forall s\ t\ n.\ (c,s) \rightarrow\hat{}n\ (SKIP,\ t) \ \longrightarrow (\exists\ n' \geq n.\ (c',\ s) \rightarrow\hat{}n'\ (SKIP,\ t)))$"

Based on the pre-order on programs, define an equivalence relation ≈ on programs.

Now prove commutativity and associativity of ∥. You are free to do either automatic or Isar proofs.

lemma *Par_commute*: "$c \parallel d \approx d \parallel c$"

lemma *Par_assoc*: "$(c \parallel d) \parallel e \approx c \parallel (d \parallel e)$"