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

WS 2017/18
28.11.2017

Semantics of Programming Languages

Exercise Sheet 7

Exercise 7.1 Deskip

Define a recursive function

fun *deskip* :: *"com ⇒ 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
assumes "(*WHILE b DO c*, *s*) ⇒ *t*" **and** "∀ *s t*. (*c*, *s*) ⇒ *t* ⟶ (*c'*, *s*) ⇒ *t*"
shows "(*WHILE b DO c'*, *s*) ⇒ *t*"
using *assms*
by (*induction "WHILE b DO c" s t rule*: *big_step_induct*) *auto*

lemma *"deskip c ∼ c"*

Exercise 7.2 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.

General homework instructions

- All proofs in the homework must be carried out in Isar style.
- You can upload multiple files in the submission interface.

Homework 7.1 Loop Compiler

Submission until Tuesday, December 5, 10:00am.

For this exercise we have replaced the normal *WHILE* loop in IMP by a new *Loop c b* construct (without nice syntax). The modified type *com* and the big-step semantics are given at the beginning of the template file. Your task is to define the compiler *ccomp* for the new loop construct and prove the correctness theorem *ccomp_bigstep*; both are found at the end of the template file.

Homework 7.2 Compilation of exceptions

Submission until Tuesday, December 5, 10:00am.

In the previous homework, we extended IMP with the exception throwing and handling constructs *THROW* and *ATTEMPT _ CLEANUP _*. In this homework you have to extend the command compiler *ccomp* to deal with these two constructs. The main idea is simple: a *THROW* is compiled to a *JMP* to the *CLEANUP* code. The new *ccomp* should have type *nat ⇒ com ⇒ instr list*. The additional *nat* parameter has a similar purpose as the *nat* parameter of function *bcomp*: it tells *ccomp* how far beyond the end of the generated code the code should jump in case of a *THROW*. If execution of the source code terminates with *SKIP*, execution of the compiled code should terminate 1 step beyond end of the compiled code; if execution of the source code terminates with *THROW*, execution of the compiled code should jump *n+1* steps beyond the end of compiled code.

Start from the template file. It contains both the language definition (as a big step semantics) and an almost unchanged copy of the compiler. Go to the end of the file and update and extend the definition of function *ccomp*. Check that your compiler does the right thing for the given example and try some more examples. Finally modify the correctness statement *ccomp_bigstep* (for inspiration look at lemma *bcomp_correct*) and update the proof. If your compiler is correct, there is a good chance that you only need to give an explicit proof for case *WhileTrueSkip* in that induction; the other cases should go through automatically (with the help of the final *fastforce+*, which make take a bit of time). However, depending on your compiler you may have to spell out some of the other cases as well.