Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D., Dr. Peter Lammich
Johannes Hölzl

WS 2014/15
25. 11. 2014

Semantics of Programming Languages

Exercise Sheet 7

Exercise 7.1 Type checker as recursive functions

Reformulate the inductive predicates $\Gamma \vdash a : \tau$, $\Gamma \vdash b$ and $\Gamma \vdash c$ as three recursive functions

fun *atype* :: *"tyenv ⇒ aexp ⇒ ty option"*
fun *bok* :: *"tyenv ⇒ bexp ⇒ bool"*
fun *cok* :: *"tyenv ⇒ com ⇒ bool"*

and prove

lemma *atyping_atype*: "$(\Gamma \vdash a : \tau) = (atype\ \Gamma\ a = Some\ \tau)$"
lemma *btyping_bok*: "$(\Gamma \vdash b) = bok\ \Gamma\ b$"
lemma *ctyping_cok*: "$(\Gamma \vdash c) = cok\ \Gamma\ c$"

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 *ccomp_bigstep* to your modified compiler.

Homework 7.1 Non-zero Typing

Submission until Tuesday, December 2, 2014, 10:00am.

Start with a fresh copy of *Types.thy*. Define a language that only knows real values. The binary operators are addition and division (*op /* in Isabelle/HOL). The semantics shall get stuck if trying to divide by zero.

Define a type system, that distinguishes between positive, negative, zero, and unknown signs of variables. Well-typed programs must not divide by zero. Adapt the theory up to the *type_sound*-theorem, i.e., show that in a well-typed program, every reachable non-skip state can make another step.

We only consider real values:

type_synonym *val = real*

datatype *aexp = Rc real | V vname | Plus aexp aexp | Div aexp aexp*

The types are:

datatype *ty = Neg|Pos|Zero|Any*

Hint: For every operator, define a counterpart on types

definition *ty_of_c* :: *"real ⇒ ty"* **where**

fun *ty_of_plus* :: *"ty ⇒ ty ⇒ ty"* **where**
fun *ty_of_div* :: *"ty ⇒ ty ⇒ ty option"* **where**
— A return value of *None* means "not typeable".

The typing rules for arithmetic expressions then become:

inductive *atyping* :: *"tyenv ⇒ aexp ⇒ ty ⇒ bool"*
(*"(1_/ ⊢/ (_ :/ _))" [50,0,50] 50*)
where
Rc_ty: "$\Gamma \vdash Rc\ r : (ty\_of\_c\ r)$" |
V_ty: "$\Gamma \vdash V\ x : \Gamma\ x$" |
Plus_ty: "$\Gamma \vdash a1 : \tau 1 \Longrightarrow \Gamma \vdash a2 : \tau 2 \Longrightarrow \Gamma \vdash Plus\ a1\ a2 : ty\_of\_plus\ \tau 1\ \tau 2$" |
Div_ty: "$\Gamma \vdash a1 : \tau 1 \Longrightarrow \Gamma \vdash a2 : \tau 2 \Longrightarrow ty\_of\_div\ \tau 1\ \tau 2 = Some\ \tau \Longrightarrow \Gamma \vdash Div\ a1\ a2 : \tau$"

Note: Unlike in the original int/real type system, a single value does not have a unique type any more. E.g., the value π is described by both types, *Pos* and *Any*.

However, we can define a function that assigns each type a set of described values:

fun *values_of_type* :: *"ty ⇒ real set"* **where**

Then, a well-typed state is expressed as follows:

definition *styping* :: *"tyenv ⇒ state ⇒ bool"* (**infix** *"⊢" 50*)
where "$\Gamma \vdash s \longleftrightarrow (\forall x.\ s\ x \in values\_of\_type\ (\Gamma\ x))$"

Homework 7.2 Compiling *REPEAT*

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

We extend *com* with a *REPEAT c UNTIL b* statement. With 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 usual schema *FirstnameLastname2.thy* (don't forget to also rename the Isar theory-header).