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

\begin{align*}
\text{fun } \text{atype} &::= \text{"tyenv } \Rightarrow \text{aexp } \Rightarrow \text{ty option"} \\
\text{fun } \text{bok} &::= \text{"tyenv } \Rightarrow \text{bexp } \Rightarrow \text{bool"} \\
\text{fun } \text{cok} &::= \text{"tyenv } \Rightarrow \text{com } \Rightarrow \text{bool"}
\end{align*}

and prove

\begin{align*}
\text{lemma } \text{atyping} &:\text{"}(\Gamma \vdash a : \tau) = (\text{atype }\Gamma a = \text{Some }\tau)" \\
\text{lemma } \text{btyping} &:\text{"}(\Gamma \vdash b) = (\text{bok }\Gamma b)" \\
\text{lemma } \text{ctyping} &:\text{"}(\Gamma \vdash c) = (\text{cok }\Gamma c)"
\end{align*}

Exercise 7.2 Compiler optimization

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

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

(“(L_ ⊢_ / (_. :/ .))” [50,0,50] 50)

where

Rc_ty: “Γ ⊢ Rc r : (ty_of_c r)” |
V ty: “Γ ⊢ V x : Γ x” |
Plus ty: “Γ ⊢ a1 : τ1 ⇒ Γ ⊢ a2 : τ2 ⇒ Γ ⊢ Plus a1 a2 : ty_of_div τ1 τ2” |
Div ty: “Γ ⊢ a1 : τ1 ⇒ Γ ⊢ a2 : τ2 ⇒ τy_of_div τ1 τ2 = Some τ ⇒ Γ ⊢ Div a1 a2 : τ”

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 “Γ ⊢ s ←→ (∀ x. s x ∈ values_of_type (Γ x))”
Homework 7.2  Compiling \texttt{REPEAT}

\textit{Submission until Tuesday, December 2, 10:00am.}

We extend \texttt{com} with a \texttt{REPEAT c UNTIL b} statement. With adding the following rules to our big-step semantics:

\begin{align*}
\text{RepeatTrue: } & \left[ ((c, s_1) \Rightarrow s_2; \text{bval } b \text{ } s_2) \Rightarrow (\text{REPEAT } c \text{ UNTIL } b, s_1) \Rightarrow s_2 \right] \\
\text{RepeatFalse: } & \left[ ((c, s_1) \Rightarrow s_2; \neg \text{bval } b \text{ } s_2; (\text{REPEAT } c \text{ UNTIL } b, s_2) \Rightarrow s_3) \Rightarrow (\text{REPEAT } c \text{ UNTIL } b, s_1) \Rightarrow s_3 \right]
\end{align*}

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

Download the files \texttt{Repeat_Big_Step.thy} and \texttt{Repeat.Compiler_Template.thy}. Finish the definition of \texttt{ccomp} and the proof of \texttt{ccomp_bigstep} in \texttt{Repeat.Compiler_Template.thy}, and submit this theory using as filename the usual schema \texttt{FirstnameLastname2.thy} (don’t forget to also rename the Isar theory-header).