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

WS 2017/18
05.12.2017

Semantics of Programming Languages

Exercise Sheet 8

Exercise 8.1 Type coercions

Adding and comparing integers and reals can be allowed by introducing implicit conversions: Adding an integer and a real results in a real value, comparing an integer and a real can be done by first converting the integer into a real. Implicit conversions like this are called *coercions*.

1. Modify, in the theory *Types*, the inductive definitions of *taval* and *tbval* such that implicit coercions are applied where necessary.
2. Adapt all proofs in the theory *Types* accordingly.

Hint: Isabelle already provides the coercion function *real_of_int* (*int* ⇒ *real*).

Exercise 8.2 Security type system: bottom-up with subsumption

Recall security type systems for information flow control from the lecture. Such a type systems can either be defined in a top-down or in a bottom-up manner. Independently of this choice, the type system may or may not contain a subsumption rule (also called anti-monotonicity in the lecture). The lecture discussed already all but one combination: a bottom-up type system with subsumption.

1. Define a bottom-up security type system for information flow control with subsumption rule (see below, add the subsumption rule).
2. Prove the equivalence of the newly introduced bottom-up type system with the bottom-up type system without subsumption rule from the lecture.

inductive *sec_type2'* :: *"com ⇒ level ⇒ bool"* (*"(⊢' _ : _)"* *[0,0] 50*) **where**
Skip2': *"⊢' SKIP : l"* |
Assign2': *"sec x ≥ sec a ⟹ ⊢' x ::= a : sec x"* |
Semi2': *"⟦ ⊢' c_1 : l; ⊢' c_2 : l ⟧ ⟹ ⊢' c_1 ;; c_2 : l"* |
If2': *"⟦ sec b ≤ l; ⊢' c_1 : l; ⊢' c_2 : l ⟧ ⟹ ⊢' IF b THEN c_1 ELSE c_2 : l"* |
While2': *"⟦ sec b ≤ l; ⊢' c : l ⟧ ⟹ ⊢' WHILE b DO c : l"*

General homework instructions

All proofs in the homework must be carried out in Isar style.

Homework 8.1 Security type systems: bottom-up vs. top-down

Submission until Tuesday, December 12, 10:00am.

Prove the equivalence of the bottom-up system ($\vdash$ _ : _) and the top-down system (_ $\vdash$ _) without subsumption rule. Carry out a direct correspondence proof in both directions without using the $\vdash'$ system.

lemma *bottom_up_impl_top_down*: "$\vdash c : l \Longrightarrow l \vdash c$"
lemma *top_down_impl_bottom_up*: "$l \vdash c \Longrightarrow \exists\ l' \geq l.\ \vdash c : l'$"

Homework 8.2 Explicit type coercions

Submission until Tuesday, December 12, 10:00am.

In the tutorial, we have introduced *implicit* coercions in the typing of arithmetic and boolean expressions. Here, we want to use *explicit* coercions. In particular, we want to

- add a new constructor *Real* :: *aexp* $\Rightarrow$ *aexp* to *aexp*,
- extend *taval* to support this new constructor,
- extend *atyping* and *btyping* to return an expression with coercions inserted (which we will call an *elaborated expression*), and
- treat addition and comparison of different types as runtime errors in the semantics.

Copy and modify *Types* as necessary, including all proofs below (you don't have to implement nor prove soundness for command elaboration).
Note: It is not recommended to keep the existing introduction and elimination rules, as they might make the automated tactics loop.

datatype *aexp* = *Ic int* | *Rc real* | *Real aexp* | *V vname* | *Plus aexp aexp*

Note that coercing an arithmetic expression of type *Rty* using *Real* should be considered a type error. Your elaboration implementation should add coercions where possible and necessary, but you're free to insert them where they fit. For example, the expression *Plus* (*Rc 0*) (*Plus* (*Ic 1*) (*Ic 2*)) could be elaborated to *Plus* (*Rc 0*) (*Real* (*Plus* (*Ic 1*) (*Ic 2*))).

datatype *bexp* = *Bc bool* | *Not bexp* | *And bexp bexp* | *Less aexp aexp*

inductive *aelab* :: *"tyenv ⇒ aexp ⇒ aexp ⇒ ty ⇒ bool"*
(*"(1_/ ⊢/ (_ ⇝ _ :/ _))" [50,0,0,50] 50*)
inductive *belab* :: *"tyenv ⇒ bexp ⇒ bexp ⇒ bool"* (*"(1_/ ⊢/ (_ ⇝ _))" [50,0,50] 50*)

Syntax examples: $\Gamma \vdash t \rightsquigarrow t' : \tau$ and $\Gamma \vdash t \rightsquigarrow t' : \tau$

You have to come up with the following lemma statements yourself.

lemma *apreservation*:
lemma *aprogress*:
lemma *bprogress*:

Is your *aelab* predicate deterministic? If yes, give an informal proof sketch, if no, give a counterexample.