Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Fabian Huch, Kevin Kappelmann

WS 23/24
5.12.2023

Semantics of Programming Languages

Exercise Sheet 7

There are strikes on Dec 5 for fair inflation-adjusted wages as well as tariff contracts for student employees, starting at 9:30 at the bakery on the campus Garching – so no tutorial session!

Exercise 7.1 Security type system: bottom-up with subsumption

Recall security type systems for information flow control from the lecture. Such a type system 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.

- Define a bottom-up security type system for information flow control with subsumption rule (see below, add the subsumption rule).
- 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': "$\vdash'$ *SKIP* : *l*" |
Assign2': "$sec\ x \geq sec\ a \Longrightarrow\ \vdash' x ::= a : sec\ x$" |
Seq2': "$\llbracket \vdash' c_1 : l;\ \vdash' c_2 : l \rrbracket \Longrightarrow\ \vdash' c_1\ ;;\ c_2 : l$" |
If2': "$\llbracket sec\ b \leq l;\ \vdash' c_1 : l;\ \vdash' c_2 : l \rrbracket \Longrightarrow\ \vdash' IF\ b\ THEN\ c_1\ ELSE\ c_2 : l$" |
While2': "$\llbracket sec\ b \leq l;\ \vdash' c : l \rrbracket \Longrightarrow\ \vdash' WHILE\ b\ DO\ c : l$"

lemma "$\vdash c : l \Longrightarrow\ \vdash' c : l$"

lemma "$\vdash' c : l \Longrightarrow \exists\, l' \geq l.\ \vdash c : l'$"

Homework 7 Security Types

Submission until Monday, December 11, 23:59pm.

Prove the equivalence of the bottom-up system (⊢ _ : _) and the top-down system (_ ⊢ _) without subsumption rule. Carry out a direct correspondence proof in both directions without using the ⊢′ system. Give an Isar proof - no *apply* allowed!

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

Homework 7.1 Definite Initialization Analysis

Submission until Monday, Dec 11, 23:59pm.

In the lecture, you have seen a definite initialization analysis that was based on the big-step semantics. Definite initialization analysis can also be based on a small-step semantics. For this, the ternary predicate D from the lecture can be split into two parts: a function AV :: *com* ⇒ *vname set*, which collects the names of all variables assigned by a command, and a relation D :: *vname set* ⇒ *com* ⇒ *bool*, which checks whether a command accesses only previously assigned variables. Conceptually, the ternary predicate from the lecture (call it D_{lec}) and the two-step approach should be related by the equivalence $D\ V\ c \longleftrightarrow D_{lec}\ V\ c\ (V \cup AV\ c)$

Define the functions Av and D, where the latter checks whether a command accesses only assigned variables assuming that the variables in the argument set are already assigned).

fun AV :: *"com ⇒ vname set"*
fun D :: *"vname set ⇒ com ⇒ bool"*

The following two proofs are already given as they are quite similar to D_{lec}. If the proof doesn't work, check your definitions (and adapt the proof, if necessary)! You may not define any auxiliary lemma (due to the constraints of the next part).

Now we want to prove preservation of D with respect to the small-step semantics, and from progress and preservation conclude soundness of D. You may need the lemmas *D_incr* and *D_mono* for this.

Proofs like this can often seem magical when using the full proof automation of Isabelle, which makes them very hard to read. Hence, your proofs for this exercise must *not* contain any of the advanced solvers like *fastforce*, *smt*, *auto*, etc. This means that only the following proof methods may be used:

- *cases/induction*
- *rule/intro/erule/drule/frule*
- *./..*
- *simp/simp_all*
- *blast*

Of course all of the Isar syntax (fix, obtain, ...), forward proofs, instantiation, local lemmas, etc. are still allowed. Give the proofs in Isar. No *apply*s are allowed!

Note: The submission system will not check these constraints but the graders will.

lemma *D_mono*: "$A \subseteq A' \Longrightarrow D\ A\ c \Longrightarrow D\ A'\ c$"

theorem *D_preservation*: "$(c,s) \rightarrow (c',s') \Longrightarrow D\ (dom\ s)\ c \Longrightarrow D\ (dom\ s')\ c'$"

theorem *D_sound*: "$(c,s) \rightarrow\!*\ (c',s') \Longrightarrow c' \neq SKIP \Longrightarrow D\ (dom\ s)\ c \Longrightarrow \exists\, cs''.\ (c',s') \rightarrow cs''$"