Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Kevin Kappelmann, Lukas Stevens

WS 24/25
12.12.2023

Semantics of Programming Languages

Exercise Sheet 9

Exercise 9.1 Denotational Semantics

Define a denotational semantics for *REPEAT c UNTIL b*-loops that run a command c (at least once) until b is true.

datatype *com* = *SKIP*
  | *Assign vname aexp* ("_ ::= _" [1000, 61] 61)
  | *Seq com com* ("_;;/ _" [60, 61] 60)
  | *If bexp com com* ("(IF _/ THEN _/ ELSE _)" [0, 0, 61] 61)
  | *While bexp com* ("(WHILE _/ DO _)" [0, 61] 61)
  | *Repeat com bexp* ("(REPEAT _/ UNTIL _)" [0, 61] 61)

inductive
  big_step :: "com × state ⇒ state ⇒ bool" (**infix** "⇒" 55)
where
Skip: "(SKIP,s) ⇒ s" |
Assign: "(x ::= a,s) ⇒ s(x := aval a s)" |
Seq: "⟦ $(c_1,s_1) \Rightarrow s_2$; $(c_2,s_2) \Rightarrow s_3$ ⟧ ⟹ $(c_1;;c_2,\ s_1) \Rightarrow s_3$" |
IfTrue: "⟦ bval b s; $(c_1,s) \Rightarrow t$ ⟧ ⟹ (IF b THEN c_1 ELSE c_2, s) ⇒ t" |
IfFalse: "⟦ ¬bval b s; $(c_2,s) \Rightarrow t$ ⟧ ⟹ (IF b THEN c_1 ELSE c_2, s) ⇒ t" |
WhileFalse: "¬bval b s ⟹ (WHILE b DO c,s) ⇒ s" |
WhileTrue: "⟦ bval b s_1; $(c,s_1) \Rightarrow s_2$; (WHILE b DO c, s_2) ⇒ s_3 ⟧
  ⟹ (WHILE b DO c, s_1) ⇒ s_3"

type_synonym *com_den* = "(state × state) set"

definition *W* :: "(state ⇒ bool) ⇒ com_den ⇒ (com_den ⇒ com_den)" **where**
"W db dc = (λdw. {(s,t). if db s then (s,t) ∈ dc O dw else s=t})"

fun *D* :: "com ⇒ com_den" **where**
"D SKIP = Id" |
"D (x ::= a) = {(s,t). t = s(x := aval a s)}" |
"D (c1;;c2) = D(c1) O D(c2)" |
"D (IF b THEN c1 ELSE c2)
 = {(s,t). if bval b s then (s,t) ∈ D c1 else (s,t) ∈ D c2}" |
"D (WHILE b DO c) = lfp (W (bval b) (D c))"

Exercise 9.2 Chains

A function $c :: nat \Rightarrow 'a$ is called an $\omega-chain$ on $'a$ if and only if:

definition *"ωchain (c :: nat ⇒ 'a::order) ≡ ∀n. c n ≤ c (Suc n)"*

lemma *ωchainI* [*intro*]:
  assumes *"⋀n. c n ≤ c (Suc n)"*
 shows *"ωchain c"*
 unfolding *ωchain_def* **using** *assms* **by** *blast*

Next, we set up the lifting of a partial order on $'a$ to a partial order on $'a$ *option*, defined in the expected way - don't worry about the specifics here, you will learn about *instantiation* later in the course.

instantiation *option* :: (*order*) *order*
begin

fun *less_eq_option* :: *"'a option ⇒ 'a option ⇒ bool"* **where**
 "None ≤ _ ⟷ True"
| *"_ ≤ None ⟷ False"*
| *"Some x ≤ Some y ⟷ x ≤ y"*

definition *"(x :: 'a option) < y ≡ x ≤ y ∧ x ≠ y"*

instance by *standard*
 (*force simp*: *less_option_def elim*!: *less_eq_option.elims intro*: *less_eq_option.elims*(*1*))+
end

We want to show that every (non-empty) $\omega-chain$ on $'a$ *option* induces an $\omega-chain$ on $'a$.

Complete the following as a structured Isar proof. It is recommended (but not mandatory) to follow the given proof structure. You must only use *simp*, *auto*, *blast*, *fastforce*, *cases* as proof methods. You must not use *apply*, *metis*, *meson*, *smt*, etc.

Recall that definitions within a lemma statement are available under the usual ..._*def* name.

declare *Suc_lessI*[*intro*]

theorem *option_chain*:
 assumes *chainc*: *"ωchain (c :: nat ⇒ ('a :: order) option)"*
 and *cn_eq*: *"c n_0 = Some x"*
 and *clt*: *"⋀m. m < n_0 ⟹ c m = None"*
 defines *"c' ≡ λn. case c n of None ⇒ x | Some y ⇒ y"*
 shows *"ωchain c'"*
proof (*rule ωchainI*)
 have *cge*: *"⋀m. n_0 ≤ m ⟹ c m = Some (c' m)"*
 proof −
  fix *m* **assume** *"n_0 ≤ m"*

then show *"c m = Some (c′ m)"*
qed

fix *m* **show** *"c′ m ≤ c′ (Suc m)"*
qed

Homework 9.1 Denotational Semantics (5 points)

Submission until Wednesday, Dec 18, 23:59pm.

We again consider the extension of IMP with non-determinism from exercise sheet 5. This time, we also add a construct *LOOP c* for non-deterministic looping. The idea is that *LOOP c* can non-deterministically decide to either stop iteration and do nothing or to execute the loop body *c* for one more time.

datatype *com = SKIP | Assign (char list) aexp | Seq com com | com.If bexp com com | While bexp com | Or com com | ASSUME bexp | Loop com*

First extend the big-step semantics with this new construct:

inductive
big_step :: *"com × state ⇒ state ⇒ bool"* (**infix** *"⇒" 55*)
where
Skip: *"(SKIP,s) ⇒ s"* |
Assign: *"(x ::= a,s) ⇒ s(x := aval a s)"* |
Seq: "⟦ $(c_1,s_1) \Rightarrow s_2$; $(c_2,s_2) \Rightarrow s_3$ ⟧ $\Longrightarrow (c_1;;c_2,\ s_1) \Rightarrow s_3$" |
IfTrue: "⟦ $bval\ b\ s$; $(c_1,s) \Rightarrow t$ ⟧ $\Longrightarrow (IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s) \Rightarrow t$" |
IfFalse: "⟦ $\neg bval\ b\ s$; $(c_2,s) \Rightarrow t$ ⟧ $\Longrightarrow (IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s) \Rightarrow t$" |
WhileFalse: "$\neg bval\ b\ s \Longrightarrow (WHILE\ b\ DO\ c,s) \Rightarrow s$" |
WhileTrue: "⟦ $bval\ b\ s_1$; $(c,s_1) \Rightarrow s_2$; $(WHILE\ b\ DO\ c,\ s_2) \Rightarrow s_3$ ⟧ $\Longrightarrow (WHILE\ b\ DO\ c,\ s_1) \Rightarrow s_3$" |
OrLeft: "⟦ $(c_1,s) \Rightarrow s'$ ⟧ $\Longrightarrow (c_1\ OR\ c_2,s) \Rightarrow s'$" |
OrRight: "⟦ $(c_2,s) \Rightarrow s'$ ⟧ $\Longrightarrow (c_1\ OR\ c_2,s) \Rightarrow s'$" |
Assume: "$bval\ b\ s \Longrightarrow (ASSUME\ b,\ s) \Rightarrow s$" |
— Your cases here:

declare *big_step.intros* [*intro*]
lemmas *big_step_induct = big_step.induct*[*split_format*(*complete*)]

inductive_cases *skipE*[*elim!*]: *"(SKIP,s) ⇒ t"*
inductive_cases *AssignE*[*elim!*]: *"(x ::= a,s) ⇒ t"*
inductive_cases *SeqE*[*elim!*]: *"(c1;;c2,s1) ⇒ s3"*
inductive_cases *OrE*: *"(c1 OR c2,s1) ⇒ s3"*
inductive_cases *AssumeE* [*elim!*]: *"(ASSUME b, s1) ⇒ s2"*
inductive_cases *IfE*[*elim!*]: *"(IF b THEN c1 ELSE c2,s) ⇒ t"*
inductive_cases *WhileE*[*elim*]: *"(WHILE b DO c,s) ⇒ t"*

Now, give a denotational semantics for this language:

type_synonym *com_den = "(state × state) set"*

fun D :: *"com ⇒ com_den"* **where**
"D SKIP = Id" |
"D (x ::= a) = {(s,t). t = s(x := aval a s)}" |
"D (c1;;c2) = D(c1) O D(c2)" |
"D (IF b THEN c1 ELSE c2)
= {(s,t). if bval b s then (s,t) ∈ D c1 else (s,t) ∈ D c2}" |
"D (WHILE b DO c) = lfp (W (bval b) (D c))"
— Your cases here:

Then correct the proof of the equivalence theorem between big-step and denotational semantics:

theorem *denotational_is_big_step*:
"(s,t) ∈ D(c) = ((c,s) ⇒ t)"

Use theory *HOL−IMP.Denotational* as a template for the proof!

Homework 9.2 Formalization of Formal Language Theory

Submission until Wednesday, Jan 8, 23:59pm.

Over the next few weeks, you will have the opportunity to create some new Isabelle theories outside of semantics.

We are currently working on an all-singing, all-dancing formalization of the *theory of regular and context-free gramars and languages.* Some of it is already in the Archive of Formal Proofs (e.g. Regular Expressions, Finite Automata) and some is still in private repositories. We are in the process of unifying all of it — and of adding some new, never-before formalized parts of the theory. We want you to help us in building up this unique formalization of a fundamental area of computer science. Here are some of the building blocks that you can help with, from easy to difficult:

- Grammar cleaning: Elimination of
  - unit productions
  - epsilon productions
  - unreachable symbols
  - unproductive symbols
- Deciding if a word is in $L(G)$.
- Converting between right-linear grammars and regular expressions
- Punping Lemma applications:
  - Prove that $a^n b^n c^n$ is not context-free
  - Prove that context-free languages are not closed under intersection.
- Conversions to Greibach Normal Form

- Ogden's Lemma
- Parikh's Theorem – this one has never been formalized!
- Existence of an inherently ambiguous language

If you are interested, you must get in touch with Tobias Nipkow (who coordinates the whole enterprise), discuss the topics with him and "sign up" for a specific topic. Email: nipkow@in.tum.de Difficult topics can also be tackled by teams of two students by request (include your request in your email). We can generate more topics on demand.

If for some reason you want to avoid context-free languages, you can also formalize some topic of your own choice from any area of mathematics or computer science but it should contain some interesting proof(s). Creativity is encouraged and will be rewarded, but keep in mind that formalizations can often be more difficult than anticipated. Set yourself realistic goals. We recommend to discuss your project with one of the tutors beforehand.

Whatever topic you decide to work on:

- **Aim for readable, structured proofs.**
- **Comment your formalization well. We need to read and understand it.**
- **Incomplete or unfinished formalizations are welcome and will be graded (but clean them up so it is obvious what is there and what is missing).**

In total, this exercise will be worth 15 points, plus bonus points for nice submissions. (We value your work and have awarded up to 30 points in the past.)