Universität des Saarlandes
Fachrichtung Informatik
Dr. Jasmin Blanchette
Mathias Fleury

SS 2015
01.06.2015

Concrete Semantics with Isabelle/HOL

Exercise Sheet 5

Exercise 5.1 Program Equivalence

Prove or disprove (by giving counterexamples) the following program equivalences.

1. *IF And b1 b2 THEN c1 ELSE c2* $\sim$ *IF b1 THEN IF b2 THEN c1 ELSE c2 ELSE c2*
2. *WHILE And b1 b2 DO c* $\sim$ *WHILE b1 DO WHILE b2 DO c*
3. *WHILE And b1 b2 DO c* $\sim$ *WHILE b1 DO c;; WHILE And b1 b2 DO c*
4. *WHILE Or b1 b2 DO c* $\sim$ *WHILE Or b1 b2 DO c;; WHILE b1 DO c*

Hint: Use the following definition for *Or*:

definition *Or* :: *"bexp* $\Rightarrow$ *bexp* $\Rightarrow$ *bexp"* **where**
"Or b1 b2 = Not (And (Not b1) (Not b2))"

Exercise 5.2 Nondeterminism

In this exercise we extend our language with nondeterminism. We will define *nondeterministic choice* (c_1 *OR* c_2), that decides nondeterministically to execute c_1 or c_2; and *assumption* (*ASSUME* b), that behaves like *SKIP* if b evaluates to true, and returns no result otherwise.

1. Modify the datatype *com* to include the new commands *OR* and *ASSUME*.
2. Adapt the big-step semantics to include rules for the new commands.
3. Prove that c_1 *OR* $c_2 \sim c_2$ *OR* c_1.
4. Prove: (*IF b THEN c1 ELSE c2*) $\sim$ ((*ASSUME b*; *c1*) *OR* (*ASSUME* (*Not b*); *c2*))
5. Optional (relies on material from next week): Adapt the small step semantics, and the equivalence proof of big and small step semantics.

Note: It is easiest if you take the existing theories and modify them.

Homework 5.1 Dijkstra's Guarded Command Language (12 points)

Submission until Monday 08.06.2015 10:01 AM. Please send your submissions to jasmin.blanchette@mpi-inf.mpg.de and mathias.fleury@mpi-inf.mpg.de with "[CONCRETE5]" in the subject of your email. If your name is Fornavn Etternavn, please call your theory file `Etternavn_Fornavn.thy`.

Unless indicated otherwise, please give all your proofs in Isar, not `apply` style.

In the 1970s, Edsger Dijkstra introduced the guarded command language (GCL), a nondeterministic programming language featuring a nondeterministic `if` command with the syntax

```
if b1 --> c1
 | b2 --> c2
 ...
 | bN --> cN
fi
```

where `b1`, `b2`, ..., `bN` are Boolean conditions and `c1`, `c2`, ..., `cN` are commands. When executing the statement, an arbitrary branch with a condition that evaluates to true is selected. If no condition is true, execution simply blocks.

To keep things simple, we will have no looping command in our language. Here is the Isabelle datatype:

datatype *gcom* =
  Skip
| *Ass vname aexp*
| *Sq gcom gcom*
| *IfBlock "(bexp × gcom) list"*

First, define the big-step semantics with infix syntax $\Rightarrow g$:

inductive
  big_stepg :: *"gcom × state ⇒ state ⇒ bool"* (**infix** *"⇒g" 55*)

Use `~~/src/HOL/IMP/Big_Step.thy` as an inspiration. Remember to give names to your introduction rules, so that you can refer to each rule by name in your proofs by rule induction.

Like in `~~/src/HOL/IMP/Big_Step.thy`, we declare the introduction rules as *intro* rules for *auto*, *blast*, *fast*, *fastforce*, *force*, and *extreme_violence*. We also do some magic with the induction rule to make it more suitable—this is necessary only because the first argument to $\Rightarrow g$ is tupled.

declare *big_stepg.intros* [*intro*]

lemmas *big_stepg_induct* = *big_stepg.induct*[*split_format*(*complete*)]

This will come in handy later:

inductive_cases *SqE*: "$(Sq\ c1\ c2,\ s) \Rightarrow g\ t$"
inductive_cases *IfBlockE*: "$(IfBlock\ Gs,\ s) \Rightarrow g\ t$"

thm *SqE IfBlockE*

A useful lemma. Prove it:

lemma *IfBlock_subset_big_stepg*:
  assumes
    Gs: "$(IfBlock\ Gs,\ s) \Rightarrow g\ s'$" **and**
    Gs': "$set\ Gs \subseteq set\ Gs'$"
  shows "$(IfBlock\ Gs',\ s) \Rightarrow g\ s'$"

Write various schematic lemmas in the style of `~~/src/HOL/IMP/Big_Step.thy` and try out your big-step semantics on them. In particular, try to take both branches of a two-way `if` block. For this part, you are allowed (indeed, encouraged) to write your proofs in **apply** style.

schematic_lemma *ex1*: "$(Sq\ (Ass\ ''x''\ (N\ 5))\ (Ass\ ''y''\ (V\ ''x'')),\ s) \Rightarrow g\ ?s'$"

...

thm *ex1*[*simplified*]

schematic_lemma *ex2*: "$(IfBlock\ [(Less\ (N\ 4)\ (N\ 5),\ Ass\ ''x''\ (N\ 2))],\ s) \Rightarrow g\ ?s'$"

...

thm *ex2*[*simplified*]

schematic_lemma *ex3a*:
  "$(IfBlock\ [(Less\ (N\ 4)\ (N\ 5),\ Ass\ ''x''\ (N\ 2)),\ (Less\ (N\ 6)\ (N\ 7),\ Ass\ ''x''\ (N\ 3))],\ s)$
  $\Rightarrow g\ ?s'$"

...

thm *ex3a*[*simplified*]

schematic_lemma *ex3b*:
  "$(IfBlock\ [(Less\ (N\ 4)\ (N\ 5),\ Ass\ ''x''\ (N\ 2)),\ (Less\ (N\ 6)\ (N\ 7),\ Ass\ ''x''\ (N\ 3))],\ s)$
  $\Rightarrow g\ ?s'$"

...

thm *ex3b*[*simplified*]

Is the language deterministic? Prove or disprove.

lemma "$(c,\ s) \Rightarrow g\ s' \Longrightarrow (c,\ s) \Rightarrow g\ s'' \Longrightarrow s' = s''$"

Is the language total? Prove or disprove.

lemma "$\exists\, s'.\ (c,\ s) \Rightarrow g\ s'$"

Next, define the semantics as a function. In cases where several guard conditions evaluate to true, it can arbitrarily select a true branch (e.g., the first true branch). If the program blocks, the function returns *None*.

fun *big_stepgf* :: *"gcom ⇒ state ⇒ state option"*

Specify names for the cases of the induction rule generated for the above function, to enhance the readability of Isar proofs (i.e., replace ... with appropriate names):

lemmas *big_stepgf_induct* = *big_stepgf.induct*[*case_names* ...]

A useful lemma. Prove it:

lemma *big_stepgf_Some_imp_ex_inter*:
"big_stepgf (Sq c c′) s = Some s′′ ⟹
∃ s′. big_stepgf c s = Some s′ ∧ big_stepgf c′ s′ = Some s′′"

Prove or disprove that the function *big_stepgf* is sound with respect to the inductive predicate *op ⇒g*:

theorem *big_stepgf_sound*: *"big_stepgf c s = Some s′ ⟹ (c, s) ⇒g s′"*

Hint: If you go for a proof, make sure to use the most appropriate induction principle, and ask yourself whether you need *arbitrary*:.

Finally, prove or disprove completeness:

lemma *big_stepgf_complete*: *"(c, s) ⇒g s′ ⟹ ∃ t. big_stepgf c s = Some t"*

Homework 5.2 More GCL (8 points)

Submission until Monday 08.06.2015 10:01 AM. Please send your submissions to jasmin.blanchette@mpi-inf.mpg.de and mathias.fleury@mpi-inf.mpg.de with "[CONCRETE5]" in the subject of your email. If your name is Fornavn Etternavn, please call your theory file `Etternavn_Fornavn.thy`.

Please give all your proofs in Isar, not `apply` style.

This is a continuation of the previous homework exercise.

Define a notion of program equivalence for GCL:

abbreviation *equiv_cg* :: *"gcom ⇒ gcom ⇒ bool"* (**infix** *"∼g" 50*)

Show that $\sim g$ is an equivalence relation:

lemma *reflp_equiv_cg*: *"reflp (op ∼g)"*
lemma *symp_equiv_cg*: *"symp (op ∼g)"*
lemma *transpp_equiv_cg*: *"transp (op ∼g)"*

Prove the congruence lemma for *Sq*:

lemma *Sq_cong*:
  assumes
    c1: "$c1 \sim g\ c1'$" **and**
    c2: "$c2 \sim g\ c2'$"
  shows "$Sq\ c1\ c2 \sim g\ Sq\ c1'\ c2'$"

Homework 5.3 Even more GCL (5 **bonus** points)

Submission until Monday 08.06.2015 10:01 AM. Please send your submissions to jasmin.blanchette@mpi-inf.mpg.de and mathias.fleury@mpi-inf.mpg.de with "[CONCRETE5]" in the subject of your email. If your name is Fornavn Etternavn, please call your theory file `Etternavn_Fornavn.thy`.

Please give all your proofs in Isar, not `apply` style.

This is a continuation of the previous homework exercise. **Warning:** This is a difficult exercise. Do not spend too much time on it.

Prove the congruence lemma for *IfBlock*. There are many ways of stating it. Use whichever style you prefer, including these:

lemma *IfBlock_cong_v1*:
  assumes *allc*: "$\forall (b, (c, c')) \in set\ GGs.\ c \sim g\ c'$"
  shows
    "$IfBlock\ (map\ (\lambda(b, (c, \_)).\ (b, c))\ GGs)$
    $\sim g\ IfBlock\ (map\ (\lambda(b, (\_, c')).\ (b, c'))\ GGs)$"

lemma *IfBlock_cong_v2*:
  assumes *len*: "$length\ bs = n$" "$length\ cs = n$" "$length\ cs' = n$"
  assumes *alli*: "$\bigwedge i.\ i < n \Longrightarrow cs\ !\ i \sim g\ cs'\ !\ i$"
  shows "$IfBlock\ (zip\ bs\ cs) \sim g\ IfBlock\ (zip\ bs\ cs')$"

The ! operator is syntactic sugar for *nth*, defined in *List*. It returns the $(n-1)$st element of a list (not the nth!).

Finally, show that the order of the elements in the guard block list and any duplicates are irrelevant:

lemma *IfBlock_set_eq_cong*:
  assumes *set*: "$set\ Gs = set\ Gs'$"
  shows "$IfBlock\ Gs \sim g\ IfBlock\ Gs'$"

The end is the beginning is the

end