Technische Universität München
Fakultät für Informatik
Prof. Tobias Nipkow, Ph.D.
Simon Wimmer

WS 2019/20
25.11.2019

Semantics of Programming Languages

Exercise Sheet 6

Exercise 6.1 Compiler optimization

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

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

Exercise 6.2 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 *HOL−IMP.Types* (copy it first), the inductive definitions of *taval* and *tbval* such that implicit coercions are applied where necessary.
2. Adapt all proofs in the theory *HOL−IMP.Types* accordingly.

Hint: Isabelle already provides the coercion function *real_of_int* ($int \Rightarrow real$).

Homework 6.1 Pairs

Submission until Monday, December 6, 10:00am.

In this exercise, we extend the expression language of *IMP* with pair values.

datatype *val* = *Iv int* | *Pv val val*

type_synonym *vname* = *string*
type_synonym *state* = *"vname ⇒ val"*

datatype *aexp* = *N int* | *V vname* | *Plus aexp aexp* | *Pair aexp aexp*

Complete the following inductive predicate for evaluating expressions:

inductive *taval* :: *"aexp ⇒ state ⇒ val ⇒ bool"* **where**
"taval (N i) s (Iv i)" |
"taval (V x) s (s x)" |

It should also be able to add pairs. In this case, the addition should be performed pair-wise. This should also work for nested pairs.

For simplicity, we do not modify Boolean expressions. *Less* can only compare two integer values:

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

inductive *tbval* :: *"bexp ⇒ state ⇒ bool ⇒ bool"* **where**
"tbval (Bc v) s v" |
"tbval b s bv ⟹ tbval (Not b) s (¬ bv)" |
"tbval b1 s bv1 ⟹ tbval b2 s bv2 ⟹ tbval (And b1 b2) s (bv1 & bv2)" |
"taval a1 s (Iv i1) ⟹ taval a2 s (Iv i2) ⟹ tbval (Less a1 a2) s (i1 < i2)"

We add an assignment construct for pairs (x, y) ::== a to the command language:

datatype
  com = *SKIP*
    | *Assign vname aexp* (*"_ ::= _"* [1000, 61] 61)
    | *AssignP "vname × 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)

Adopt the small-step semantics accordingly:

inductive
  small_step :: *"(com × state) ⇒ (com × state) ⇒ bool"* (**infix** *"→"* 55)

We now also add a pair type to the typing system:

datatype *ty* = *Ity* | *Pty ty ty*

type_synonym *tyenv* = *"vname ⇒ ty"*

Complete the typing rules:

inductive *atyping* :: *"tyenv ⇒ aexp ⇒ ty ⇒ bool"*
  (*"(1_/ ⊢/ (_ :/ _))" [50,0,50] 50*)
where
Ic_ty: "$\Gamma \vdash$ *N i* : *Ity*" |
V_ty: "$\Gamma \vdash$ *V x* : Γ *x*" |

inductive *btyping* :: *"tyenv ⇒ bexp ⇒ bool"* (**infix** *"⊢" 50*)
where
B_ty: "$\Gamma \vdash$ *Bc v*" |
Not_ty: "$\Gamma \vdash b \Longrightarrow \Gamma \vdash$ *Not b*" |
And_ty: "$\Gamma \vdash b1 \Longrightarrow \Gamma \vdash b2 \Longrightarrow \Gamma \vdash$ *And b1 b2*" |

inductive *ctyping* :: *"tyenv ⇒ com ⇒ bool"* (**infix** *"⊢" 50*) **where**
Skip_ty: "$\Gamma \vdash$ *SKIP*" |
Assign_ty: "$\Gamma \vdash a : \Gamma(x) \Longrightarrow \Gamma \vdash x ::= a$" |
Seq_ty: "$\Gamma \vdash c1 \Longrightarrow \Gamma \vdash c2 \Longrightarrow \Gamma \vdash c1;;c2$" |
If_ty: "$\Gamma \vdash b \Longrightarrow \Gamma \vdash c1 \Longrightarrow \Gamma \vdash c2 \Longrightarrow \Gamma \vdash$ *IF b THEN c1 ELSE c2*" |
While_ty: "$\Gamma \vdash b \Longrightarrow \Gamma \vdash c \Longrightarrow \Gamma \vdash$ *WHILE b DO c*" |

This function determines the type of a value:

fun *type* :: *"val ⇒ ty"* **where**
"type (Iv i) = Ity" |
"type (Pv v1 v2) = Pty (type v1) (type v2)"

lemma *type_eq_Ity*[*simp*]: *"type v = Ity ⟷ (∃ i. v = Iv i)"*
by (*cases v*) *simp_all*

Hint: You will also need a similar lemma for *Pty t1 t2*.

definition *styping* :: *"tyenv ⇒ state ⇒ bool"* (**infix** *"⊢" 50*)
where "$\Gamma \vdash s \longleftrightarrow (\forall x.\ type\ (s\ x) = \Gamma\ x)$"

Complete the proofs of preservation and progress:

lemma *apreservation*:
  "$\Gamma \vdash a : \tau \Longrightarrow taval\ a\ s\ v \Longrightarrow \Gamma \vdash s \Longrightarrow type\ v = \tau$"
lemma *aprogress*: "$\Gamma \vdash a : \tau \Longrightarrow \Gamma \vdash s \Longrightarrow \exists v.\ taval\ a\ s\ v$"
lemma *bprogress*: "$\Gamma \vdash b \Longrightarrow \Gamma \vdash s \Longrightarrow \exists v.\ tbval\ b\ s\ v$"
theorem *progress*:
  "$\Gamma \vdash c \Longrightarrow \Gamma \vdash s \Longrightarrow c \neq SKIP \Longrightarrow \exists cs'.\ (c,s) \rightarrow cs'$"
theorem *styping_preservation*:
  "$(c,s) \rightarrow (c',s') \Longrightarrow \Gamma \vdash c \Longrightarrow \Gamma \vdash s \Longrightarrow \Gamma \vdash s'$"
theorem *ctyping_preservation*:
  "$(c,s) \rightarrow (c',s') \Longrightarrow \Gamma \vdash c \Longrightarrow \Gamma \vdash c'$"
abbreviation *small_steps* :: *"com ∗ state ⇒ com ∗ state ⇒ bool"* (**infix** *"→∗" 55*)

where *"x →* y == star small_step x y"*

Finally, we can recover the proof of type-soundness:

theorem *type_sound*:
  $“(c,s) \to\!* (c',s') \Longrightarrow \Gamma \vdash c \Longrightarrow \Gamma \vdash s \Longrightarrow c' \neq SKIP$
  $\Longrightarrow \exists\, cs''.\ (c',s') \to cs''”$
apply(*induction rule*:*star_induct*)
apply (*metis progress*)
by (*metis styping_preservation ctyping_preservation*)

Homework 6.2 Continue

Submission until Monday, December 2, 10:00am.

Your task is to add a continue command to the IMP language. The continue command should skip all remaining commands in the innermost while loop.

The new command datatype is:

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*)
  | *CONTINUE*

The idea of the big-step semantics is to return not only a state, but also a continue flag, which indicates that a continue has been triggered. Modify/augment the big-step rules accordingly:

inductive
  big_step :: $“com \times state \Rightarrow bool \times state \Rightarrow bool”$ (**infix** *"⇒"* *55*)

Your next task is to adopt the compiler such that *CONTINUE* is also supported. The now compiler will have the following signature:

fun *ccomp* :: $“com \Rightarrow nat \Rightarrow instr\ list”$ **where**

The extra argument keeps track of the offset from the head of the last preceding while-loop.

To improve automation, first prove the following lemma:

definition
  "len_of c = length (ccomp c 0)"

lemma *length_ccomp*[*simp*]:
  "length (ccomp c i) = len_of c"

Now show that your new compiler is correct. To do so, prove the following modified correctness lemma. The modified lemma adds an instruction prefix *pre*, which you can think of as the list of instructions that separates the current instruction and the last loop head.

Note that the original proof made us of heavy automation that is likely going to break after making the changes from above. Use Isar to explore the proof in more detail.

lemma *ccomp_bigstep1*:
"$(c,s) \Rightarrow (f,\ t) \Longrightarrow i \le \mathit{length\ pre}$
$\Longrightarrow \mathit{pre}\ @\ \mathit{ccomp}\ c\ i \vdash$
$(\mathit{length\ pre},s,\mathit{stk}) \rightarrow\!*\ (\mathit{if}\ f\ \mathit{then\ length\ pre} - i\ \mathit{else\ size}(\mathit{pre}\ @\ \mathit{ccomp}\ c\ i),t,\mathit{stk})$"

Finally, re-prove the old correctness theorem:

corollary *ccomp_bigstep*:
"$(c,s) \Rightarrow (\mathit{False},\ t) \Longrightarrow \mathit{ccomp}\ c\ 0 \vdash (0,s,\mathit{stk}) \rightarrow\!*\ (\mathit{size}(\mathit{ccomp}\ c\ 0),t,\mathit{stk})$"