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

WS 2022/23
29.11.2022

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 *ccomp2* which reduces the number of instructions for programs of the form *IF b THEN c*. Try to finish *ccomp2* without looking up *ccomp*!
3. Extend the proof of *comp_bigstep* to your modified compiler.

```
value "ccomp (IF Less (V ''x'') (N 5) THEN ''y'' ::= N 3 ELSE SKIP)"

fun ccomp2 :: "com ⇒ instr list" where
  "ccomp2 SKIP = []" |
  "ccomp2 (x ::= a) = acomp a @ [STORE x]" |
  "ccomp2 (c₁;;c₂) = ccomp2 c₁ @ ccomp2 c₂" |
  "ccomp2 (WHILE b DO c) =
    (let cc = ccomp2 c; cb = bcomp b False (size cc + 1)
    in cb @ cc @ [JMP (−(size cb + size cc + 1))])"

value "ccomp2 (IF Less (V ''x'') (N 5) THEN ''y'' ::= N 3 ELSE SKIP)"

lemma ccomp_bigstep:
  "(c,s) ⇒ t ⟹ ccomp2 c ⊢ (0,s,stk) →* (size(ccomp2 c),t,stk)"
```

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*/*tbval* and *atyping*/*btyping* 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* ⇒ *real*).

Homework 6.1 Loop Compiler

Submission until Monday, December 5, 23:59pm.

For this exercise we have replaced the normal *WHILE* loop in IMP by a new *Loop c b* construct (without nice syntax). The modified type *com* and the big-step semantics are given in the definitions file. Your task is to define the compiler *ccomp* for the new loop construct and prove the correctness theorem *ccomp_bigstep*.

```
fun ccomp :: "com ⇒ instr list" where
"ccomp SKIP = []" |
"ccomp (x ::= a) = acomp a @ [STORE x]" |
"ccomp (c₁;;c₂) = ccomp c₁ @ ccomp c₂" |
"ccomp (IF b THEN c₁ ELSE c₂) =
  (let cc₁ = ccomp c₁; cc₂ = ccomp c₂; cb = bcomp b False (size cc₁ + 1)
   in cb @ cc₁ @ JMP (size cc₂) # cc₂)"
value "ccomp (Loop (''u'' ::= Plus (V ''u'') (N 1)) (Less (N 0) (V ''u'')))"

lemma ccomp_bigstep:
  "(c,s) ⇒ t ⟹ ccomp c ⊢ (0,s,stk) →* (size(ccomp c),t,stk)"
```

Homework 6.2 Pairs

Submission until Monday, December 5, 23:59pm.

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

```
datatype val = Iv int | Pv val val
```

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)" |
"taval a1 s (Iv i1) ⟹ taval a2 s (Iv i2)
 ⟹ taval (Plus a1 a2) s (Iv (i1 + i2))"

inductive_cases [elim!]:
  "taval (N i) s v"
  "taval (V x) s v"
  "taval (Plus a1 a2) s v"
```

Boolean expressions stay the same - pairs can not be compared.

```
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 a command to swap the first and second element of a pair:

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

Adapt the small-step semantics accordingly:

inductive
  small_step :: "(com × state) ⇒ (com × state) ⇒ bool" (**infix** "→" 55)
where
Assign: "taval a s v ⟹ (x ::= a, s) → (SKIP, s(x := v))" |
Seq1: "(SKIP;;c,s) → (c,s)" |
Seq2: "(c1,s) → (c1′,s′) ⟹ (c1;;c2,s) → (c1′;;c2,s′)" |
IfTrue: "tbval b s True ⟹ (IF b THEN c1 ELSE c2,s) → (c1,s)" |
IfFalse: "tbval b s False ⟹ (IF b THEN c1 ELSE c2,s) → (c2,s)" |
While: "(WHILE b DO c,s) → (IF b THEN c;; WHILE b DO c ELSE SKIP,s)"

lemmas small_step_induct = small_step.induct[split_format(complete)]

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

datatype *ty = Ity | Pty ty ty*

Complete the atyping and ctyping rules:

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

declare atyping.intros [intro!]
inductive_cases [elim!]:
  "Γ ⊢ V x : τ" "Γ ⊢ N i : τ" "Γ ⊢ Plus a1 a2 : τ"

inductive btyping :: "tyenv ⇒ bexp ⇒ bool" (**infix** "⊢" 50) **where**
B_ty: "Γ ⊢ Bc v" |
Not_ty: "Γ ⊢ b ⟹ Γ ⊢ Not b" |
And_ty: "Γ ⊢ b1 ⟹ Γ ⊢ b2 ⟹ Γ ⊢ And b1 b2" |
Less_ty: "Γ ⊢ a1 : Ity ⟹ Γ ⊢ a2 : Ity ⟹ Γ ⊢ Less a1 a2"

declare btyping.intros [intro!]
inductive_cases [elim!]: "Γ ⊢ Not b" "Γ ⊢ And b1 b2" "Γ ⊢ Less a1 a2"

inductive ctyping :: "tyenv ⇒ com ⇒ bool" (**infix** "⊢" 50) **where**
Skip_ty: "Γ ⊢ SKIP" |
Assign_ty: "Γ ⊢ a : Γ(x) ⟹ Γ ⊢ x ::= a" |
Seq_ty: "Γ ⊢ c1 ⟹ Γ ⊢ c2 ⟹ Γ ⊢ c1;;c2" |
If_ty: "Γ ⊢ b ⟹ Γ ⊢ c1 ⟹ Γ ⊢ c2 ⟹ Γ ⊢ IF b THEN c1 ELSE c2" |

While_ty: "$\Gamma \vdash b \Longrightarrow \Gamma \vdash c \Longrightarrow \Gamma \vdash$ *WHILE b DO c*"

declare *ctyping.intros* [*intro*!]

inductive_cases [*elim*!]:
"$\Gamma \vdash x ::= a$" "$\Gamma \vdash c1;;c2$"
"$\Gamma \vdash$ *IF b THEN c1 ELSE c2*"
"$\Gamma \vdash$ *WHILE b DO c*"

Complete the proofs of preservation and progress.

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

theorem *apreservation*:
"$\Gamma \vdash a : \tau \Longrightarrow taval\ a\ s\ v \Longrightarrow \Gamma \vdash s \Longrightarrow type\ v = \tau$"

theorem *aprogress*: "$\Gamma \vdash a : \tau \Longrightarrow \Gamma \vdash s \Longrightarrow \exists v.\ taval\ a\ s\ v$"

theorem *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* $\Rightarrow$ *com* $*$ *state* $\Rightarrow$ *bool*" (**infix** "$\rightarrow*$" *55*)
where "$x \rightarrow* y$ == *star small_step x y*"

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

theorem *type_sound*:
"$(c,s) \rightarrow* (c',s') \Longrightarrow \Gamma \vdash c \Longrightarrow \Gamma \vdash s \Longrightarrow c' \neq SKIP$
$\Longrightarrow \exists cs''.\ (c',s') \rightarrow cs''$"