CmSc336: Types
Homework 1 Solutions

Ido Rosen

18 January 2005

Administrativia

This section applies to all graded work and solution sets.

Any questions regarding specific grades or comments on homework should be sent via email to <ido@cs.uchicago.edu>. If you wish to make an appointment to discuss something or to correct me if I have made a mistake, please email me with at least a day's notice. Most late afternoons (3-5PM), I am in Ry254.

Homework should be handed in on time. If your homework is not complete by the beginning of class, don't do it during class–just email it to me as soon as possible. You should hand in late homework, even if it is too late to be graded, so that I at least have a chance to read your work and give you some feedback. (If you aren't getting a grade, you can still get some acknowledgement of your effort this way.)

Any programming assignments or homework with programming should be handed in electronically (via email). If you are feeling extra-sneaky, you may encrypt it to my PGP key or use my web interface: https://idorosen.com/encrypt.

If your solution is exemplary, it may be used as a sample solution. If your solution is particularly bad, it may be displayed for entertainment purposes. (Just kidding!)

Learning is more important than grades.

1 Big-step evaluation

4.2.2 Exercise [Recommended, ⋆⋆⋆ ↛]: Change the definition of the `eval` function in the `arith` implementation to the big-step style introduced in Exercise 3.5.17. □

Listing 1: Code courtesy of Dave MacQueen

```
structure Core =
struct

open Syntax

exception NoRuleApplies

fun isnumericval t =
    case t
      of TmZero _ => true
       | TmSucc (_,t1) => isnumericval t1
       | TmPred (_,t1) => isnumericval t1
       | _ => false

fun isval t =
    case t
      of TmTrue _ => true
       | TmFalse _ => true
       | t => isnumericval t

(* invariant: eval2 t returns a normal form (not necessarily a value),
 * or raises NoRuleApplies. *)

(* eval2:  big-step evaluation, with shallow runtime type checking *)
(* This produces some nonvalue results, e.g.
 *  eval2(TmSucc(TmTrue)) ==> TmSucc(TmTrue)
 *
 * One could add some deeper checks, such as checking that isnumericval
 * is true for result of eval2 t1 in the TmSucc case.
 *)
fun eval2 t =
    case t
      of TmZero _ => t
       | TmTrue _ => t
       | TmFalse _ => t
       | TmIf(_,t1,t2,t3) =>
          (case eval2 t1
             of TmTrue _ => eval2 t2
```

```
            | TmFalse _ => eval2 t3
            | _ => raise NoRuleApplies)
      | TmSucc(info,t1) => TmSucc(NONE, eval2 t1)
      | TmPred(info,t1) =>
        (case eval2 t1
           of TmZero _ => TmZero NONE
            | TmSucc(_,t1') => t1'
            | _ => raise NoRuleApplies)
      | TmIsZero(info,t1) =>
        (case eval2 t1
           of TmZero _ => TmTrue NONE
            | TmSucc _ => TmFalse NONE
            | _ => raise NoRuleApplies)
      | _ => raise NoRuleApplies

fun eval t =
    eval2 t
    handle NoRuleApplies => t

end (* structure Core *)
```

```
(* the following version that checks the evaluated argument of TmSucc
 * should always return a valid value if NoRuleApplies is not raised.
 * The name "NoRuleApplied" should probably be changed to "TypeError".
 *)
fun eval3 t =
    case t
      of TmZero _ => t
       | TmTrue _ => t
       | TmFalse _ => t
       | TmIf(_,t1,t2,t3) =>
          (case eval2 t1
             of TmTrue _ => eval2 t2
              | TmFalse _ => eval2 t3
              | _ => raise NoRuleApplies)
       | TmSucc(info,t1) =>
          let val t1' = eval2 t1
           in (* prevent nonvalue normal forms from being constructed *)
              if isnumericval t1
              then TmSucc(NONE, t1')
              else raise NoRuleApplies
          end
       | TmPred(_,t1) =>
          (case eval2 t1
             of TmZero _ => TmZero NONE
              | TmSucc(_,t1') => t1'
              | _ => raise NoRuleApplies)
       | TmIsZero(info,t1) =>
          (case eval2 t1
             of TmZero _ => TmTrue NONE
              | TmSucc _ => TmFalse NONE
              | _ => raise NoRuleApplies)
       | _ => raise NoRuleApplies
```

2 Your `wrong`

5.3.7 Exercise [⋆⋆ ↛]: Exercise 3.5.16 gave an alternative presentation of the operational semantics of booleans and arithmetic expressions in which stuck terms are defined to evaluate to a special constant `wrong`. Extend this semantics to λ**NB** □

Listing 2: Changes to the operational semantics in 3.5.16

t ::= ... | $\lambda x.t$
v ::= ... | $\lambda x.t$
badnat ::= ... | $\lambda x.t$
badλ ::= wrong | nv | true | false
badλ t $\rightarrow$ wrong
$(\lambda x.t)$v $\rightarrow [x \rightarrow v]t$

3 Shifting

6.2.3 Exercise [⋆⋆ ↛]: Show that if $\mathtt{t}$ is an n-term, then $\uparrow {}^{d}_{c}(\mathtt{t})$ is an $(n+d)$-term. □

Proof by structural induction on $\mathtt{t}$. Cases taken from (6.2.1). Assume $c < n$.

Hypothesis $\mathtt{t}$ is an n-term $(\mathtt{t} \in \tau_n) \Longrightarrow \uparrow {}^{d}_{c}\mathtt{t} \in \tau_{n+d}$

Base Case $\mathtt{t} = k$, where $0 \leq k < n$

$$\uparrow {}^{d}_{c} k = \begin{cases} k \text{ if } k < c \\ k + d \text{ if } k \geq c \end{cases}$$

Suppose $k < c$, then $\uparrow {}^{d}_{c} k = k$. $k < n \leq n + d$.
$\therefore k \in \tau_{n+d}$

First Inductive Case $\mathtt{t} = \lambda.t_1$
$\mathtt{t} \in \tau_n \Rightarrow \lambda.t_1 \in \tau_n \Rightarrow t_1 \in \tau_{n+1}$ (inversion of the definition of τ_n)
Induction Hypothesis: $\uparrow {}^{d}_{c} t_1 \in \tau_{(n+1)+d} (= \tau_{(n+d)+1})$
By definition of τ_n, $\lambda(\uparrow {}^{d}_{c} t_1) \in \tau_{n+d}$

Second Inductive Case $\mathtt{t} = t_1\, t_2$
$\mathtt{t} \in \tau_n \Rightarrow t_1 \in \tau_n, t_2 \in \tau_n$ (inversion of the definition of τ_n)
Induction Hypothesis: $\uparrow {}^{d}_{c} t_1 \in \tau_{n+d}$ and $\uparrow {}^{d}_{c} t_2 \in \tau_{n+d}$
By definition of τ_n, $((\uparrow {}^{d}_{c} t_1)(\uparrow {}^{d}_{c} t_2)) \in \tau_{n+d}$
By definition of $\uparrow {}^{d}_{c}$: $((\uparrow {}^{d}_{c} t_1)(\uparrow {}^{d}_{c} t_2)) = \uparrow {}^{d}_{c}(t_1\, t_2) = \uparrow {}^{d}_{c}\mathtt{t}$
$\therefore \uparrow {}^{d}_{c}\mathtt{t} \in \tau_{n+d}$

Inversion Lemma

$$\lambda t \in \tau_n \quad \Rightarrow \quad t \in \tau_{n+1} \tag{1}$$
$$(t_1 t_2) \in \tau_n \quad \Rightarrow \quad t_1 \in \tau_n, t_2 \in \tau_n \tag{2}$$

The lemma can be proved by inspection for our purposes.

4 Not to be negative, but...

6.3.1 Exercise [$\star$]: Should we be worried that the negative shift in this rule might create ill-formed terms containing negative indices? □

No.
Variables shifted down have previously been shifted up.
Zero indices have been eliminated by substitution.