CS 320: MidTerm

Total: 100 pts

CS 320 Course Staff

Name: Nathan Mull

BU ID: 12345678

1 Good/Bad Programming Languages [50 pts]

In this section, we will introduce a small language called MidLang (Midterm Language), using its syntax, type system, and semantics. The rules of the language are exactly like (a subset of) what we have in OCaml.

Syntax

$$\langle expr \rangle ::= \mathsf{true} \mid \mathsf{false} \mid n \mid \mathsf{if}\ e\ \mathsf{then}\ \langle expr \rangle\ \mathsf{else}\ \langle expr \rangle \mid \mathsf{let}\ x = \langle expr \rangle\ \mathsf{in}\ \langle expr \rangle \mid v$$

In this syntax, true and false represent the standard boolean values, and n stands for integer constants (e.g., 0, 1, -1, etc.). The if and let expressions are standard and v is a variable.

Type System

There are two types in MidLang: bool and int. The typing rules are defined below:

$$\frac{}{\Gamma \vdash \mathsf{true} : \mathsf{bool}}\ \textsc{TrueLit} \qquad \frac{}{\Gamma \vdash \mathsf{false} : \mathsf{bool}}\ \textsc{FalseLit} \qquad \frac{(n \text{ is an integer literal})}{\Gamma \vdash n : \mathsf{int}}\ \textsc{IntLit}$$

$$\frac{\Gamma \vdash e : \mathsf{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau}\ \textsc{If} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau}\ \textsc{Let} \qquad \frac{v : \tau \in \Gamma}{\Gamma \vdash v : \tau}\ \textsc{Var}$$

The rules are standard. The Var rule states that variable v has type τ in the set Γ.

Semantics

$$\frac{}{\mathsf{true} \Downarrow \mathsf{true}}\ \textsc{TrueEval} \qquad \frac{}{\mathsf{false} \Downarrow \mathsf{false}}\ \textsc{FalseEval} \qquad \frac{}{n \Downarrow n}\ \textsc{IntEval}$$

$$\frac{e \Downarrow \mathsf{true} \qquad e_1 \Downarrow v_1}{\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \Downarrow v_1}\ \textsc{If-True} \qquad \frac{e \Downarrow \mathsf{false} \qquad e_2 \Downarrow v_2}{\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \Downarrow v_2}\ \textsc{If-False} \qquad \frac{e_1 \Downarrow v_1 \qquad [v_1/x]e_2 \Downarrow v_2}{\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \Downarrow v_2}\ \textsc{LetEval}$$

An important property of a programming language is called **preservation**: if an expression has type τ, then it must evaluate to a value of the same type τ. Formally, if $\cdot \vdash e : \tau$ and $e \Downarrow v$, then $\cdot \vdash v : \tau$. For your benefit, we are telling you that **MidLang satisfies this property**.

For the following problems, we will introduce variants of MidLang, namely MidLang1, MidLang2, and MidLang3, that change some of the rules of either the type system or semantics or both. You task is to figure out if MidLangN still satisfies this preservation property after these changes.

If you believe the preservation property is violated, give a counterexample expression e such that $\cdot \vdash e : \tau_1$ and $e \Downarrow v$ and $\cdot \vdash v : \tau_2$ and $\tau_1 \neq \tau_2$. **In this case, give a**

(i) typing derivation of $\cdot \vdash e : \tau_1$,

(ii) semantic derivation of $e \Downarrow v$, and

(iii) typing derivation of $\cdot \vdash v : \tau_2$

If you believe the preservation property is still satisfied, give a **detailed intuitive explanation of why**.

Note: These problems don't overlap. For each section, you only need to change the rules that are provided in that section, not the rules from the previous sections.

1.1 MidLang1

Suppose, for the if e then e_1 else e_2 expression, we allowed expressions e_1 and e_2 to have different types τ_1 and τ_2 respectively. And suppose the typing rule is defined as follows:

$$\frac{\Gamma \vdash e : \mathsf{bool} \qquad \Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau_1}\ \textsc{If}$$

The type of the if expression is the same as the type of then-branch expression e_1. The semantics rules of the if expression remain exactly the same.

Problem 1 (20 pts) *Is it possible for an expression e in MidLang1 to have a different type than its value v? If yes, give an example of such an expression along with (i) typing derivation of $\cdot \vdash e : \tau_1$, (ii) semantic derivation of $e \Downarrow v$, and (iii) typing derivation of $\cdot \vdash v : \tau_2$. If not, give an intuitive explanation why this is not possible.*

Solution.

YES : if false then 2 else false

(i)
$$\frac{\dfrac{}{\emptyset \vdash \text{false} : \text{bool}}\,(\text{false}) \quad \dfrac{}{\emptyset \vdash 2 : \text{int}}\,(\text{intLit}) \quad \dfrac{}{\emptyset \vdash \text{false} : \text{bool}}\,(\text{false})}{\emptyset \vdash \text{if false then 2 else false} : \text{int}}\,(\text{if})$$

(ii)
$$\frac{\dfrac{}{\text{false} \Downarrow \text{false}}\,(\text{falseEval}) \qquad \dfrac{}{\text{false} \Downarrow \text{false}}\,(\text{falseEval})}{\text{if false then 2 else false} \Downarrow \text{false}}\,(\text{ifEval})$$

(iii)
$$\frac{}{\emptyset \vdash \text{false} : \text{bool}}\,(\text{false})$$

int ≠ bool

Solution.

1.2 MidLang2

Suppose we allow variables to be used at an arbitrary type. The typing rule can be defined as follows:

$$\frac{v : \tau_1 \in \Gamma}{\Gamma \vdash v : \tau_2}\ \textsc{Var}$$

This rule means that if v has type τ_1 in context Γ, we can use that to infer that $v : \tau_2$.

Problem 2 (20 pts) *Is it possible for an expression e in MidLang2 to have a different type than its value v? If yes, give an example of such an expression along with (i) typing derivation of $\cdot \vdash e : \tau_1$, (ii) semantic derivation of $e \Downarrow v$, and (iii) typing derivation of $\cdot \vdash v : \tau_2$. If not, give an intuitive explanation why this is not possible.*

Solution.

YES : let x = 2 in x

(i)
$$\frac{\dfrac{}{\emptyset \vdash 2 : \text{int}}\,(\text{intLit}) \qquad \dfrac{}{x : \text{int} \vdash x : \text{bool}}\,(\text{var})}{\emptyset \vdash \text{let } x = 2 \text{ in } x : \text{bool}}\,(\text{let})$$

(ii)
$$\frac{\dfrac{}{2 \Downarrow 2}\,(\text{iLit}) \qquad \dfrac{}{2 \Downarrow 2}\,(\text{iLit})}{\text{let } x = 2 \text{ in } x \Downarrow 2}\,(\text{letEval})$$

(iii)
$$\frac{}{\emptyset \vdash 2 : \text{int}}\,(\text{intLit})$$

bool ≠ int

Solution.

1.3 MidLang3

Suppose we change the type of true and false to be an integer. And we change the type of $\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$ expression so that e must be an integer. The typing rules can be defined as follows:

$$\frac{}{\Gamma \vdash \mathsf{true} : \mathsf{int}}\ \textsc{TrueLit} \qquad \frac{}{\Gamma \vdash \mathsf{false} : \mathsf{int}}\ \textsc{FalseLit} \qquad \frac{\Gamma \vdash e : \mathsf{int} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau}\ \textsc{If}$$

Problem 3 (10 pts) *Is it possible for an expression e in MidLang3 to have a different type than its value v? If yes, give an example of such an expression along with (i) typing derivation of $\cdot \vdash e : \tau_1$, (ii) semantic derivation of $e \Downarrow v$, and (iii) typing derivation of $\cdot \vdash v : \tau_2$. If not, give an intuitive explanation why this is not possible.*

Solution.

NO: In this system there is only one type

Solution.

2 OCaml Programming [20 pts]

Implement the following function. You are also welcome to define helper functions. You should not use any library function for this problem. If you'd like to use a library function, please define the function first.

Problem 4 (20 pts) *Define a function called* `sublists` *that takes a list ℓ as input and generates all possible sublists of ℓ as output:*

val `sublists` : 'a list → 'a list list

A sublist of a list ℓ is a list containing a subset of the elements in ℓ, in the same order as they appear in ℓ, e.g., $[1;2]$, $[2;3]$, $[]$, and $[1;3]$ are sublists of the list $[1;2;3]$. You can assume that the elements of the input list are distinct.

Solution.

```
let sublists l =
  let rec go l =
    match l with
    | [] -> [[]]
    | x::xs ->
      let subs = go xs in
      subs @ map (fun l -> x::l) subs
  in go l
```

```
let rec map f l =
  match l with
  | [] -> []
  | x::xs -> f x :: map f xs
```

```
let (@) l r =
  match l with
  | [] -> r
  | x::xs -> x :: xs@r
```

Solution.

3 Types and their Values [10 pts]

It is common to think of types as a collection of its values. For example, the unit type has one value (). The bool type has two values: true and false. The int type has infinitely many values represented by the set of integers: $\{0, 1, -1, 2, -2, \ldots\}$.

Problem 5 (5 pts) *Give an example of a type that has exactly 3 values. Write down the corresponding values as well. You are welcome to define your own type.*

Solution.

```
type foo =
  | One
  | Two
  | Three
```

Problem 6 (5 pts) *Using the type from the previous example, give an example of a type that has exactly 9 values.*

Solution.

```
type bar = foo * foo
```

Extra Page

4 Inference Rules [20 pts]

In this problem we will consider a set of inference rules for nonsense judgments. We will take an *expression* to be a nonempty sequence of the words 'foo', 'bar', 'cat', and 'dog', e.g.

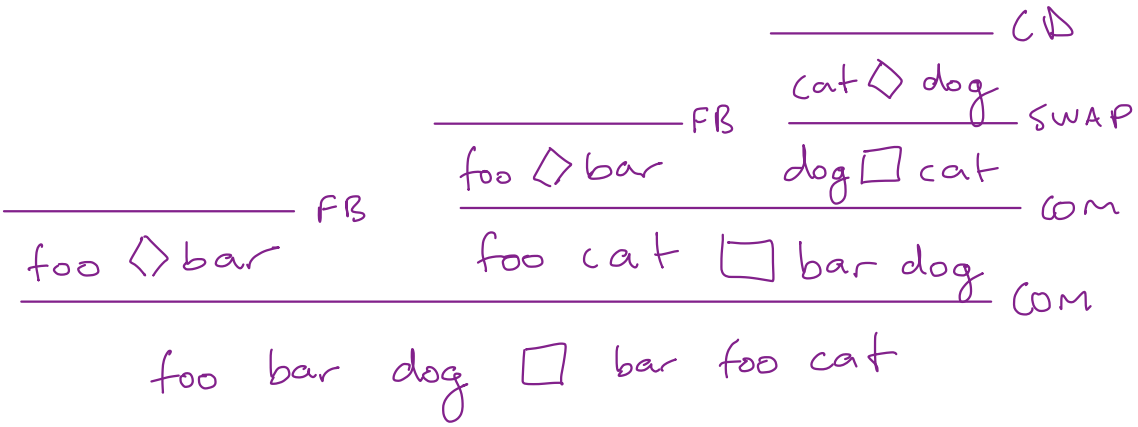
- foo
- foo bar foo
- foo cat foo bar foo
- dog dog cat

There are two kinds of judgments: $e_1 \Diamond e_2$ and $e_1 \Box e_2$, where e_1 and e_2 stand for expressions, and there are 4 inference rules:

$$\frac{}{\mathsf{foo} \Diamond \mathsf{bar}}\ \text{FB} \qquad \frac{}{\mathsf{cat} \Diamond \mathsf{dog}}\ \text{CD} \qquad \frac{e_1 \Diamond e_2}{e_2 \Box e_1}\ \text{SWAP} \qquad \frac{e_1 \Diamond e_2 \qquad e_3 \Box e_4}{e_1\ e_4 \Box e_2\ e_3}\ \text{COM}$$

Problem 7 (20 pts) *Use the inference rules above to derive the judgment:* foo bar dog □ bar foo cat

Solution.



Solution.