1 OCaml Basics

- Suppose we have a type \texttt{intlist} defined as:

  \[
  \text{type intlist} = \text{Nil} | \text{Cons of int * intlist}
  \]

  Write a function \texttt{max : intlist \rightarrow int} that returns the largest integer in an \texttt{intlist}. If the argument to \texttt{max} is an empty list, it should return 0.

2 BNF Grammars and ASTs

Consider the following BNF grammar:

\[
A ::= \langle \text{ident} \rangle | \text{true} | \text{false} | \neg A
\]

\[
P ::= A | P \land P | P \lor P
\]

- Draw the abstract syntax tree in this grammar for the term \((p \land (q \lor \text{false})) \lor \text{true}\).
• Write OCaml datatypes representing the ASTs for $A$ and $P$.

• Write the OCaml value corresponding to the AST for the term $(p \land (q \lor \text{false})) \lor \text{true}$.

3 Type Systems

The type system for a simple imperative language is given in Appendix A.

• Given a type context $\Gamma$ such that $\Gamma(x) = \text{int}$, is the term

$$x := \text{if } 3 = 3 \text{ then } 4 \text{ else false}$$

type-correct? Why or why not?

• Write the proof tree for the typing judgment described above. If it is not type-correct, indicate the place in the proof tree where a rule fails to apply or is impossible to complete.
Suppose we were writing a type-checking function \( \text{typecheck} : \text{context} \rightarrow \text{cmd} \rightarrow \text{bool} \) that takes a type context and an AST for a command and returns true if it is type-correct. Fill in the skeleton below by translating the typing rule for the while command into OCaml code.

\[
\text{let rec typecheck } (\gamma : \text{context}) (c : \text{cmd}) : \text{bool} = \\
\quad \text{match } c \text{ with} \\
\quad \quad | \text{While } (e, c) \rightarrow \\
\]

4 Operational Semantics

The operational semantics for a simple imperative language is given in Appendix B.

- Write the next configuration that

\[
(a := \text{if } x = 2 \text{ then } y \text{ else } x; b := (x \ast y) = 21, \{x = 3, y = 7\})
\]

steps to.

- Write the proof true for the step above.
• Suppose we wanted to extend our simple imperative language with a command “c unless e” that executes c if e is false, and does nothing otherwise. Give small-step semantics rules for “c unless e”.

• Values in our simple imperative language are either integers or booleans. Write a type value that represents values.

• Suppose we were writing an interpreter function eval_exp : exp -> state -> value that evaluates an expression to a value (assuming that there are no errors). Fill in the code for evaluating an if-then-else expression.

```ocaml
let rec eval_exp (e : exp) (s : state) : value =
  match e with
  | If (e, e1, e2) ->
```

• Suppose we were writing a step function step_cmd : cmd -> state -> (cmd * state) returns the next command-state pair that a command steps to (assuming there are no errors). Fill in the code for stepping the while command.

```ocaml
let rec step_cmd (c : cmd) (s : state) : (cmd * state) =
  match c with
  | While (e, c) ->
```
A Typing Rules for a Simple Imperative Language

\[
\frac{(n \text{ is a number})}{\Gamma \vdash n : \text{int}}
\]
\[
\frac{(b \text{ is a boolean})}{\Gamma \vdash b : \text{bool}}
\]
\[
\frac{(\Gamma(x) = \tau)}{\Gamma \vdash x : \tau}
\]
\[
\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 \oplus e_2 : \text{int}} \quad \text{where } \oplus \text{ is an arithmetic operator}
\]
\[
\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \otimes e_2 : \text{bool}} \quad \text{where } \otimes \text{ is a boolean operator}
\]
\[
\frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 = e_2 : \tau}
\]
\[
\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash c_1 : \text{ok} \quad \Gamma \vdash c_2 : \text{ok}}{\Gamma \vdash \text{if } e \text{ then } c_1 \text{ else } c_2 : \text{ok}}
\]
\[
\frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e_1 \text{ then } e_1 \text{ else } e_2 : \tau}
\]
\[
\frac{\Gamma \vdash e_1 : \tau}{\Gamma \vdash x := e : \text{ok}}
\]
\[
\frac{\Gamma \vdash e : \text{bool} \quad \Gamma \vdash c_1 : \text{ok} \quad \Gamma \vdash c_2 : \text{ok}}{\Gamma \vdash \text{while } e \text{ do } c : \text{ok}}
\]

B Operational Semantics for a Simple Imperative Language

\[
\frac{(n \text{ is a number})}{(n, \sigma) \Downarrow n}
\]
\[
\frac{(b \text{ is a boolean})}{(b, \sigma) \Downarrow b}
\]
\[
\frac{(\sigma(x) = v)}{(x, \sigma) \Downarrow v}
\]
\[
\frac{(e_1, \sigma) \Downarrow v_1 \quad (e_2, \sigma) \Downarrow v_2 \quad (v_1 \oplus v_2 = v)}{(e_1 \oplus e_2, \sigma) \Downarrow v} \quad \text{where } \oplus \text{ is an arithmetic or boolean operator}
\]
\[
\frac{(e, \sigma) \Downarrow \text{true}}{(e_1, \sigma) \Downarrow v}
\]
\[
\frac{(e, \sigma) \Downarrow \text{false}}{(e_2, \sigma) \Downarrow v}
\]
\[
\frac{(e, \sigma) \Downarrow v}{(x := e, \sigma) \rightarrow (\text{skip}, \sigma[x \mapsto v])}
\]
\[
\frac{(c_1, \sigma) \rightarrow (c_1', \sigma')}{(c_1; c_2, \sigma) \rightarrow (c_1'; c_2, \sigma')}
\]
\[
\frac{(c_1, \sigma) \rightarrow (c_1', \sigma')}{(\text{skip}; c_2, \sigma) \rightarrow (c_2, \sigma)}
\]
\[
\frac{(e, \sigma) \Downarrow \text{true}}{(if \ e \ then \ c_1 \ else \ c_2, \sigma) \rightarrow (c_1, \sigma)}
\]
\[
\frac{(e, \sigma) \Downarrow \text{false}}{(if \ e \ then \ c_1 \ else \ c_2, \sigma) \rightarrow (c_2, \sigma)}
\]
\[
\frac{(\text{while } e \ do \ c, \sigma) \rightarrow (if \ e \ then \ (c; \text{while } e \ do \ c) \ then \ \text{skip}, \sigma)}{(if \ e \ then \ (c; \text{while } e \ do \ c) \ else \ \text{skip}, \sigma)}
\]