

CS 320: Mock Mid-Term

Total: 100 pts

CS 320 Course Staff

1 Weird Programming Languages [50 pts]

In this section, we will design slightly different programming languages than (the subset of) OCaml that we know and love. As usual, this would mean defining the formal aspects of these languages: syntax, type system, and semantics. For your benefit, we will be defining some parts of the formalism and relying on you to fill in the missing details.

The syntax of the language looks exactly like we have in OCaml. We won't write all the expressions here, just the ones relevant for the following problems.

$$\langle expr \rangle ::= \text{true} \mid \text{false} \mid \text{if } \langle expr \rangle \text{ then } \langle expr \rangle \text{ else } \langle expr \rangle \mid n \mid \langle expr \rangle + \langle expr \rangle \mid \dots$$

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 addition expressions are also standard.

1.1 Integers Inside If Expressions

For the expression `if e then e_1 else e_2` , we usually assume that e has type `bool`. Hence, the typing rule is written as

$$\frac{\Gamma \vdash e : \text{bool} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau} \text{IF}$$

and the semantics rules are written as

$$\frac{e \Downarrow \text{true} \quad e_1 \Downarrow v_1}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_1} \text{IF-THEN} \qquad \frac{e \Downarrow \text{true} \quad e_2 \Downarrow v_2}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_2} \text{IF-ELSE}$$

Suppose, instead, we allowed expression e to be an integer also with the following behavior: If e is positive (i.e., $e \geq 0$), then the expression goes into evaluating the `then` branch. Otherwise, if e is negative, then the expression goes into evaluating the `else` branch.

Problem 1 (5 pts) Write the corresponding typing rule(s) for this new `if` expression.

Problem 2 (5 pts) Write the corresponding semantics rule(s) for this new `if` expression.

Problem 3 (10 pts) Consider the following expression:

if 1 + 2 then 3 else 4

Is this expression valid (i.e., well-typed)? If yes, please show a valid typing derivation. Otherwise, give an intuitive explanation of why it's not well-typed.

Problem 4 (10 pts) If it's valid, what is the value of the expression above from Problem 3? Show a semantic derivation.

1.2 Adding Integers and Floats

Suppose we allowed adding integers and floats in our language using the $+$ operator such that if either of the summands is a float, then the result is a float.

Problem 5 (10 pts) Write all possible typing rule(s) for this new addition expression.

Problem 6 (10 pts) With the if expression updated according to Section 1.1, consider the following expression:

if $1 + 2.0$ then 3 else 4

Is this expression valid (i.e., well-typed)? If yes, please show a valid typing derivation. Otherwise, give an intuitive explanation of why it's not well-typed.

2 Polymorphic Higher-Order Programming [20 pts]

Problem 7 (20 pts) Below, we will provide a list of function types. Either define a function of this type, or briefly explain why it is impossible to do so. You are welcome to use the standard library functions that we have discussed in the course.

1. $\text{int} \rightarrow 'a \text{ list}$
2. $'a \text{ list} \rightarrow 'b \text{ list}$
3. $('b \rightarrow 'a) \rightarrow 'a \text{ list} \rightarrow 'b \text{ list}$
4. $'a \rightarrow 'a \text{ list}$
5. $'a \text{ list} \rightarrow 'a$

3 Good Old OCaml Programming [30 pts]

No midterm is complete without some OCaml programming. Implement the following functions: again you are welcome to use standard library functions unless explicitly stated. You are also welcome to define helper functions.

Problem 8 (10 pts) Define a function called `merge_sort` that sorts an input list

`val merge_sort : 'a list → 'a list`

using the standard merge sort algorithm. You cannot (obviously) use the `List.sort` function.

Problem 9 (10 pts) Define a function called `pack` which packs together consecutive elements of a list that are equal into a sublist.

`val pack : 'a list → 'a list list`

For instance, with the input `[1; 1; 2; 3; 3; 2; 4; 4; 4]`, the output should be `[[1; 1]; [2]; [3; 3]; [2]; [4; 4; 4]]`. Note that the second occurrence of 2 is not packed with the first occurrence of 2 since only **consecutive** elements that are equal are packed together.

Problem 10 (10 pts) Define a function called `factorize` which computes the prime factors of the input and produces a list of tuples containing the prime factor and the corresponding exponent.

`val factorize : int → int * int list`

For input n , if the number n can be written as $p_1^{e_1} \cdot p_2^{e_2} \cdots p_n^{e_n}$, then the output should be the list $[(p_1, e_1); (p_2, e_2), \dots, (p_n, e_n)]$. Example, `factorize 360 = [(2, 3); (3, 2); (5, 1)]`. The prime numbers in the list must be in increasing order. For input 1, just return the empty list. Your implementation only needs to work for positive numbers.