

Example Solutions for Programming Languages G22.2110 Summer 2007 hw09

Assigned Th 7/19/2007, due We 7/25/2007 at 1pm.

These are example solutions. Please keep in mind that often, there is not just one correct solution to a question. If you come up with different answers to the homework, then it may be that both your answers and these answers here are correct. Of course, these answers here may also contain mistakes; if you spot some, please let us know so we can correct them.

Reading Assignments

- For lecture on 7/19/2007: Scott 7.2.4 (on CD);
Cumming (<http://www.dcs.napier.ac.uk/course-notes/sml/manual.html>)
 - For lecture on 7/26/2007: Scott 12.3.4-12.3.5, 12.4.2-12.4.3
-

Homework Assignments

1. SML (6 = 3 + 3 points)

While you write the SML code for answering Question 2 below, you will probably get some error messages. Describe two error messages using the following format:

- Code: *a very short piece of code that triggers the error*
- Symptom: *the error message itself*
- Cause: *an explanation for what triggered the error message*
- Solution: *how to fix the code to prevent the error*

Example solutions

```
- first error
- Code: let val s = TextIO.inputLine TextIO.stdIn in print s end;
- Symptom: operator and operand don't agree [tycon mismatch]
      operator domain: string
      operand:         string option
      in expression:   print s
- Cause: The result of TextIO.inputLine TextIO.stdIn is of type
string option, which can be either something like
      SOME "hello"
or, when the I/O was not successful, the error value
      NONE
But print expects just a string, not a string option.
```

```

- Solution: turn the string option into a pure string with valOf
  let val s = valOf (TextIO.inputLine TextIO.stdIn) in print s end;
- second error
- Code: valOf TextIO.inputLine TextIO.stdIn;
- Symptom: operator and operand don't agree [tycon mismatch]
  operator domain: 'Z option
  operand:         TextIO.instream -> string option
  in expression:   valOf TextIO.inputLine
- Cause: function calls in SML are left-associative, so the code
  is equivalent to
  (valOf TextIO.inputLine) TextIO.stdIn;
  where the operator is valOf, which expects a 'Z option, and
  the operand is TextIO.inputLine, which is a function from
  TextIO.instream to string option.
- Solution: put in parentheses to group the expression to the right
  valOf (TextIO.inputLine TextIO.stdIn);

```

2. SML (20 = 5 + 5 + 5 + 5 points)

Write SML programs exercising the fundamental features.

2a. I/O (5 points)

Write a program that prompts the user for his or her name, reads the name from input, then politely greets the user by name. Here is an example interactive session:

```

What is your name?
Bob
Hello, Bob, nice to meet you!

```

Example solutions

```

print "What is your name?\n";
let val input = valOf (TextIO.inputLine TextIO.stdIn)
    val trimmed = substring (input, 0, size input - 1)
in
  print ("Hello, " ^ trimmed ^ ", nice to meet you!\n")
end;

```

2b. Libraries (5 points)

Write a program that uses SML basis library functions to compute $\sqrt{2}$, $\sin(3.5)$, and $e^{2.5}$, and then prints the results like this (don't worry if the numbers are displayed in a slightly different format):

```

square root of 2.0:    1.41421356237
sine of 3.5:          ~0.35078322769
e to the power of 2.5: 12.1824939607

```

Example solutions

```

let val sqrt2 = Math.sqrt(2.0)
    val sin35 = Math.sin(3.5)
    val exp25 = Math.exp(2.5)
in
  print ("square root of 2.0:    " ^ Real.toString(sqrt2) ^ "\n");
  print ("sine of 3.5:          " ^ Real.toString(sin35) ^ "\n");
  print ("e to the power of 2.5: " ^ Real.toString(exp25) ^ "\n")
end;

```

2c. Types (5 points)

The following code creates a variable `c` with the character value 'Z', and then prints a description and the value of the variable:

```
let val c = #"Z"
in print ("name c, type char, value " ^ (Char.toString c) ^ "\n") end;
```

Extend this program by creating and printing more variables of different types. Your program should produce the following output:

```
name b, type bool, value true
name c, type char, value Z
name i, type int, value 42
name r, type real, value 3.141
name s, type string, value hello
```

Example solutions

```
let val b = true
in print ("name b, type bool, value " ^ (Bool.toString b) ^ "\n") end;
let val c = #"Z"
in print ("name c, type char, value " ^ (Char.toString c) ^ "\n") end;
let val i = 42
in print ("name i, type int, value " ^ (Int.toString i) ^ "\n") end;
let val r = 3.141
in print ("name r, type real, value " ^ (Real.toString r) ^ "\n") end;
let val s = "hello"
in print ("name s, type string, value " ^ s ^ "\n") end;
```

2d. Control flow (5 points)

Write an SML function `countOccurrences` that takes a pair of parameters, a string and a character, and returns the number of occurrences of the character in the string. For example, `countOccurrences ("hello", #"l")` should return 2. Do not use a loop, use recursion. Hint: you can use `String.explode` and write a helper function that counts occurrences of a character in a list of characters.

Example solutions

```
fun countOccurrencesList ([], _) = 0
| countOccurrencesList (h::t, c) =
    (if h = c then 1 else 0) + countOccurrencesList (t, c);
fun countOccurrences (str, chr) =
    countOccurrencesList (String.explode str, chr);
```

3. Type inference (12 = 3 + 1 + 8 points)

Consider the following SML function:

```
fun a (_, []) = [] | a (n, h::t) = (n+h)::a(n, t);
```

Assume that “+” operates on integers only.

3a. (3 points) Describe in words what `a` does.

Example solutions

Function `a` adds an integer to every element of a list, and returns the resulting list.

3b. (1 points) What is the type of a?

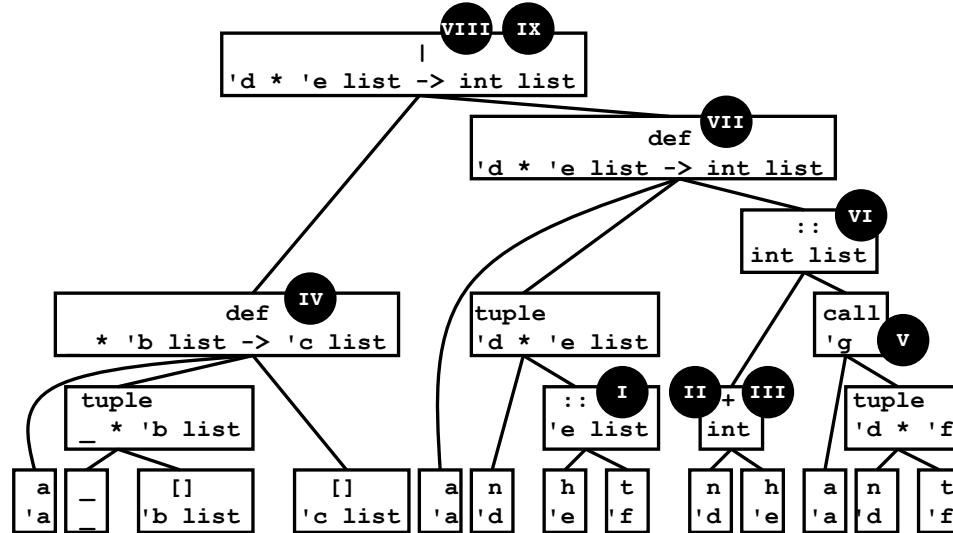
Example solutions

a : int * int list -> int list

3c. (8 points) Show the type inference steps for a.

Example solutions

Drawing the definition as a tree, assigning types to leaves, propagating types up to internal nodes:



```
fun a (_, [] ) = [] | a (n, h::t) = (n + h)::a (n, t);
```

Constraints:

- I 'f = 'e list
- II 'd = int
- III 'e = int
- IV 'a = _ * 'b list -> 'c list
- V 'a = 'd * 'f -> 'g
- VI 'g = int list
- VII 'a = 'd * 'e list -> int list
- VIII 'b = 'e
- IX 'c = int

Unification: the type of a is 'a, which is

- 'a = 'd * 'e list -> int list // constraint VII
- = int * 'e list -> int list // constraint II
- = int * int list -> int list // constraint III

Result:

a : int * int list -> int list

4. Type inference (12 = 3 + 1 + 8 points)

Consider the following SML function:

```
fun addOrMul (oper, x, y) = (if "add" = oper then op + else op * ) (x, y);
```

Assume that the type of "op +" is "int * int -> int".

4a. (3 points) Describe in words what addOrMul does.

Example solutions

If the first argument is the string "add",
 addOrMul returns the sum of the second and third argument,
 otherwise,
 it returns the product of the second and third argument.

4b. (1 points) What is the type of addOrMul?

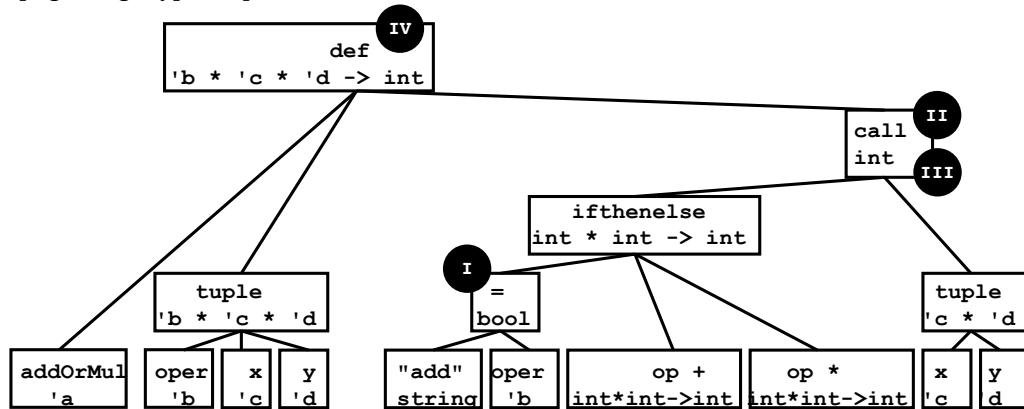
Example solutions

string * int * int -> int

4c. (8 points) Show the type inference steps for addOrMul.

Example solutions

Drawing the definition as a tree, assigning types to leaves,
 propagating types up to internal nodes:



```
fun addOrMul (oper, x, y) = (if "add" = oper then op + else op * ) (x, y);
```

Constraints:

- I 'b = string
- II 'c = int
- III 'd = int
- IV 'a = 'b * 'c * 'd -> int

Unification: the type of addOrMul is 'a, which is

- 'a = 'b * 'c * 'd -> int // constraint IV
- = string * 'c * 'd -> int // constraint I
- = string * int * 'd -> int // constraint II
- = string * int * int -> int // constraint III

Result:

a : string * int * int -> int