15-150 Assigment 1

Jonathan Li
jlli
S
May 19, 2016

Task 2.1

The pinned picture on the Piazza post is of a long-eared rabbit with what seems like deer antlers on the top of its head.

Task 2.2

1. Jon and Yoni had a discussion about lecture content, which is permitted by the collaboration policy. Since their discussion was not about homework, the whiteboard policy doesn't need to be applied.
2. Jon and Michael took notes from their discussion about homework, but they discarded them after the end of their discussion. This is akin to a discussion at a whiteboard, where neither of them would have been able to take home the content written on the whiteboard, so this discussion is permitted by the course collaboration policy, provided they each wrote the homework four hours after the discussion and they cited each other in their respective writeups.
3. According to course policy, students are not allowed to search the internet, libraries, or other external sources for homework solutions, so Brandon's actions in looking up a functional programming textbook for help on homework are not permitted.
4. In this situation, even if Michael cited Brandon as a collaborator on his homework writeup, since their 'collaboration' could not have taken place at a whiteboard (Michael just read Brandon's notes), both are in violation of course policy. Michael should not have read Brandon's work, and Brandon should have taken care to safeguard his work.
5. According to course policy, homeworks must not be derived in part or whole from work of others. By looking at the staff solution from a previous semester, Jon is deriving part of his work from the work of others, and so his actions are not permitted by course policy.

Task 3.1

In the expression `(intToString 7) ^ (intToString 9)`, the left hand side of the expression `(intToString 7)` evaluates to the string '7', which has type string. The right hand side of the expression `(intToString 9)` evaluates to the string '9', which also has type string. Since the left and right hand sides of the expression match the input types of the infix operator `^`, the whole expression is well-typed and has the type of the infix operator's return type, string. More formally:

(i) `intToString : int -> string`

(ii) `7 : int`

(iii) `(intToString 7) : string` by (APP)

(iv) `9 : int`

(v) `(intToString 9) : string` by (APP)

(vi) `^ : string * string -> string`

(vii) `(intToString 7) ^ (intToString 9) : string` by (APP)

Task 3.2

The expression `intToString 2.0` is not well-typed because intToString is a function with type `int -> string`, and 2.0 has type `real`. The argument taken by intToString must have type `int`, so since 2.0 does not have the right type, the expression is not well-typed.

Task 4.1

Since the fucntions argument is an expression, we must perform call-by-value evaluation before we substitute the argument into the function definition. The expression `fact 4` evaluates to `24`, so the whole expression steps to `intToString 24`. `intToString` is a function of type `int -> string`, so the expression will evaluate to the string `‘24’`.

Task 4.2

(i) `fact 4` $\Longrightarrow$ 24

(ii) `intToString (fact 4)` $\Longrightarrow$ `intToString 24`

(iii) `intToString 24` $\Longrightarrow$ `‘24’`

Task 5.1

When the unmodified `hw01.sml` file is run, a syntax error is raised.The error is caused by a lack of a bar before the second case expression `zerop _ = false` in line 8. This error can be fixed by inserting a bar `|` before the expression, like so: `|zerop _ false`.

Task 5.2

The first error message raised is a type mismatch between the operator `fact` and the operand 5.0 in line 16. `fact REQUIRES` an integer operand, but the operand provided, 5.0, is of type real. This can be fixed by replacing 5.0 with an input of type int, probably 5, like so: `val result = fact 5`.

Task 5.3

Now the first error message raised is a that in line 21, the compiler can't find function arguments in the function declaration for the function `semi`. Specifically, the line reads `fun semi : real = pi * r`. This expression can be fixed by introducing the function arguments into the function declaration, by writing `fun semi (r : real) : real = pi * r`.

Task 5.4

After line 21 is fixed, the first error message is now on line 27, again a type mismatch where the operator and operand don't agree. Specifically, in the function declaration `fun area (d : int) : real = pi * d *` the input `d` is specified as being of type `int`, while the function definition uses the * operator to multiply pi and d. However, earlier the `pi` was defined as a value with type `real`, and the * operator cannot multiply together a real and an int. This can be fixed by declaring the type of the input d as type `real` so as to match the type of pi, like so: `fun area (d : real) : real = pi * d * d`.

Task 5.5

Once line 27 is fixed, an error is now raised at line 33, that there is an unbound variable or constructor called `Pi`. SML appears to be case-sensitive, and `Pi` (spelled with an uppercase "P") is not defined earlier, only `pi` (lowercase "p") is, so this would be fixed by replacing `Pi` with `pi`, like so: `fun vol (r : real) : real = 4.0 / 3.0 * pi * r * r * r`.

Task 6.1

The function is not satisfied in this specification. If the function were given a negative integer as an input, say `decimal ~5`, the return value would be the `int list [~5]`. However, the `ENSURES` specification says that the expression should evaluate to a list of *digits* , defined as an integer in the range 0 - 9. 5 is clearly not in the range 0 - 9, and so is not a digit, so the function does not satisfy this particular specification.

Task 6.2

The function satisfies this specification.

Task 6.3

The function satisfies this specification.

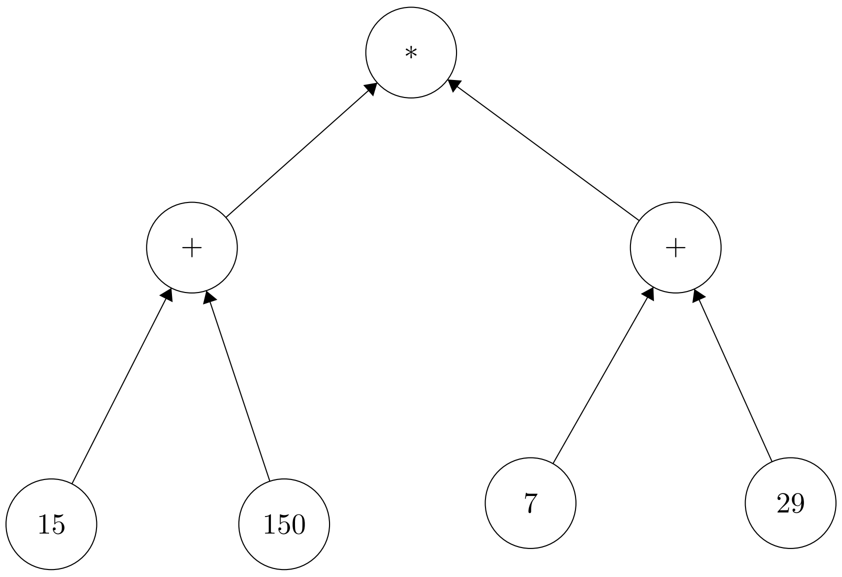
Task 6.4

The function satisfies this specification.

Task 6.5

The specification described in Task 6.3 gives the most information. The `ENSURES` comment in Task 6.4 says that decimal(n) evaluates to a list of digits, but since the `REQUIRES` comment is the same between the specifications in Task 6.3 and 6.4, it's more accurate to say that `decimal` will evaluate to a *non-empty* list of digits. The specification in Task 6.2 only `REQUIRES n > 0`, while Task 6.3 `REQUIRES n >= 0`, which also fulfills the `ENSURES` specification, which is the same across both, so Task 6.3 provides some more information about how decimal can be applied.

Task 7.1



Task 7.2

The above computation tree has work 3 and span 2.

Task 7.3

If we let $P = 2$, then by Brent's Theorem the number of steps to evaluate the expression $15 + 15) * (7 + 29)$ is at least max($\frac{W}{P}$,S). By the answer for Task 7.2, $W = 3$ and $S = 2$, so the lower bound on the number of steps is max($\frac{3}{2}$,2). This evaluates to 2, which is our lower bound for the number of steps required to evaluate the expression.

Task 7.4

	Processor 1	Processor 2
Step 1	$15 + 150 \implies 175$	$7 + 29 \implies 36$
Step 2	$175 * 36 \implies 6300$	Idle

Task 7.5

For each tree, the entire planting process takes t minutes to dig a hole, t minutes to sow the seed, and then t minutes to sprinkle water. While the sower is sowing the first seed, the digger can move on to the next seed, and while the sprinkler is sprinkling the first seed, the sower can sow the second, while the digger can dig a hole for the third seed. Following this logic, we can make the inference that if the digger is working on the nth seed, the sower is working the $(n-1)$th seed, and the sprinkler is working on the $(n-2)$th seed, for $n \geq 3$. For n apple trees, it will take the digger nt minutes to dig holes for all of them. By the time the digger is done, the sower still need to sow the nth seed, taking another t minutes, and the sprinkler needs to sprinkle both the nth and the $(n-1)$th seed, taking $2t$ minutes. Thus, the whole process will take $(n+2)t$ minutes.

Task 7.6

If there were an infinite number of diggers, sowers, and sprinklers, then we perform digging, sowing, and sprinkling of all n of the seeds in just 3 steps. It would take t minutes for n diggers out of our infinite supply to dig holes for n seeds, another t minutes for n sowers to sow them, and another t minutes for n sprinklers to sprinkle them, taking $t+t+t=3t$ minutes total.