15-150 Spring 2012
Homework 1

Out: Thursday, 19 January
Due: Wednesday, 25 January, 09:00

1 Introduction

Welcome to 15-150! The normal homework schedule this semester will be for the assignment to be posted to the course Web site on Tuesday and due on the following Wednesday morning. As this is the first week of class, the assignment was not posted until Thursday and is correspondingly shorter (worth only 50 points), to adjust for the amount of time you have to complete it.

1.1 Getting The Homework Assignment

The starter files for the homework assignment have been distributed through our `git` repository. In the first lab, you set up a clone of this repository in your AFS space. To get the files for this homework, log in to the UNIX timeshares via SSH or sit down at a cluster machine, change into your clone of the repository, and run

```
git pull
```

This should add a directory for Homework 1 to your copy of the repository, containing a copy of this PDF and some starter code in subdirectories. If this is does not work for you, contact course staff immediately. For more information about `git`, please read the documentation at

`http://www.cs.cmu.edu/~15150/resources/git.pdf`

1.2 Submission

To submit your solutions, place your `hw01.pdf` and modified `hw01.sml` files in your handin directory on AFS:

```
/afs/andrew.cmu.edu/course/15/150/handin/<yourandrewid>/hw01/
```

Your files must be named exactly `hw01.pdf` and `hw01.sml`. After you place your files in this directory, run the check script located at

```
/afs/andrew.cmu.edu/course/15/150/bin/check/01/check.pl
```

then fix any and all errors it reports.

The check script does some basic checks on your submission: making sure that the file names are correct; making sure that no files are missing; making sure that your PDF is valid; making sure that your code compiles cleanly. Note that the check script is *not* a grading script—a timely submission that passes the check script will be graded, but will not necessarily receive full credit.

Remember that your written solutions must be submitted in PDF format—we do not accept MS Word files.

Your `hw01.sml` file must contain all the code that you want to have graded for this assignment and compile cleanly. If you have a function that happens to be named the same as one of the required functions but does not have the required type, it will not be graded.

1.3 Due Date

This assignment is due on Wednesday, 25 January 2012, at 09:00 EST. Remember that this deadline is final and that we do not accept late submissions.

2 Course Resources and Policy

Please make sure you have access to the various course resources. We will post important information often. You can find more information about these resources in the Tools page of the course's Web site.

We are using Web-based discussion software called Piazza for the class. You are encouraged to post questions, but please do not post anything that gives away answers or violates the academic integrity policy.

Task 2.1 (1%). You will receive an e-mail with instructions on signing up for Piazza. Activate your account. There is announcement there that tells you a 'magic number'. What is the number?

> $\boxed{\textbf{Solution 2.1}}$ The magic number is 2701.

Task 2.2 (4%). Read the collaboration policy is on the course website. Then, for each of the following situations, decide whether or not the students' actions are permitted by the policy. Explain your answers.

1. Eric and Amy are discussing Problem 3 over Skype. Meanwhile, Eric is writing up his solution to that problem.

   > $\boxed{\textbf{Solution 2.2}}$ Eric's actions are not permitted. You may discuss Problems but you are not supposed to take any kind of notes during discussions about the homework.

2. Brandon and Abby eat lunch (at noon) while talking about their homework, and by the end of lunch, they have covered their napkins with notes and solutions. They throw out all of the napkins and go to class from 1pm-5pm. Then, each individually writes up his or her solution.

   > $\boxed{\textbf{Solution 2.2}}$ This is fine; the notes were discarded and more than 4 hours passed before anyone wrote up his or her solutions to the problem set.

3. Esha and Tyler write out a solution to Problem 4 on a whiteboard in the Gates-Hillman Center. Then, they erase the whiteboard and run to the computer cluster. Sitting at opposite sides of the room, each student types up the solution.

   > $\boxed{\textbf{Solution 2.2}}$ This is not permitted. They did not allow sufficient time to elapse between discussing the problems and writing up their solutions. If you're "running" to type up the answer, you are less likely to really understand the work that you're submitting. Letting time pass is intended to ensure that you are actually thinking about the problem rather than copying from memory.

4. Laura is working on a problem alone on a whiteboard in Gates. She accidentally forgets to erase her solution and goes home to write it up. Later, Rob walks by, reads it, waits 4 hours, and then writes up his solution. Is Laura in violation of the policy? Is Rob?

   $\boxed{\textbf{Solution 2.2}}$ Both are in violation of the policy.

   According to the University Policy on Cheating and Plagarism, you are responsible for preventing other students from accessing your work. This includes erasing whiteboards and setting appropriate file permissions. Thus, Laura is in violation of the policy.

   The only exception to the non-collaborative nature of homeworks is the whiteboard policy, which applies to discussions—which implies that both parties must be involved. Thus, reading someone else's solution when they are not present is in violtion of the policy, so Rob is in trouble.

3 Type Checking and Evaluation

In this section we will explore the step-by-step reasoning of type checking and evaluation to better understand when an SML expression is well-typed, what its type is, how it will evaluate, and what its value will be. We will also do some basic analysis of the number of steps in the evaluation of an expression.

We will start with an example. Consider the expression `intToString 7` and assume that we know `intToString : int -> string`. (Assume that `intToString` has this type throughout this section.) To determine the type of this expression, we first note that the expression `7` has the type `int`. Now, using this and the fact that `intToString` has the type `int -> string` we conclude that the application of these two expressions has the type `string` since the first expression has a function type and the type of the second expression matches the type of the argument for this function.

Task 3.1 (2%). Determine the type of the expression:

```
(intToString 3) ^ (intToString 6)
```

Describe your reasoning in the same manner as the example. If part of your reasoning exactly corresponds to that found in the example feel free to cite the correspondence rather than copying everything.

$\boxed{\textbf{Solution 3.1}}$ By the same reasoning as in the example, `intToString 3` and `intToString 6` each have type `string`. Therefore, the whole expression has type `string` since the two subexpressions of `^` each have type `string`.

Task 3.2 (2%). Explain why the expression, `intToString "1"`, is not well-typed.

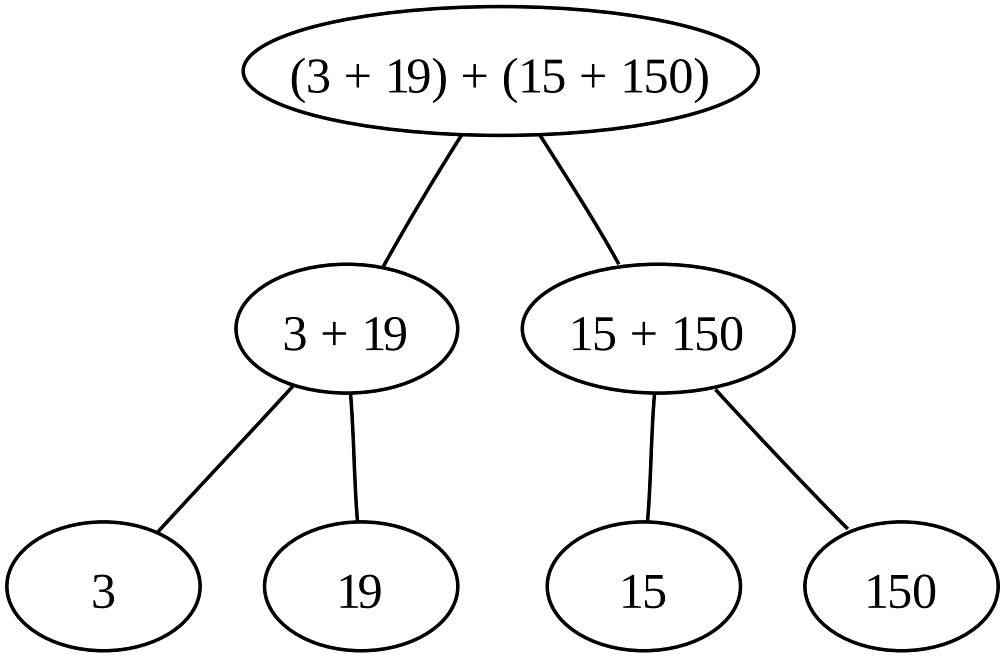

Figure 1: Arithmetic expression tree for $(3 + 19) + (15 + 150)$

$\boxed{\textbf{Solution 3.2}}$ Note that the expression `"1"` has type `string`. We assume that the expression `intToString` has the type `int -> string`. Therefore, the argument in this application does not have the type indicated by the type of the function resulting in a type error.

We will now look at an example of reasoning about evaluation. Consider the expression `(intToString 7) ^ "1"` and assume that we know the application, `(intToString 7)`, evaluates to the value `"7"`. Note that the value `"1"` evaluates to itself. Using these two facts, we conclude that the whole expression evaluates to `"71"` since the `^` operator evaluates its two subexpressions and then evaluates to the concatenation of the two strings that result from these evaluations.

Task 3.3 (2%). Determine the value that results from the evaluation of the expression:

```
intToString (fact 4)
```

Describe your reasoning in the same manner as the example. Assume that the subexpression `fact 4` evaluates to `24` and that the `intToString` function has the familiar semantics.

$\boxed{\textbf{Solution 3.3}}$ The expression evaluates to the value `"24"`. The function `intToString` evaluates to itself, and the subexpression `factorial 4` evaluates to `24` by assumption. Therefore, the outer application takes the values resulting from these two evaluations and applies them to produce the value `"24"`.

3.1 Classifying expressions

Recall, from Lecture 2, the following sets of expressions, each of which is a strict superset of the next:

- syntactically correct expressions
- well-typed expressions
- valuable expressions
- values

Task 3.4 (4%). For each of the following expressions, state the **most specific** set that the expression belongs to. Explain why your answer is correct in one or two short sentences.

1. `f (5 div (1 - 1))` where

   ```
   fun f (x : int) : int = 47
   ```

2. `6`
3. `1+1`
4. `(intToString 5) + 1`

$\boxed{\textbf{Solution 3.4}}$

1. This expression belongs to the set of well-typed expressions because it passes the type checker but it is not valuable (it raises the exception `Div`).
2. This expression belongs to the set of values because 6 is already a value.
3. This expression belongs to the set of valuable expressions because it computes to a value (2).
4. This expression belongs to the set of syntactically correct expressions because it is syntactically correct, but you will get a type error at compile-time because you can't add an `int` to a `string`.

4 Parallel Computing

In lab we discussed how to reason about the number of steps in the evaluation of an arithmetic expression. Consider the following expression:

```
(3 + 19) + (15 + 150)
```

This arithmetic expression corresponds to the tree shown in Figure 1. Each compound expression is recursively broken down into subexpressions, shown as children in the tree. Leaves correspond to *values* that cannot be broken down further (e.g. the number 3).

We define the *work*, W, of the expression tree as the total number of nodes that can be broken down further (i.e. the total number of compound expressions). The *span*, S, is the number of edges along the longest path from the root to a leaf. Regardless of the number of processors used to evaluate a compound expression in parallel, the number of steps required to evaluate the final result must be at least as great as the span because evaluating a parent requires first evaluating its children (a *data dependency*). Also, if each of P processors performs one evaluation step in parallel during each *time cycle*, it would require at least W/P time cycles to perform all W operations. These observations give the intuition behind *Brent's Theorem*:

Theorem 1 (Brent's Theorem)**.** *If an expression, e, evaluates to a value with work, W, and span, S, then evaluating e on a P-processor machine requires at least* $\max(W/P, S)$ *steps.*

Task 4.1 (5%). What are the values of work and span for the tree in Figure 1? Use Brent's Theorem to determine a lower bound on the number of steps required to evaluate this arithmetic expression on a machine with $P = 2$ processors.

Solution 4.1 By definition, $W = 3$ and $S = 2$. Therefore, Brent's Theorem states that the number of steps required is bounded by $\max(3/2, 2) = 2$. This cost graph does not have enough work to keep both processors busy so the number of steps is constrained by the depth.

Task 4.2 (5%). Describe one possible assignment of the nodes in the tree to the two processors to achieve this lower bound. In particular, for each time step, state what node each processor is evaluating. If a processor is idle during a time step state this.

Solution 4.2 During the first time step, one processor evaluates the `3 + 19` node while the other processor evaluates the `15 + 150` node. During the second time step, one processor evaluates the root while the other processor is idle.

In the first lecture, we acted out the process of counting the number of students in the class who took 15-122 last semester. Recall that this task required time proportional to the number of students when it was performed sequentially, but it could be performed in significantly less time when students worked in parallel.

Task 4.3 (5%). Following this model, describe (informally, in words), a process involving bulk operations on collections of objects or people that occurs in your life. Analyze the work (*i.e.*, how many operations it takes overall to do the task) and the span (*i.e.*, how long it takes if you could parallelize arbitrarily much).

> **Solution 4.3** Many answers are acceptable, but your answer should follow the same general pattern as the example.

5 Interpreting Error Messages

Download the file `hw01.sml` from the `git` repository as described in Section 1.1.

You can evaluate the SML declarations in this file using the command

```
use "hw01.sml";
```

at the SML REPL prompt. Unfortunately, the file has some errors that must be corrected. The next five tasks will guide you through the process of correcting these errors.

Task 5.1 (2%). What error message do you see when you evaluate the unmodified `hw01.sml` file? What caused this error and how can it be fixed?[1]

> **Solution 5.1** The syntax error is:
>
> ```
> ./hw1.sml:30.9 Error: syntax error: inserting  BAR
> ```
>
> This syntax error is caused by a missing vertical bar in the syntax of the case statement and is fixed by inserting a vertical bar in front of the second arm of the case in the `fact` function.

Correct this one error in the `hw01.sml` file and evaluate it again using the same command.

Task 5.2 (2%). Now with that error corrected what is the first of the remaining errors? What caused this error?[2]

> **Solution 5.2** The first of the new errors is:

[1] *Hint:* Compare the syntax of the case statement in `double` and the one in `fact`. What is different?

[2] *Hint:* Compare the syntax of the first line of the declaration of the function `double` and the first line of the declaration of the function `fact`. What is different?

```
./hw1.sml:11.5-11.11 Error: can't find function arguments in clause
```

This error is caused by the missing argument in the declaration of the function `double`. It is fixed by adding the variable `(n : int)` after `double`.

Again, correct just this one error in the `hw01.sml` file and evaluate it.

Task 5.3 (2%). What is the first of the remaining error messages after these two bugs have been corrected? What does this error message mean? How do you fix this error?

Solution 5.3 The first of the remaining error messages is:

```
./hw1.sml:28.10 Error: unbound variable or constructor: y
```

This error is caused by the reference to the unbound variable, `y`, in the case statement. It is fixed by changing the variable `y` to `n`

Fix this error and evaluate the file again. There are still a couple more errors.

Task 5.4 (2%). What error message does the evaluation of the resulting file produce? How do you fix this error?[3]

Solution 5.4 The remaining error message is:

```
./hw1.sml:30.19-30.28 Error: unbound variable or constructor: factorial
```

This error is caused by the application of the unbound variable, `factorial`, in the case statement. It is fixed by changing the application of `factorial` to `fact`

After you fix this error there should be two type error messages remaining.

Task 5.5 (2%). What are these error messages? What do these error messages mean? How do you fix this error?[4]

Solution 5.5 The remaining type error messages are:

[3] *Hint:* There are two simple ways to fix this error. Please choose the one that keeps the name of the `fact` function the same.

[4] *Hint:* Both error messages are caused by the same error. All expressions on the right-hand-side of a given case statement must have the same type, and this is the type of the case statement.

```
./hw1.sml:28.5-30.32 Error: types of rules don't agree [tycon mismatch]
  earlier rule(s): int -> real
  this rule: int -> int
  in rule:
    n => n * fact (n - 1)
./hw1.sml:27.5-30.32 Error: right-hand-side of clause doesn't agree
with function result type [tycon mismatch]
  expression:  real
  result type:  int
  in declaration:
    fact =
      (fn n : int =>
            (case n
              of <pat> => <exp>
               | <pat> => <exp>): int)
```

The first error message means that the first arm of the case statement matches an `int` and produces a `real`, but the second arm matches an `int` and produces an `int`. The second error message means that the type of the body of the function does not match the expected result type of the function. This is fixed by changing the constant `1.0` of type `real` to the constant `1` of type `int`

When you correct this final error and evaluate the file there should be no more error messages.

6 Writing Functions

Now that you have a better understanding of the declarations in this file, you will make a modified version of one of these functions in the next task.

Task 6.1 (10%). Copy and paste the code for the `double` function to the bottom of your `hw01.sml` file. Change the name of the function in the new copy to `triple` and modify the code to return three times the argument instead of twice the argument.

For this problem you may only apply the `+` operator when at least one argument is constant (e.g. `2 + x` is okay, but `x + x` is not), and you may not call `double` in the definition of `triple`. Instead you must define `triple` recursively following the pattern of `double`.