15-150 Fall 2013
Homework 07

Out: Wednesday, 9 October 2013
Due: Monday, 14 October 2013 at 23:59 EST

1 Introduction

1.1 Getting The Homework Assignment

The starter files for the homework assignment have been distributed through our `git` repository, as usual.

1.2 Submitting The Homework Assignment

Submissions will be handled through Autolab, at

`https://autolab.cs.cmu.edu`

In preparation for submission, your `hw/07` directory should contain a file named exactly `hw07.pdf` containing your written solutions to the homework.

To submit your solutions, run `make` from the `hw/07` directory (that contains a `code` folder and a file `hw07.pdf`). This should produce a file `hw07.tar`, containing the files that should be handed in for this homework assignment. Open the Autolab web site, find the page for this assignment, and submit your `hw07.tar` file via the "Handin your work" link.

The Autolab handin 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 code compiles cleanly. Note that the handin script is *not* a grading script—a timely submission that passes the handin script will be graded, but will not necessarily receive full credit. You can view the results of the handin script by clicking the number (usually either 0.0 or 1.0) corresponding to the "check" section of your latest handin on the "Handin History" page. If this number is 0.0, your submission failed the check script; if it is 1.0, it passed.

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

Your `hw07.sml` file must contain all the code that you want to have graded for this assignment, and must 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 Monday, 14 October 2013 at 23:59 EST. Remember that you may use a maximum of one late day per assignment, and that you are allowed a total of three late days for the semester.

1.4 Methodology

You must use the five step methodology discussed in class for writing functions, for **every** function you write in this assignment. Recall the five step methodology:

1. In the first line of comments, write the name and type of the function.
2. In the second line of comments, specify via a `REQUIRES` clause any assumptions about the arguments passed to the function.
3. In the third line of comments, specify via an `ENSURES` clause what the function computes (what it returns).
4. Implement the function.
5. Provide testcases, generally in the format
   `val <return value> = <function> <argument value>`.

For example, for the factorial function presented in lecture:

```
(*  fact : int -> int
 *  REQUIRES:  n >= 0
 *  ENSURES: fact(n) ==> n!
*)

fun fact (0 : int) : int = 1
  | fact (n : int) : int = n * fact(n-1)

(* Tests: *)

val 1 = fact 0
val 720 = fact 6
```

2 Regular Expressions

In class, we introduced six different constructors to describe regular expressions. We will be extending this definition with two new constructors: `Whatever` and `Both`. The extended datatype is given below.

```
datatype regexp =
    Zero
  | One
  | Char of char
  | Plus of regexp * regexp
  | Times of regexp * regexp
  | Star of regexp
  | Whatever
  | Both of regexp * regexp
```

The new constructors have the following definitions:

- `Whatever` is a string wildcard that accepts whatever, that is any finite list of characters:

$$\mathcal{L}(\texttt{Whatever}) = \{L \mid L \text{ is a list of characters}\}.$$

- $\texttt{Both}(R_1, R_2)$ accepts a string if and only if it is in both $\mathcal{L}(R_1)$ and $\mathcal{L}(R_2)$:

$$\mathcal{L}(\texttt{Both}(R_1, R_2)) = \{L \mid L \in \mathcal{L}(R_1) \text{ and } L \in \mathcal{L}(R_2)\}.$$

The regular expression matcher `match` from class is included in the support code for the assignment. We have extended the `datatype` definition of `regexp` to include the new constructors `Whatever` and `Both`. Your job is to extend `match` to deal with these new constructors, and prove the correctness of your implementation. We define correctness as follows:

Theorem 1 (Correctness)**.** *For R : `regexp`, let $P(R)$ be the following statement:*

For all values L : `char list` and total functions p : `char list` $\to$ `bool`,

a. `match` R L p = `true` *if there exist L_1, L_2 : `char list` such that $L = L_1 @ L_2$, $L_1 \in \mathcal{L}(R)$, and $p(L_2)$ = `true`,*

b. `match` R L p = `false` *otherwise.*

Then, `match` is correct if for all R : `regexp`, P(R) holds.

In each of the following coding tasks, we strongly recommend that you think through the correctness spec when you are writing the code. If you're stuck on the implementation, try doing the proof first—this will guide you to the answer.

Task 2.1 (8%). Implement the case of `match` for `Whatever`, the string wildcard.

Solution 2.1 See solution in `hw07.sml`.

Task 2.2 (10%). Implement the case of `match` for `Both`(R_1, R_2).

Solution 2.2 See solution in `hw07.sml`.

Task 2.3 (7%). We give part **(b)** of the correctness proof of `Times`(R_1, R_2) below, with certain parts left out. Fill in the seven sections of blanks with the appropriate statements. Note that some of the sections have more than one blank, and some of the sections are used more than once. Each numbered section is worth 1 point.

Proof. Assume L, p such that `match` (`Times`(R_1, R_2)) L p = `true`. We need to show that $\exists L1, L2$ such that $L1@L2 = L$ with $L1 \in \mathcal{L}$ (`Times`(R_1, R_2)) and p $(L2)$ = `true`.

Inductive Hypotheses Assume $P(R_1)$ and $P(R_2)$, i.e.

1. $\forall L_1$: `char list`, p_1 : `char list` $\rightarrow$ `bool` s.t. p_1 total, if `match` R_1 L_1 p_1 = `true`, then $\exists L1_1, L2_1$ *s.t.* $L1_1@L2_1 = L_1$, $L1_1 \in \mathcal{L}(R_1)$, p_1 $L2_1$ = `true`
2. $\forall L_2$: `char list`, p_2 : `char list` $\rightarrow$ `bool` s.t. p_2 total, if `match` R_2 L_2 p_2 = `true`, then $\exists L1_2, L2_2$ *s.t.*$L1_2@L2_2 = L_2$, $L1_2 \in \mathcal{L}(R_2)$, p_2 $L2_2$ = `true`.

By stepping:

$$\begin{aligned} & \texttt{match (Times}(R_1, R_2)\texttt{)}\ L\ p \\ = {} & \texttt{case Times(}R_1\texttt{,}R_2\texttt{) of ...} \quad \texttt{| Times(R1,R2) => ...} \quad \texttt{| ...} \\ = {} & (1)\ \texttt{match}\ R1\ L\ \texttt{(fn L' => match}\ R2\ \texttt{L' }p\texttt{)} \end{aligned}$$

By assumption, `match` (`Times`(R_1, R_2)) L p = `true`. So by transitivity

$$(i)\quad (1)\ \texttt{match}\ R1\ L\ \texttt{(fn L' => match}\ R2\ \texttt{L' }p\texttt{)} = \texttt{true}$$

Thus, by (2) $P(R1)$, taking p_1 to be

$$\texttt{(fn L' => match}\ R_2\ \texttt{L' p)},$$

L_1 to be L, and using (i) to satisfy the premise, we know that $\exists L1_1, L2_1$ such that (3) $L1_1@L2_1 = L$, (3) $L1_1 \in L(R_1)$, and (3) `(fn L' => match` R_2 `L'` $p)L2_1$ = `true`.

By stepping

$$\begin{aligned} & \texttt{(fn L' => match}\ R_2\ \texttt{L' }p\texttt{)}L2_1 \\ = {} & \texttt{match}\ R_2\ L2_1\ p \end{aligned}$$

Thus, by transitivity,

$$(ii)\ \texttt{match}\ R_2\ L2_1\ p = \texttt{true}$$

By (4) IH2, taking p_2 to be p and L_2 to be $L2_1$ and using (ii) to satisfy the premise, we know that $\exists L1_2, L2_2$ such that (5) $L1_2@L2_2 = L2_1$, (5) $L1_2 \in \mathcal{L}(R_2)$, and ($p\ L2_2 = \texttt{true}$) .

Since $L1_1 \in \mathcal{L}(R_1)$ and $L1_2 \in \mathcal{L}(R_2)$, (6) $L1_1@L1_2 \in \mathcal{L}(\texttt{Times}(R_1, R_2))$ by (7) by the definition of `Times`.

Now, take $L1$ to be (6) $L1_1@L1_2$ and $L2$ to be $L2_2$. Then, $L1 =$ (6) $L1_1@L1_2 \in \mathcal{L}(\texttt{Times}(R_1, R_2))$, and $L1@L2 =$ (6) $L1_1@L1_2@L2_2 = L1_1@L2_1 = L$, and $p\ L2 = p\ L2_2 = \texttt{true}$. So, the theorem holds for this case. □

Task 2.4 (15%). Do part **(b)** of the correctness proof of $\texttt{Both}(R_1, R_2)$. Your proof should follow the format of the proof given above. You should assume $P(R_1)$, $P(R_2)$, and that `match` is total. Then, proving **(b)** is the same as proving the following theorem:

> For all values L : `char list` and total functions p : `char list` $\rightarrow$ `bool`,
> If `match` $(\texttt{Both}(R_1, R_2))\ L\ p = \texttt{true}$, then there exist L_1, L_2 such that $L = L_1@L_2$, $L_1 \in \mathcal{L}(\texttt{Both}(R_1, R_2))$, and $p(L_2) = \texttt{true}$.

Do this proof carefully! There is a plausible-looking, but incorrect, implementation of `Both`; this case of the proof will fail if your code has this bug.

Solution 2.4

Proof. Assume L, p such that `match` $(\texttt{Both}(R_1, R_2))\ L\ p = \texttt{true}$. We need to show that $\exists L_1, L_2$ such that $L_1@L_2 = L$ with $L_1 \in \mathcal{L}(\texttt{Both}(R_1, R_2))$ and $p\ L_2 = \texttt{true}$.

Inductive Hypotheses :

1. $\forall L_1$:`char list`, p_1 :`char list -> bool`, if `match` $R_1\ L_1\ p_1 = \texttt{true}$, then $\exists L1_1, L2_1$ *s.t.* $L1_1@L2_1 = L_1, L1_1 \in \mathcal{L}(R_1), p_1\ L2_1 = \texttt{true}$
2. $\forall L_2$:`char list`, p_2 :`char list -> bool`, if `match` $R_2\ L_2\ p_2 = \texttt{true}$, then $\exists L1_2, L2_2$ *s.t.* $L1_2@L2_2 = L_2, L1_2 \in L(R_2), p_2\ L2_2 = \texttt{true}$

By stepping:

```
  match (Both (R1, R2)) L p
= case (Both (R1, R2)) of ...  | Both (R1, R2) => ...  | ...
= match R1 L (fn L' => match R2 L (fn L'' => L' = L'' andalso p L''))
```

By assumption, `match (Both (`R_1`, `R_2`)) L p` $=$ `true`. So by transitivity

(i) `match` R_1 `L (fn` L' `=> match` R_2 `L (fn L'' => L' = L'' andalso p L''))` $=$ `true`

We can then apply IH1, taking p_1 to be

`(fn` L' `=> match` R_2 `L (fn L'' => L' = L'' andalso p L''))`

and L_1 to be L, and using (i) to satisfy the premise. Then we know that $\exists L1_1, L2_1$ such that $L1_1@L2_1 = L$, $L1_1 \in L(R_1)$, and $p_1\ L2_1 =$`true`.

By stepping

`(fn` L' `=> match` R_2 `L (fn L'' => L' = L'' andalso p L''))` $L2_1$
$=$ `match` R_2 `L (fn L'' =>` $L2_1$ `= L'' andalso p L'')`

Thus, by transitivity,

(ii) `match` R_2 `L (fn L'' =>` $L2_1$ `= L'' andalso p L'')` $=$ `true`

We can then apply IH2, taking p_2 to be

`(fn L'' =>` $L2_1$ `= L'' andalso p L'')`

and L_2 to be `L` and using (ii) to satisfy the premise. Then we know that $\exists L1_2, L2_2$ such that $L1_2@L2_2 = L$, $L1_2 \in L(R_2)$, and $p_2\ L2_2 =$`true`.

By stepping

`(fn L'' =>` $L2_1$ `= L'' andalso p L'')` $L2_2$
$=$ $L2_2 = L2_1$ `andalso p` $L2_2$

Thus, by transitivity, $L2_1 = L2_2$ `andalso p` $L2_2$ evaluates to `true`. By inversion on `andalso`, $(L2_1 = L2_2) =$ `true` and `p` $L2_2 =$ `true`. By lemma, since $L1_1@L2_1 = L$, $L1_2@L2_2 = L$, and $L2_1 = L2_2$, it must be the case that $L1_1 = L1_2$—if there are two splittings of L with the same suffix, then the prefixes must be the same.

So, take L_1 to be $L1_1$ and L_2 to be $L2_1$. Then $L_1 = L1_1 \in L(R_1)$ and $L_1 = L1_1 = L1_2 \in L(R_2)$. So $L_1@L_2 = L1_1@L2_1 = L$, $L_1 \in L(\texttt{Both}(R_2, R_2))$ by definition, and $p\ L_2 = p\ L2_1 =$ `true`. □

3 Irregular Expressions

Turns out, our implementation of a regular expression matcher can recognize more than just regular languages[1]. In this section, we will explore this idea further.

Task 3.1 (10%). Write the function `halfmatch` which takes two regular expressions, R_1 and R_2, a character list, L, and evaluates to `true` if and only if there exist L_1, L_2 : `char list` such that:

- $L = L_1$ @ L_2
- $length\ L_1 = length\ L_2$
- $L_1 \in \mathcal{L}(R_1)$ and $L_2 \in \mathcal{L}(R_2)$.

For this task, we want to use the power of the regular expression system we have in place; you should define your answer in terms of regular expressions and `match`. Your solution should not be recursive, and you should not define any helper functions. Do not use list functions apart from `length`.

Solution 3.1 See solution in `hw07.sml`.

Task 3.2 (0%). Now, let `irregular` = `halfmatch` (`Star`(`Char` #"0")) (`Star`(`Char` #"1")). Then, $\mathcal{L}$(`irregular`) = $\{0^n 1^n \mid n \in \mathbb{N}\}$. This language turns out to be irregular. (You may prove this fact in higher level courses like 15-251 or 15-453.)

[1] It is actually Turing complete.