# 15-150 Assignment 07

Jack Kasbeer
jkasbeer@andrew.cmu.edu
Section K
October 28, 2015

## 2: Continuations

1. `findOne` is implemented in `hw07.sml`
2. `findTwo` is implemented in `hw07.sml`

## 3: Advanced Path-Finding

1. `path` is implemented in `hw07.sml`
2. `optionPath` is implemented in `hw07.sml`
3. No, the span of `path` is not less than the work of `path` because nothing is being executed in parallel. We're only ever working with one branch of a tree; there are no simultaneous recursive calls to more than one branch. Hence, the work and span will be the same.
4. `chop` is implemented in `hw07.sml`
5. `pathLength` is implemented in `hw07.sml`

## 4: Regexp

1. `badDiff` is incorrect because `not(match r2 cs k)` is a very different result than `p`, from the match found in the call to `match r1 cs k` (satisfying `p@s == cs`), not being in `L(r2)`; it will yield `false` more often than it should. This is because `match` is going to be looking for ways to compose `cs` with the language of `r2`, which is incorrect. Instead, after the first call to `match`, the function should be checking to see if `p` is in the language of `r2`, and returning false if it is, and true otherwise.

   Consider the case of `r1 = bo`, `r2 = boo`, and our `cs = ["b","o","o"]`.
   `badDiff r1 r2 cs k => (match r1 cs k) andalso not(match r2 cs k)`. Both calls to `match` will evaluate to `true` since the call to `r1` will be matched by `["b","o"] @ ["o"]`, and the second call to `r2` will be matched by `["b","o","o"] @ []`. But, notice that `true andalso not(true) => false`, which is an incorrect result. The `p` found in the first call to `match` was `["b","o"]`, and the specification requires that this is not in `L(r2)`, which it isn't! So this should, in fact, evaluate to `true`.
2. `diff` is implemented in `hw07.sml`

3. Prove **Theorem 1:**

   For all values r1: `regexp`, r2: `regexp`, cs: `char list`, k: `char list -> bool`, if there exist values $p, s$ such that $p@s = cs$ with $p \in L(r)$ and $k\,s =$`true`,
   then `match` r cs k = `true`.

   *Proof.* Assume for all values r1: `regexp`, r2: `regexp`, cs: `char list`, k: `char list -> bool`, `diff r1 r2 cs k = true`.
   WWTS that there exists $p, s$ such that $p@s = cs$, where $p \in L(r1/r2)$ and $k\,s =$ `true`.

   By assumption and def. of `diff`,
   `diff r1 cs k =>`
   `(match r1 cs (fn C => (k C) andalso not(match r2 cs (fn A => C = A)))) = true`.

   This implies that $\exists p, s$ such that $p@s = cs$, where $p \in L(r1)$ (by Theorem 2: Soundness).
   Notice that by binding `L:s`, this means `k s andalso not(match r2 cs (fn A => s = A)) = true`.
   By Lemma 1, this trivially implies `k s = true` and `not(match r2 cs (fn A => s = A)) = true`.
   And by Lemma 2, `match r2 cs (fn A => s = A) = false` (use of `not`).

   By Theorem 3, we know if there $\exists p, s$ such that $p@s = cs, p \in L(r2)$ and $k'\,s =$ `true`, then `match r2 cs k' = true`. Since `match r2 cs k' = false`, it must be the case that `not(p`$\in L(r2)$ and `k' s = true) is \verb`true—, or `(p` $\in L(r2)$ and `k' s = true` is false. If this wasn't the case then completeness would not be maintained. We know that `k' s = true` for `k' = (fn A => s = A)`, which means that $p \notin L(r2)$.

   By definition of Set Difference, $p \in L(r1)$ and $p \notin L(r2) \Rightarrow p \in L(r1/r2)$. Hence, there exists $p, s$ such that $p@s = cs$, where $p \in L(r1/r2)$, and $k\,s =$`true`, which means that `diff r1 r2 cs k` is sound, proving **Theorem 1**, and we're done. □

4. `messageKey` is defined in `hw07.sml`

5. `findMessage` is implemented in `hw07.sml`

6. After wandering through the cave aimlessly, I was able to find the treasure in the fountain: `(fn x => (fn y => x))`.