Can continuations help us escape from this alligator?

Intuitive Justification

Several real-world situations that may or may not analogous to use of continuations are described below. For each situation, explain why or why not the situation resembles use of a continuation.

1. Alan is reading a Goosebumps “choose your own adventure” book. Page 32 says “A large purple alligator is standing in front of you. It gnashes its teeth menacingly. A steel crowbar lies on the ground in front of you. To pick up the crowbar, turn to page 73. To run away, turn to page 89.” Alan holds the current page (32) with his finger and turns to page 73, which reads “as you reach down to pick up the crowbar, the alligator charges! The force of the blow loosens your grasp on the crowbar and knocks you to the floor. The alligator begins to feast on your entrails. You are dead. The end.” Alan turns back to page 32 and decides to run away instead.

Solution: This situation resembles use of a continuation. Alan is trying to find an adventure scenario in which he does not die, so whenever he makes a choice that might lead to death, he must make sure he remembers which choice he made so he can make the other choice if he ends up dying.

2. Leslie wants to know if she has the Pokemon cards for the entire evolution tree for the Char* Pokemon (i.e., Charmander, Charmeleon, Charizard). She flips through her deck of Pokemon cards and confirms that she does indeed have a Charmander, a Charmeleon, and a Charizard.
3. Maurice is looking for his lost dog. He creates a list of three locations where his dog may be: the park, the town square, and the mall. Maurice checks the park first and does not find his dog, then checks the town square, where he finds his dog and returns home. He does not go on to search the mall.

**Solution:** This situation could resemble use of a continuation. Maurice has three choices for locations where he might check for the dog, and he needs to remember what the other locations are if he does not find the dog at the current place.

What is common to all of the situations that involve continuations?

**Solution:** In each continuation-using situation, there is a search that requires making choices that might need to be undone later in order to make sure that the entire search space is explored. We sometimes call this kind of problem backtracking search.

**Implementation**

1. Write a tail-recursive function `substr(sub : List[Char], str : List[Char]) : Boolean` that uses continuations and returns true if `sub` is a continuous substring of `str`. For example, “ab” is a continuous substring of “aaab”, but “ab” is not a continuous substring of “tacsbx” or “baaac”.

Hints:

- Your function should only compare single characters to each other (not lists to lists or char’s to lists)
- You should use a helper function that uses a failure continuation with type (Unit => Boolean).
- What should the default behavior of the failure continuation be?
- When can we be sure that we’ve found a match?
- When do we have to invoke the failure continuation?
• Make sure your implementation produces the correct result on the following
(sub,str) pairs:

aaa aa
cd fcd
ab acb
cd ecd
ab xxacb

**Solution:** Here’s a simple version that’s correct, but inefficient. Can you spot the
inefficiency?

```scala
def substr(sub : List[Char], str : List[Char]) : Boolean = {
def helper (subH : List[Char], strH : List[Char],
          fc : ((Unit) => Boolean)) : Boolean = {
    (subH, strH) match {
      case (Nil, _) => true
      case (_, Nil) => fc()
      case (c0::c1, c2::c3) =>
        if (c0 == c2) {
          helper(c1, c3, ((_ : Unit) => helper(sub, c3, fc)))
        } else {
          helper(sub, c3, fc)
        }
    }
  }
  helper(sub, str, ((_ : Unit) => false))
}
```

Here’s another version that’s slightly more complicated, but is more efficient.

```scala
def subStrMatchTR(small: String, big: String): Boolean = {
def subStrMatchTRH(small: List[Char], big: List[Char],
inMatch: Boolean, fc: Unit => Boolean): Boolean = {
  (small, big) match {
    case (Nil, _) => true
    case (_, Nil) => fc()
    case (sh :: st, bh :: bt) =>
      if (sh == bh && inMatch) subStrMatchTRH(st, bt, true, fc)
      else if (sh == bh) {
        subStrMatchTRH(st, bt, true,
          (_: Unit) => subStrMatchTRH(small, bt, false, fc))
      } else if (inMatch) fc()
  }
}
```
else subStrMatchTRH(small, bt, false, fc)
}
subStrMatchTRH(small.toList, big.toList, false, ((_: Unit) => false)
}