COP4020 Programming Assignment 3

1. Write new Scheme functions det, noun, verb, and adj such that:

   • det takes a list and returns the cdr of the list if the first word in the list is a determiner. Otherwise, it returns an empty list ’().
   • noun takes a list and returns the cdr of the list if the first word in the list is a noun. Otherwise, it returns an empty list ’().
   • verb takes a list and returns the cdr of the list if the first word in the list is a verb. Otherwise, it returns an empty list ’().
   • adj takes a list and returns the cdr of the list if the first word in the list is an adjective. Otherwise, it returns an empty list ’().

The vocabulary is limited to the words: a, an, the, tree, truck, cat, rain, hurts, rides, hits, hairy, freezing, green, hot, black. You are allowed to use the det?, noun?, verb?, and adj? functions introduced in Programming Assignment 2.

Save your Scheme functions in a file named pr3.scm. Login to linprog.cs.fsu.edu and type ’scheme’. Test your Scheme functions:

1 ]=> (load "pr3"
2 ]=> (det ’(a cat))
;Value: (cat)
3 ]=> (det ’(black truck))
;Value: ()
3 ]=> (adj ’(black truck))
;Value: (truck)

2. Consider the syntax of a simple sentence that is composed of a determiner followed by a noun:

   \[
   \langle \text{ simplesentence} \rangle ::= \langle \text{ det} \rangle \langle \text{ noun} \rangle
   \]

   To check if a sentence is well formed, i.e. it starts with a determiner followed by a noun, we use the functions as building blocks. For example, we can parse the sentence "a cat" by composing the noun and det functions:

1 ]=> (load "pr3"
2 ]=> (noun (det ’(a cat $)))
;Value: ($)
Use the det and noun functions to write a parser for simple sentences such that

1 ]=> (load "pr3")
2 ]=> (simplesentence '(a cat $))
   ;Value: ($)

3. Write a parser to analyze sentences defined by the grammar

   ⟨nounphrase⟩ ::= [ ⟨det⟩ ] ⟨noun⟩

   The determiner is optional, so a syntactically correct noun phrase may or may not start with a determiner.

   Write a function that implements a parser for nounphrase, such that:

1 ]=> (load "pr3")
2 ]=> (nounphrase '(a cat $))
   ;Value: ($)
3 ]=> (nounphrase '(black cat $))
   ;Value: ()
3 ]=> (nounphrase '(cat $))
   ;Value: ($)

   Use the det? function to test the presence of a determiner.

4. Write functions to implement a parser for the following grammar:

   ⟨sentence⟩ ::= ⟨nounphrase1⟩ ⟨verbphrase⟩
   ⟨nounphrase1⟩ ::= [ ⟨det⟩ ] ⟨nounphrase2⟩
   ⟨nounphrase2⟩ ::= ⟨adj⟩ ⟨nounphrase2⟩
   ⟨nounphrase2⟩ ::= ⟨noun⟩
   ⟨verbphrase⟩ ::= ⟨verb⟩ ⟨nounphrase1⟩

   Such that:

1 ]=> (load "pr3")
2 ]=> (sentence '(the hot green truck hits a hairy black cat $))
   ; Value: ($)
3 ]=> (sentence '(a rides tree $))
   ; Value: ()

   Each nonterminal has one Scheme function to implement the parsing operations. When more than one production is given for a nonterminal, a test in the function’s body is necessary to choose which grammar production to apply and which operations to perform. Repetitions should be implemented by recursion, e.g. the nounphrase2 function checks for the presence of an adjective then calls adj and passes the returned list as an argument to a recursive call to nounphrase2.