University of Auckland, Department of Computer Science

07.363 — Assignment 4 — English Grammar Checker

Christian Collberg
September 28, 1994

1 Introduction

Some modern word processors have a limited ability to check a text for grammatical correctness and conformance to stylistic conventions. Such programs are invaluable for those who have English as a second language, but they are certainly useful for native English speakers as well.

2 A Grammar Checker (400 marks)

Your task is to write a grammar checker for a fragment of English. You should provide two predicates: `checks` which prompts the user for a sentence and checks it for correctness, and `checkf` which does the same for an entire file of English text.

Here is an example interaction:

```
?- checkf(myfile).
   (Line 15) "Lucy like John."
      *** Error:  Subject-verb-agreement
   (Line 19) "Boy, is Lucy ever hot on John!"
      *** Information:  Cannot parse this sentence.
   (Line 23) "The boy which Lucy likes has green hair."
      *** Error:  Use <who> not <which> to refer to a person.
   (Line 26) "I like the Lucy"
      *** Error:  You can't have an article before a proper noun.
   (Line 31) "The dog that licked he was brown."
      *** Error:  You need the object form of the pronoun:  <him>, not <he>.
   (Line 67) "Him likes it."
      *** Error:  You need the subject form of the pronoun:  <he>, not <him>.
```

```
(Line 74) "John killed"
   *** Error:  <killed> needs an object.
(Line 83) "John died the dog."
   *** Error:  <died> cannot take an object.
(Line 84) "The man on the corner."
   *** Error:  Every sentence needs a verb.
(Line 90) "Like him."
   *** Error:  Every sentence needs a subject.
(Line 97) "He not like Lucy."
   *** Error:  <Not> needs an auxiliary verb (do, have,...).
(Line 99) "He ain't got no money."
   *** Error:  Avoid double negations.
```

The exact form of the error messages are left up to you, but the more informative and user friendly the better. The ultimate grammar checker will (much like an interactive spelling checker) provide a list of possible *corrected* sentences!

The aspects of English grammar necessary to complete this assignment can be found in Lecture Note 15.

3 Extension I (100 marks)

When the parser encounters a word not in the lexicon, it should prompt the user for a definition of that word. For example,

```
?- checkf(myfile).
   I don't know the word <men>.  Is it:
      n) a noun?
      v) a verb?
      a) an adjective?
n
   Is it
      s) a singular noun?
      p) a plural noun?
p
   I'm adding the plural noun <men> to my lexicon.
```

You may of course make the interaction different, for example by using dialog boxes.

4 Extension II (100 marks)

Extend `checks` to display a parse tree for correct sentences. A rudimentary tree printing package will be provided for you on the server.

5 Useful Predicates

Here are some Prolog predicates that you might find useful:

`old('TEXT', File, Volume)` Opens the standard Macintosh dialog box to allow the user to select a text file. Does *not* open the file. `File` is the name of the file (an atom).

`open(File, Volume)` Opens a file for reading and writing.

`close(File)` Closes a file.

`get0(File, C)` Reads a character from a file. `File` is the same atom (the file name) as was used in the calls to `old` and `open`. There is also a `get(File, C)` predicate which skips non-printable characters.

`:- dynamic(Pred)` Allows `asserts` of compiled predicates. I haven't been able to get this to work under LPA Prolog 2.5. Instead you can select the `"Interpreted"` compilation mode in the `"Defaults"` dialog box under the `"File"` menue. However, this will make your program run slightly slower.

If you're running MacProlog32 rather than LPA Prolog 2.5 or 4.5 then some of the calls are different:

`old('TEXT', File)` Selects a text file.

`open(File, read)` Opens a file for reading.

`get0(C)` $<\sim$ `File` Reads a character.

`:- dynamic(Pred/Arity)` Allows `asserts` of compiled predicates. For example, the declaration `:- dynamic(noun/2)` will allow you to add new facts `noun(X, Y)` to your database, even if your program defined some nouns initially.

Slides 32–34 in Lecture Note 12 show a predicate `read_in` which you can modify to read in sentences from a file.

6 Administrivia

This assignment is worth 400–600 marks. You may do one, both, or none of the extensions. You will be marked on

1. the number and difficulty of the English language constructs your parser accepts
2. the clarity and structure of your parser and lexicon
3. the quality of your error messages, and
4. the quality of the user interaction.

It is important that you clearly document what your program can and can't do, and that you *provide example input files and sentences* that demonstrate its abilities. Furthermore, you should indicate to the marker which Prolog version your program was developed under.

The assignment should be handed in electronically *and on paper* no later than noon, Tuesday, October 4.