

Project: Text Generation (Part Two)

Summary: In this laboratory, you will explore techniques for generating “random” texts.

Contents:

- Exercises
 - Exercise 1: Preparation
 - Exercise 1: From Sequences to Strings
 - Exercise 2: More Noun Phrases
 - Exercise 3: Structures in Files
 - Exercise 4: Checking Structure Files
 - Exercise 5: Starting Unification
 - Exercise 6: Sentence Forms
 - Exercise 7: Indefinite Articles
- For Those With Extra Time

Exercises

Exercise 1: Preparation

a. In DrScheme, reopen your copy of `textgen.ss`, as we will be updating that version.

b. If you have not already done so, create a file, `iverbs.txt`, that contains a sequence of intransitive verbs in the required format. For example, you might write

```
("sleeps" 300)
("screams" 200)
("speaks" 400)
("slithers" 100)
```

c. Create a file, `exclamations.txt`, that contains a variety of exclamations in the required format. For example, you might write

```
("wow" 300)
("oh boy" 300)
("wowie zowie" 100)
("burninate" 50)
("wicked neat" 250)
```

d. Create a file, `names.txt`, that contains a variety of first names in the required format.

e. Verify that `generate-part-of-speech` works with intransitive verbs, exclamations, and names.

Exercise 1: From Sequences to Strings

- a. Add the `build-from-symbol` procedure from the reading to `textgen.ss`.
- b. Try a few examples to check that it seems to work correctly.
- c. Add the `join` procedure from the reading to `textgen.ss`.
- d. Try a few examples to check that it seems to work correctly.
- e. Add the `apply-procedure` procedure from the reading to `textgen.ss`.
- f. Try a few examples to check that it seems to work correctly. For example, you should try things like
 - `(apply-procedure 'join (list "a" "b" "c"))`
 - `(apply-procedure 'capitalize (list "hello"))`
- g. Add the `build` procedure from the reading.
- h. Try a few examples to make sure that it works correctly. For example, you might try things like
 - `(build '(join (capitalize noun) space (capitalize iverb)))`
 - `(build '(join article space adjective space noun))`
 - `(build '(capitalize (join article space adjective space noun)))`
 - `(build '(join name "'s" space adjective space noun))`

Exercise 2: More Noun Phrases

- a. Update `noun-phrase` to use `build`. You might choose to use the form from the reading (which follows) or you might choose to use your own form.

```
(define noun-phrase
  (lambda ()
    (let ((choice (random 100)))
      (cond
        ; 50% of the time, we use the article adjective noun structure
        ((< choice 50)
         (build '(join article space adjective space noun)))
        ; 25% of the time, we use the article noun structure
        ((< choice 75)
         (build '(join article space noun)))
        ; 15% of the time, we use a name
        ((< choice 90)
         (build 'name))
        ; 10% of the time, use a possessive
        (else
         (build '(join name "'s" space noun)))))))
```

- b. Generate a few noun phrases to see that the new technique works.
- c. Generate a few sentences to see that the new technique works indirectly as well as directly.

Exercise 3: Structures in Files

- a. Add the `random-structure` procedure from the reading to `textgen.ss`.
- b. Create a file, `noun-phrases.txt`, that contains descriptions of a variety of noun phrases. If you wish, you may choose the version from the reading. However, we'd encourage you to think about your own variant.

```
((join article space adjective space noun) 500)
((join article space noun) 250)
(name 150)
((join name "'s " noun) 99)
("an infrequently-occurring noun phrase" 1)
```

- c. Verify that `(random-structure (string-append root "noun-phrases.txt"))` works correctly by running it a few times.
- d. Update the `noun-phrase` procedure to call `random-structure`.
- e. Verify that this updated version works.
- f. Verify that `sentence` still works with the updated `noun-phrase`.

Exercise 4: Checking Structure Files

As many of you discovered, it is sometimes difficult to manually ensure that a words file contains the right sum of frequencies. To automate checking, we included a procedure, `check-words-file`, that verifies that a words file has the appropriate format.

Write a procedure, `check-structure-file`, that does the same thing for a file of structural descriptions, such as the one described above.

You may find the following utility procedure helpful.

```
;;; Procedure:
;;;   structure?
;;; Parameters:
;;;   val, a Scheme value
;;; Purpose:
;;;   Determine whether lst is a valid structure.
;;; Produces:
;;;   ok?, a boolean
;;; Preconditions:
;;;   (none)
;;; Postconditions:
;;;   If ok? is #t, then val is either a string, a symbol, or
;;;   a list of structures.
```

```

;;; If ok? is #f, it is something else.
(define structure?
  (lambda (val)
    (or (null? val)
        (string? val)
        (symbol? val)
        (and (list? val)
              (structure? (car val))
              (structure? (cdr val))))))

```

Exercise 5: Starting Unification

We're now ready to see whether `random-structure` really serves the same purpose as `random-word`.

- Call `random-structure` with a few of the words files (such as `"nouns.txt"`). Verify that it seems to work correctly.
- Rewrite `generate-part-of-speech` to use `random-structure` instead of `random-word`. Try a few tests to ensure that it still works correctly.
- Verify that `(generate-part-of-speech 'noun-phrase)` now works correctly.
- Eliminate the `noun-phrase` procedure, since it is no longer necessary.
- Update any procedure that called `noun-phrase` to call `generate-part-of-speech` instead.
- Add code to `build-from-symbol` to create a `noun-phrase` when given the symbol `'noun-phrase`.
- Check that you can now build now phrases using `build`. For example,

- `(build 'noun-phrase)`
- `(build '(join noun-phrase space tverb space noun-phrase period))`

Exercise 6: Sentence Forms

If you've completed all of the steps of the previous exercise, the `sentence` procedure should now be able to build sentences using `build`. Since that's the case, we can now use the same technique for generating sentences that we used for generating noun phrases.

- Create a new file, `sentences.txt`, that contains a variety of sentence structures with accompanying frequencies. You should probably check the file with `structure?`. If you do not feel like developing your own list, here's one

```

((join (capitalize noun-phrase) space iverb period) 300)
((join (capitalize noun-phrase) space tverb space noun-phrase period) 500)
((join (capitalize exclamation) "!") 100)
((join (capitalize noun-phrase) " said, \" (capitalize exclamation) "!\"") 50)

```

- b. Verify that `(random-text (string-append path "sentences"))` works correctly.
- c. Add `'sentence` to the list of parts of speech.
- d. Verify that `(generate-part-of-speech 'sentence)` works correctly.
- e. Update `build-from-symbol` to incorporate sentences.
- f. Replace the `sentence` procedure with the following.

```
(define sentence
  (lambda ()
    (build 'sentence)))
```

- g. Verify that the change is successful.

Exercise 7: Indefinite Articles

Many of you have noted that there's a problem with the articles file: It always uses a. Instead of relying on a file, we might instead use a procedure.

- a. Write a procedure, `(add-indefinite-article str)`, that, given a string, adds an appropriate indefinite article.

```
> (add-indefinite-article "dog")
"a dog"
> (add-indefinite-article "cat")
"a cat"
> (add-indefinite-article "iguana")
"an iguana"
```

- b. Update `apply-procedure` to support `add-indefinite-article`. (You might use the symbol `'add-indefinite-article` or the symbol `'indefinite-article` to signify that procedure.)
- c. Verify that `(build '(indefinite-article "dog"))` produces the expected result.
- d. Verify that `(build '(indefinite-article "iguana"))` produces the expected result.
- e. Verify that `(build '(indefinite-article noun))` produces the expected result.
- f. Update your files (primarily `noun-phrases`) to use the new article strategy.

For Those With Extra Time

If you find that you've finished all of the work, you should start working on the lab report.

Copyright © 2007 Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/> or send a letter to Creative

Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.