Cognition, Language & Communication

Computer lab I & Assignment II

(19-9-2013)

Abstract

The goals of today's computer lab are (i) to introduce you to Processing, and (ii) to sharpen your intuitions on what certain simple grammar formalisms can and cannot do.

Processing is a user-friendly programming environment, based on the popular Java programming language, but providing some simple to use tools to make interactive, graphical applications that can easily be put on the web (applets). In recent years, Processing is increasingly used as a platform for webexperiments, and webexperiments have become a popular way of quickly and cheaply doing pilot experiments.

To learn how to work with Processing, you will first play with an existing program ("engineered" by someone else). Next, you will try to "reverse-engineer" it: to infer from a working system how it could have been designed. You can then compare your guesses with the actual program. Finally, you will take an existing program and, depending on your programming skills, adapt it a little or a lot and thus create a new (web)experiment.

1 Processing

Download Processing from `processing.org`. Save the zip-file on the Desktop and unpack it. Downloading might take a while, so continue with the next instructions.

SenGen In a browser, go to `http://www.illc.uva.nl/laco/clas/clc13/assignments/sengen/applet/` . This will start the sengen-applet, a very simple program that generates a sentence every time you click with the mouse or press a key.

Question 1 *Reverse-engineer (without looking at the source code) the steps that this computer program must go through. In particular, what are the grammar rules that the program is using?*

1. Generate an orange screen of 400x400 pixels.
2. ...
3.
4.
5.
6.
7.
8. When asked to generate a noun, randomly add 'cat' or 'dog' to the current sentence.
9.

Question 2 *Below is the source code for the actual program. Indicate with numbers where you find the steps you identified in question (1).*

```
// Global variables - variables that Processing knows about
// throughout the program, and their initial values

int X=5;                            // the x-coordinate of the next sentence to be shown
int Y=20;                           // the y-coordinate of the next sentence to be shown
String nextsentence;                // the actual next sentence to be shown
boolean something_happened = false; // a variable set to 'true' or 'false' specifying
                                    // whether you have just clicked the mouse or hit a key.

// setup() - this is the function that Processing calls when you start the program

void setup() {
  size(400, 400);
  background(192, 64, 0);
  stroke(255);
  nextsentence = new String("");
}

// mousePressed() and keyPressed() - these are the functions Processing calls
// when you click the mouse or hit a key.

void mousePressed() {
  something_happened=true;
}

void keyPressed() {
  something_happened=true;
}

// draw() - this is the function that Processing calls every few milliseconds.
// In our case, it only does anything when something_happened==true.

void draw() {
  if (something_happened) {
    something_happened = false;
    generateS();
    text(nextsentence,X,Y);
    X=5;
    Y+=20;
    if (Y>380) {Y=20; background(192, 64, 0);}
    nextsentence = new String("");
  }
}

// generateX() - these are the function for generating noun phrases, determiners, nouns,
// (in)transitive or intentional verbs, verb phrases and sentences.

void generateNP() {
  int r = int(random(5));
  if (r==0) {   nextsentence = nextsentence.concat("John "); }
  if (r==1) {   nextsentence = nextsentence.concat("Mary "); }
  if (r==2) {   nextsentence = nextsentence.concat("Pete "); }
  if (r==3) {   nextsentence = nextsentence.concat("Lisa "); }
  if (r==4) {
    generateDET();
    generateN();
  }
}

void generateDET() {
  int r = int(random(2));
  if (r==0) {   nextsentence = nextsentence.concat("a "); }
  if (r==1) {   nextsentence = nextsentence.concat("the "); }
}

void generateN() {
  int r = int(random(2));
  if (r==0) {   nextsentence = nextsentence.concat("cat "); }
  if (r==1) {   nextsentence = nextsentence.concat("dog "); }
}

void generateTV() {
  int r = int(random(2));
  if (r==0) {   nextsentence = nextsentence.concat("loves "); }
  if (r==1) {   nextsentence = nextsentence.concat("hates "); }
}

void generateIV() {
  int r = int(random(2));
  if (r==0) {   nextsentence = nextsentence.concat("walks "); }
  if (r==1) {   nextsentence = nextsentence.concat("sits "); }
}

void generateEV() {
  int r = int(random(2));
  if (r==0) {   nextsentence = nextsentence.concat("says "); }
  if (r==1) {   nextsentence = nextsentence.concat("thinks "); }
}

void generateVP() {
  int r = int(random(3));
  if (r==0) {   generateIV(); }
  if (r==1) {   generateTV(); generateNP(); }
  if (r==2) {   generateEV(); generateS(); }
}

void generateS() {
  generateNP();
  generateVP();
}
```

If you don't know any programming language related to java (like C, C++, matlab etc), pay some special attention to the following features (and ask fellow students, the lecturer or assistant to make sure you understand it):

- At the beginning of the program, *variables* are "declared" and initialized. E.g., the program must specify that `x` is a variable that stands for a 'whole' number (integer) like 1,2,5, 1000.
- The rest of the program consists of *functions* that at some point are executed (they are "called"). In Processing, some functions are automatically called, e.g. `setup()` is called when the program starts. For other functions, the program has to contain explicit instructions for when they are called (inside other functions). Note that functions are defined in a format `void FUNCTIONNAME (ARGUMENTS) {...}`. Between the curly brackets we then see particular instructions.
- Note that instructions always end with a semicolon (;). Instructions might involve, e.g., setting a variable to a specific value as in `x=5;` or calling a function, as in `generateS();`.
- Further note that many instructions are only carried out if a particular condition is met. `if (CONDITION) {...}` means "carry out the instructions between the curly brackets if the specified CONDITION is met.

Start programming Start the `processing.exe` program that you have downloaded and unpacked. In the menu under File/Examples you can find many examples. Load 1 or 2 of those (for instance, the first two from the Learn Processing book), and run them by clicking on the play button on the top left of the Processing screen.

Question 3 *Adapt the program seen in question 2 to generate sentences from the languages* $(ab)^n$ *and* a^nb^n*. In order to this, write down the grammar rules that you need to generate those language, and test them in the computer program.*

2 Homework assignment (due Tuesday 24/9, 17h)

Question 4 *Give the finite-state automaton that can generates* a^nb^m*, with* $n \geq 1$ *and* $m \geq 1$*. Indicate where in the FSA you would need to access a counter (technically a* push-down stack*) to change the automaton such that it generates* a^nb^n*.*

Question 5 *Give a context-free grammar that can generate* a^nb^n*. Hint: you need a* recursive *rule, where the symbol from the left-hand side also appears on the right-hand side.*

Question 6 *Adapt the sengen program in two different ways. One version should generate* a^nb^n *using a counting-strategy, the other by applying a recursive function. Submit only the lines of code that are different from the original program.*

Important: submit your answers in *pdf* format through the *Blackboard* site.