Lecture 11: Languages and Recursion

Dr John Levine

CS103 Machines, Languages and Computation
November 2nd 2015

The First Half of the Class

- The first half of the class was about the fact that all computers are equivalent and that non-computable functions exist:

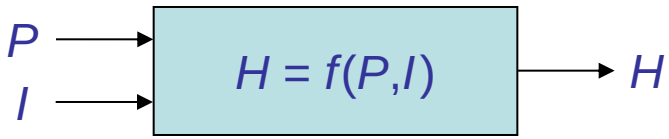
  If P is the set of all computer programs and F is the set of all functions f: $n \rightarrow m$ such that n,m are members of the set of natural numbers N, then $|F| > |P|$ and therefore there are some functions in F for which no computer program can exist.

- We now know why this is: P is infinite but countable, and F is infinite and uncountable.

University of Strathclyde Science

The Halting Problem

- One of the most famous non-computable functions is called “The Halting Problem”
- Given a computer program, *P*, and a set of inputs to that program, *I*, decide (in finite time) whether or not the program halts or runs forever:



- *H* = 1 means the program halts, *H* = 0 means that it runs forever
- It can be shown that no program can exist for *f*(*P*,*I*)

University of Strathclyde Science

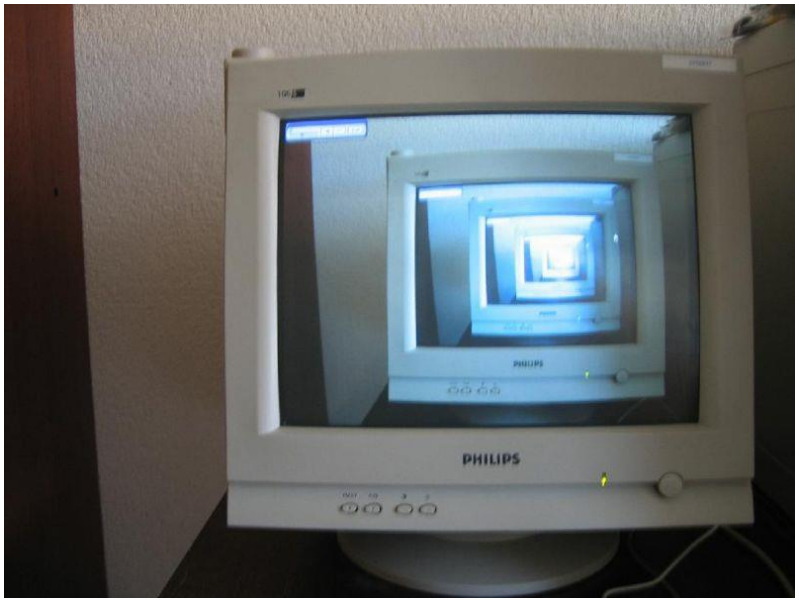
The Second Half of the Class

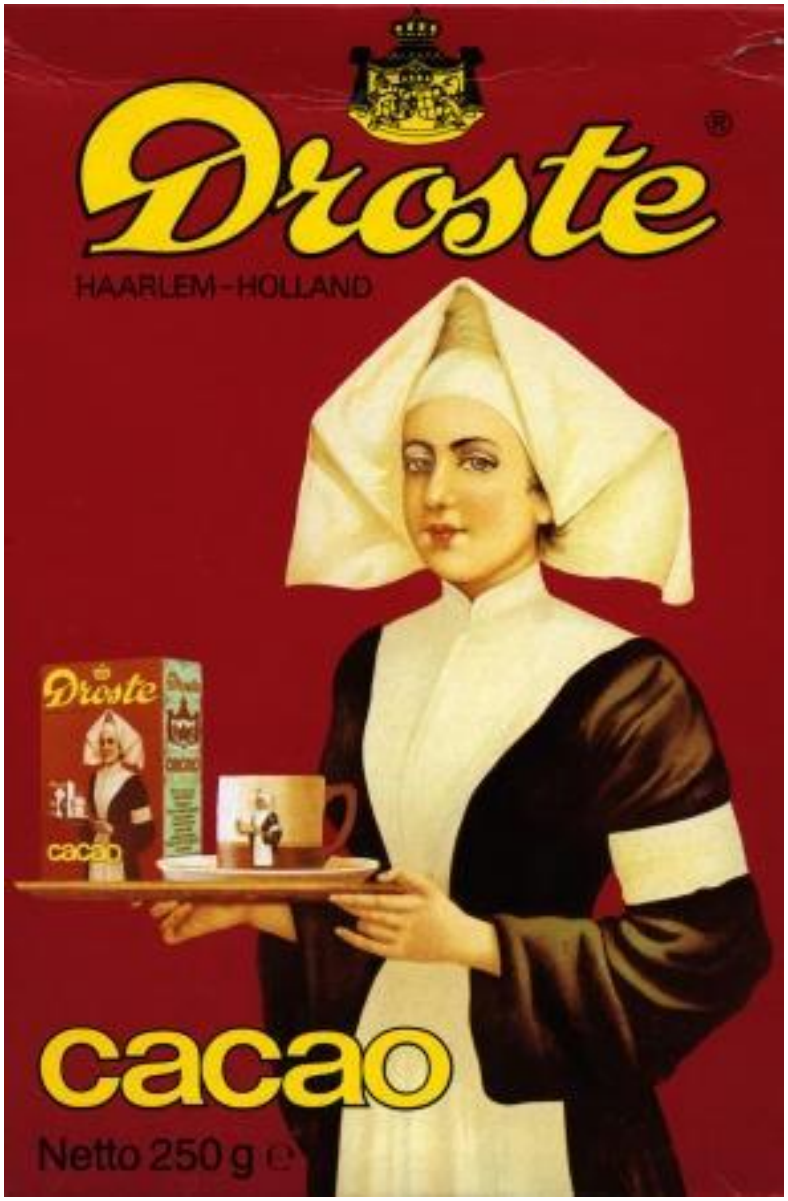
- Week 7: Recursion and Languages: how to use recursion to allow for potentially unbounded sentences.
- Week 8: Recursion and Strings: how to specify an infinite set of strings using a finite set of rules.
- Week 9: Recursion and Logical Definitions: what does it mean to be someone's ancestor?
- Week 10: The λ-calculus: how can we create and use computable functions using only symbols?
- Week 11: Recursion and Function Definitions: how can we use recursion to create functions?

University of Strathclyde Science

What is Recursion?

- Recursion is a method of defining structures in which the structure being defined may be used within its own definition.
- The term is also used more generally to describe a process of repeating objects in a self-similar way.
- Recursion leads to “nested” structures. Sometimes the nature of the nesting is clear, but often we have to look hard to find it and encode the recursive process.
- Recursion can be found in many places: in languages, logical definitions, data structures, programs, …


PHILIPS


Droste
HAARLEM - HOLLAND
Droste
cacao
cacao
Netto 250 g

University of Strathclyde Science

Recursion by Example

- How do we define the members of the set of natural numbers, $N = \{0, 1, 2, 3, \ldots\}$?
- We can do it by recursion:

  0 is a member of N
  If n is a member of N, $n+1$ is a member of N

- We need *both* statements: either one on its own is not enough to complete the definition.
- Notice the very strong similarity between this definition and proof by induction, as introduced in Lecture 4.

University of Strathclyde Science

What are Languages?

- Languages are devices used to transfer information between people, and between people and machines
- Statements in a language consist of sequences of symbols, such as “John ate a frog” or “let x = sqrt(k)+1”
- Languages have *structure*: we can’t just produce the symbols in a random order
- We can *recognise* legal and illegal sentences:

John ate a big frog. ☺
big ate John frog a. ☹
let x = sqrt(+) * if(k) ☹

University of Strathclyde Science

Syntax and Semantics

- Languages have *structure*: we can't just produce the symbols in a random order
- The structure of a language is called *syntax* and can be specified using a *grammar*
- The meaning of an expression of language is called the *semantics* of the expression (e.g. "let x=3" is an instruction to set the value of variable x to be 3)
- For the moment, let's concentrate on the structure of language and look at some of the repeating structures we can find…

Types of Languages

- Natural languages which have evolved for human communication: English, French, Italian, Chinese.
- Invented languages for communication: Esperanto, Loglan, Klingon.
- Computer languages: Java, Lisp, C, Algol, Fortran, Basic, Cobol, Pascal, ML, Haskell, Perl, Python, …
- Formal languages: mathematical logic
- Invented languages, e.g. $L = a^n b^n c^n$ – used to find out what languages we can describe with our grammars

University of Strathclyde Science

Context-Free Grammars

- Most computer languages can be specified used context-free grammars
- Rules are of the form:

  Symbol → list of Symbols and Terminal Symbols

  e.g.

  Exp → “if” Cond “then” Exp
  Exp → “print” Var
  Cond → Var “=” Value
  Var → “a” | “b” | “c” | … | “y” | “z” (| means “or”)
  Value → “0” | “1”

University of Strathclyde Science

Coping with Repeating Structure

Exp → “if” Cond “then” Exp
Exp → “print” Var
Cond → Var “=” Value
Var → “x” | “y” | “z”
Value → “0” | “1”

- This grammar can generate repeating structures:

```
if a=0 then
        if b=1 then
                if c=0 then
                        if d=0 then …
```

Recursion

- Recursion occurs when you define something in terms of itself, e.g. the rule “Exp → “if” Cond “then” Exp”
- From GEB: “Hofstadter’s Law: it always takes longer than you expect, even when you take into account Hofstadter’s law.”
- Python factorial function:

```
def factorial (n):
    if n == 0:
        return 1
    else:
        return n * factorial (n - 1)
```

University of Strathclyde Science

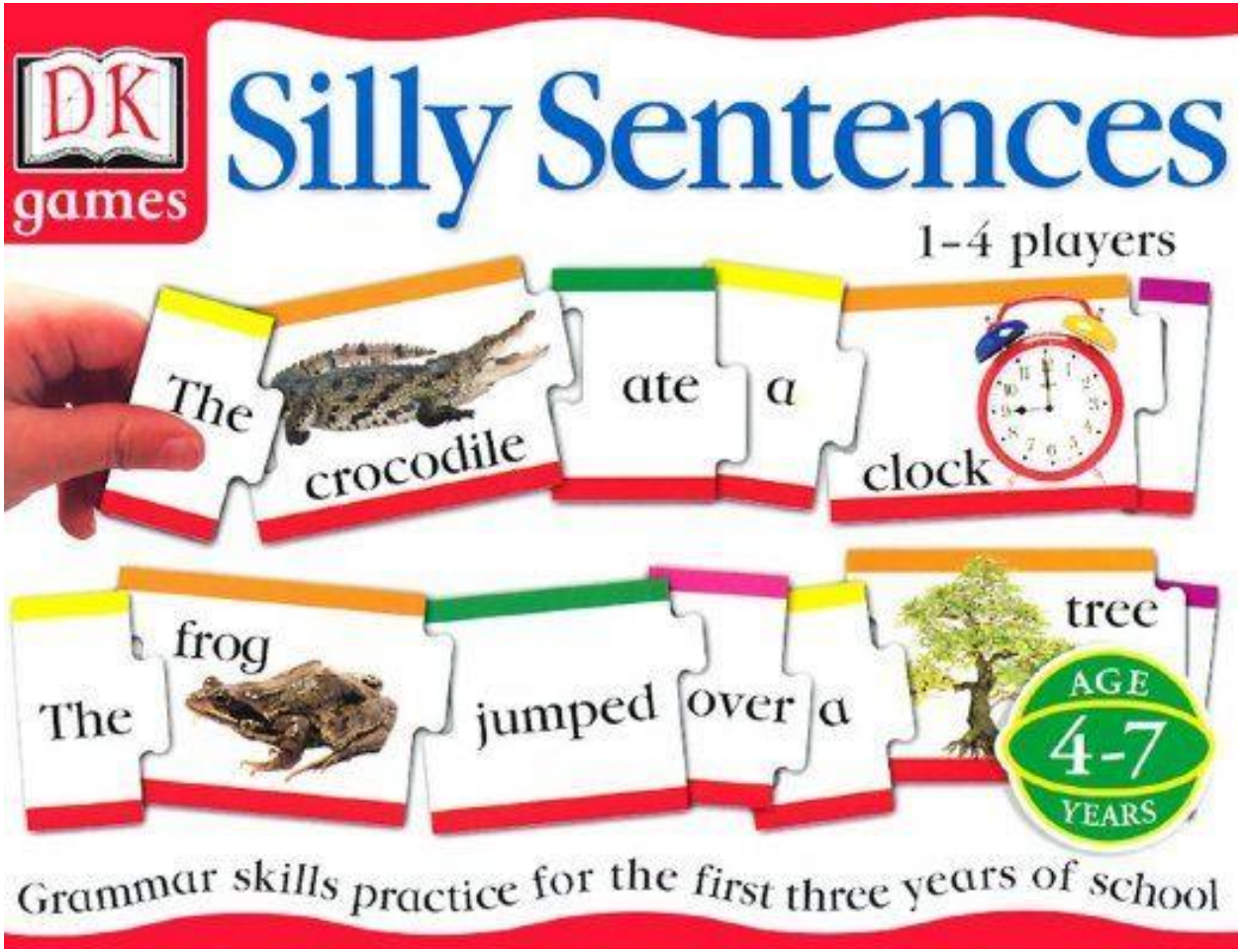
Recursion

- Recursion occurs when you define something in terms of itself, e.g. the rule "Exp → "if" Cond "then" Exp"
- From GEB: "Hofstadter's Law: it always takes longer than you expect, even when you take into account Hofstadter's law."
- Python factorial function:

```
def factorial (n):
    if n == 0:
        return 1
    else:
        return n * factorial (n - 1)
```

University of Strathclyde
Science

Silly Sentences



University of Strathclyde Science

Silly Sentences

- This is a game by Dorling Kindersley containing jigsaw pieces with words on them.
- On the box, it says “Grammar skills practice for the first three years at school”.
- You can assemble the pieces to create random (but grammatical) sentences, such as:

  The yellow crocodile ate a rubber clock.

  The old fat hairy princess snores.

  Brian ate a big purple frog.

University of Strathclyde Science

Silly Sentences

- Can we write a program to generate legal sentences?
- We can make use of production rules, similar to the rules of the MIU system:

  Sentence → NounPhrase Verb1
  NounPhrase → Name
  Name → “Brian”
  Name → “Beryl”
  Verb1 → “snores”
  Verb1 → “smells”

- Words in “…” are called *terminal symbols*

University of Strathclyde Science

Sentence Generation Algorithm

- Start with “Sentence”.
- Find the left-most non-terminal symbol.
- Expand it using a random rule that applies.
- Repeat from 2 until all the symbols are terminals.

“Sentence” →“NounPhrase Verb1”
“NounPhrase Verb1” → “Name Verb1”
“Name Verb1” → “Brian Verb1”
“Brian Verb1” → “Brian snores”

University of Strathclyde Science

A Larger Grammar

Sentence → NounPhrase Verb2 NounPhrase
NounPhrase → Name
NounPhrase → Article NounGroup
NounGroup → Noun
NounGroup → Adjective Noun

Name → “Brian” | “Beryl”
Verb2 → “ate” | “fed” | “chased” | “kissed”
Article → “a” | “the”
Adjective → “big” | “purple” | “ugly” | “fat” | “eccentric”
Noun → “frog” | “dog” | “clock” | “monster” | “princess”

University of Strathclyde Science

Derivation of a Sentence

- We can derive sentences with grammars just like we did with the MIU-system:

  Sentence
  NounPhrase Verb2 NounPhrase
  Name Verb2 NounPhrase
  “Brian” Verb2 NounPhrase
  “Brian” “ate” NounPhrase
  “Brian” “ate” Article NounGroup
  “Brian” “ate” “a” NounGroup
  “Brian” “ate” “a” Adjective Noun
  “Brian” “ate” “a” “big” Noun
  “Brian” “ate” “a” “big” “frog” ■

University of Strathclyde Science

Assignment 6

- Generate some random sentences using the grammar.
- The grammar can generate “Brian ate a big frog” or “Brian ate a purple frog”.
- But I want it to generate “Brian ate a big purple frog”.
- Or even “Brian ate a big fat ugly purple eccentric frog”.
- Can you modify my grammar so it can generate noun phrases containing any number of adjectives, without increasing the number of rules?
- Reading: Chapter 5 of Gödel, Escher, Bach.