Context-sensitive languages

Informatics 2A: Lecture 28

John Longley

School of Informatics
University of Edinburgh
jrl@inf.ed.ac.uk

24 November, 2011

Non-context-free languages

We saw in Lecture 8 that the pumping lemma can be used to show a language isn't regular.

There's also a context-free version of this lemma, which can be used to show that a language isn't even context-free:

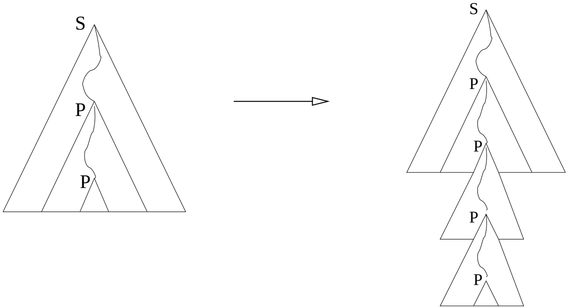
Pumping Lemma for context-free languages. Suppose L is a context-free language. Then L has the following property.

> *(P) There exists $k \geq 0$ such that any $z \in L$ with $\mid z \mid \geq k$ can be broken up into five substrings, $z = uvwxy$, such that $vx \neq \epsilon$ and $uv^iwx^iy \in L$ for all $i > 0$.*

Context-free pumping lemma: the idea

In the regular case, the key point is that any sufficiently long string will visit the same state twice.

In the context-free case, we note that any sufficiently large syntax tree will have a downward path that visits the same non-terminal twice. We can then 'pump in' extra copies of the relevant subtree and remain within the language:



Standard example

The language $L = \{a^n b^n c^n \mid n \geq 0\}$ isn't context-free!

To see this, suppose L had a CFG with m non-terminals, and take k so large that the syntax tree for any string of length $\geq k$ must contain a path of length $> m$.

Then for any $z = a^n b^n c^n$ where $\mid z \mid \geq k$, we can do pumping. There must be some splitting $z = uvwxy$ such that $uv^i wx^i y \in L$ for all i. However ...

- If v contains letters of more than one kind (e.g. $v = aab$), then $uv^2 wx^2 y \notin L$.
- Similarly if x contains letters of more than one kind.
- So there must be some letter $d \in \{a, b, c\}$ that doesn't appear in either v or x. So $uv^2 wx^2 y$ contains just n occurrences of d, but more of other stuff. So $uv^2 wx^2 y \notin L$.

More general grammars

If $\{a^n b^n c^n \mid n \geq 0\}$ isn't context-free, what is it?

In the definition of CFGs, recall that Σ was the set of terminals, N was the set of nonterminals. Productions were of the form

$$X \;\rightarrow\; \beta \qquad (X \in N,\; \beta \in (N \cup \Sigma)^*)$$

We can generalize this to allow productions

$$\alpha \;\rightarrow\; \beta \qquad (\alpha, \beta \in (N \cup \Sigma)^*)$$

It's also of interest to consider such rules with the restriction that $\mathit{length}(\alpha) \leq \mathit{length}(\beta)$ (motivation to be explained later).

- With this restriction, we get context-sensitive grammars.
- Without the length restriction, we get general or unrestricted grammars.

These are the top two levels in the Chomsky hierarchy.

Context-sensitive grammars: an example

Consider the following CSG (with start symbol S).

$$
\begin{array}{rcl@{\qquad}rcl}
S & \rightarrow & aSBC & bB & \rightarrow & bb \\
S & \rightarrow & aBC & bC & \rightarrow & bc \\
CB & \rightarrow & BC & cC & \rightarrow & cc \\
aB & \rightarrow & ab & & &
\end{array}
$$

Example derivation:

$$
\begin{array}{ll}
S & \Rightarrow aSBC \Rightarrow aaBCBC \Rightarrow aaBBCC \\
 & \Rightarrow aabBCC \Rightarrow aabbCC \Rightarrow aabbcC \\
 & \Rightarrow aabbcc
\end{array}
$$

Exercise: Convince yourself that this grammar generates exactly the strings $a^n b^n c^n$ where $n > 0$.

(N.B. With CSGs, need to think in terms of derivations, not syntax trees.)

Why 'context-sensitive'?

A common idiom in CSGs is to have rules of the form

$$\alpha X \gamma \;\rightarrow\; \alpha \beta \gamma$$

This effectively says "$X \rightarrow \beta$ can be applied in the context $\alpha[-]\gamma$".

So the ways we can expand X can be sensitive to the context in which the X occurs (contrasts with context-free).

Minor wrinkle: Length restriction on CSG disallows rules with right-hand side ϵ.

Not a serious problem, because e.g. in any context-free grammar, ϵ-rules can be removed by converting to Chomsky Normal Form. Except that the resulting language can't contain the string ϵ.

To remedy this, we make an exception to the length restriction to allow the special rule $S \rightarrow \epsilon$ (where S is the start symbol).

Context-sensitivity in programming languages

Some aspects of typical programming languages can't be captured by context-free grammars, e.g.

- Typing rules
- Scoping rules (e.g. variables can only be used in contexts where they have been 'declared')
- Access constraints (e.g. use of `public` vs. `private` methods in Java).

The usual approach is to give a CFG that's a bit 'too generous', and then separately describe these additional rules.
(E.g. typechecking done as a separate stage after parsing.)

In principle, though, all the above features fall within what can be captured by context-sensitive grammars. In fact, no programming language known to humankind contains anything that can't.

Scoping constraints aren't context-free

Consider the simple language L_1 given by

$$S \;\rightarrow\; \epsilon \;\mid\; \texttt{declare}\ v; S \;\mid\; \texttt{use}\ v; S$$

where v stands for a lexical class of variables. Let L_2 be the language consisting of strings of L_1 in which variables must be declared before use.

Assuming there are infinitely many possible variables, it's a little exercise to show L_2 is not context-free, but is context-sensitive.

(If there are just n possible variables, we could in theory give a CFG for L_2 with around 2^n nonterminals — but that's obviously silly. . .)

Context-sensitivity in natural language

Example of a NL feature that it's natural to model in a context-sensitive way: a versus an in English.

a banana	an apple
a large apple	an exceptionally large banana

Over-simplifying a bit: a before consonants, an before vowels.

Context-sensitive rules (schematic only):

DET [c-word] $\rightarrow$ **a** [c-word]
DET [v-word] $\rightarrow$ **an** [v-word]

In theory, we could use a context-free grammar, at the cost of some silly duplication in the rules:

$$\begin{array}{rcll} NP & \rightarrow & \textbf{a}\ NP1^c & \\ NP1^c & \rightarrow & N^c & \mid\ AP^c\ NP1 \\ AP^c & \rightarrow & A^c & \mid\ Adv^c\ AP \end{array} \qquad \begin{array}{rcll} NP & \rightarrow & \textbf{an}\ NP1^v & \\ NP1^v & \rightarrow & N^v & \mid\ AP^v\ NP1 \\ AP^v & \rightarrow & A^v & \mid\ Adv^v\ AP \end{array}$$

But the context-sensitive treatment seems more natural.

Agreement phenomena

In English, verbs agree in number (i.e. singular/plural distinction) with their subjects, even when they are widely separated:

The man wants a pear.
The men that Fred talked to want a pear.
The man that Alistair remembered that Harold had said that Sam had seen Fred talk to wants a pear.

Other languages have far more agreement phenomena, e.g. in French, adjectives agree in number and gender with their head noun; verbs agree in number and person with their subjects, etc.

All this is at least broadly reminiscent of typing constraints in PLs:

```
int i; if x!=5 then return x else i=1
boolean i; if x!=5 then return x else i=false
```

Modelling agreement

In principle, English verb-subject agreement can be captured by a CFG, because the Number attribute has only two values, Singular and Plural.

$$
\begin{array}{rcl}
S & \to & NP^s\ VP^s \\
NP^s & \to & N^s \quad | \quad AP\ NP^s \\
VP^s & \to & V^s\ NP
\end{array}
\qquad
\begin{array}{rcl}
S & \to & NP^p\ VP^p \\
NP^p & \to & N^p \quad | \quad AP\ NP^p \\
VP^p & \to & V^p\ NP \ldots
\end{array}
$$

But for a more economical description, it's convenient to use some formalism that goes a bit beyond the power of CFGs (e.g. linear indexed grammars).

Essentially context-sensitive phenomena

As we've seen, many 'naturally context-sensitive' aspects of NL can in theory be captured by context-free grammars, albeit in a rather silly way. . .

But are there features of NLs that can't be captured by CFGs?

It appears that there are! E.g. Dutch and Swiss German allow unbounded crossing dependencies between verbs and their objects.

Crossing dependencies in Swiss German

In Swiss German, some verbs (e.g. *let*, *paint*) take an object in accusative form, while others (e.g. *help*) take it in dative form.

Swiss-German

...das mer	d'chind	em Hans	es huus	lönd	hälfe	aastriiche
...that we	the children	Hans	the house	let	help	paint
	NP-ACC	NP-DAT	NP-ACC	V-ACC	V-DAT	V-ACC

...that we let the children help Hans paint the house

Abstracting out the key feature here, we see that the same sequence over $\{a, d\}$ (in this case *ada*) must 'appear twice'.

But $\{ss \mid s \in \{a, d\}^*\}$ isn't context-free (interesting exercise). Hence neither is Swiss German!

Summary

- Context-sensitive languages are a big step up from context-free languages in terms of their power and generality.
- Programming languages contain non-context-free features (typing, scoping etc.), but all these fall comfortably within the realm of context-sensitive languages.
- Natural languages have features that can't be captured conveniently (or at all) by context-free grammars. However, it appears that NLs are only mildly context-sensitive — they only exploit the low end of the power offered by CSGs.
- Next time: what kinds of machines are needed to recognize context-sensitive languages?