Lecture 14. Push-Down Storage Automata and Context-Free Grammars

Reading:
Chapter 18, "Pushdown Automata, Context Free Grammars and Languages" of PtMW, pp. 485-504. See also **References**.

1. Pushdown automata

A *pushdown automaton*, or *pda*, is essentially a finite state automaton augmented with an auxiliary tape on which it can read, write, and erase symbols. Its transitions from state to state can depend not only on what state it is in and what it sees on the input tape but also on what it sees on the auxiliary state, and its actions can include not only change of state but also operations on the auxiliary tape.

The auxiliary tape works as a *pushdown store*, "last in, first out", like a stack of plates in some cafeterias. You can't 'see' below the top item on the stack without first removing (erasing) that top item.

States: as in a fsa, a pda has a finite number of states, including a designated initial state and a set of "final" or "accepting" states.

Transitions: $(q_i, a, A) \rightarrow (q_j, \gamma)$, where q_i, q_j are states, a is a symbol of the input alphabet, A is either the empty string e or a symbol of the stack alphabet (which need not be the same as the input alphabet), and γ is a string of stack symbols, possibly the empty string.

Interpretation: When in state q_i, reading a on the input tape, and reading A at the top of the stack, go to state q_j, and *replace A by the string γ.* The elements of the string γ go onto the stack one at a time, so the last symbol of γ ends up as the top symbol on the stack. In case γ is the empty string e, the effect is to erase γ, that is to remove it ("pop" it) from the stack. ("Push" down and "pop" up.)

Examples of A and of γ, and their effects. First suppose that A is a particular symbol (and let's just call it A.) If γ is BC, the result is to remove A, and add B on the top of the stack, and then add C on top of B. If γ is e, the result is to remove A, and the new top symbol will be whatever was just below A, if anything. If γ is AB, the result is to add a B on top of the A that was there. If γ is A, then the stack will be unchanged.

If A is e, that means that the transition does not depend on what's on the stack; it does *not* mean that the stack must be empty. (We don't have any special way of indicating that the stack is empty, though you can design an individual machine to start by putting some particular symbol on the stack, and to be sensitive to when it sees that symbol again as a way of knowing its stack is empty except for that one symbol.) If A is e and γ is also e, the stack stays unchanged. Otherwise γ is added to the stack.

Initial configuration: the stack is assumed to be empty at the beginning of a computation, with the pda in its initial state and the reading head looking at the first (leftmost) symbol on the input tape. A pda accepts an input tape if the computation leads to a situation in which all three of the following are simultaneously true:

(i) the entire input has been read;
(ii) the pda is in a final (accepting) state;
(iii) the stack is empty.

.

Example 1. A deterministic pda for the classic context-free and non-finite-state language $\{a^n b^n \mid n \geq 0\}$.

(on blackboard. See p. 486.)

Example 2: A pda for another classic context-free and non-finite-state language: $\{xx^R \mid x \in \{a, b\}^* \}$

(on blackboard. See pp 487-8.)

This machine is non-deterministic. Acceptance for non-deterministic vs deterministic machines is defined just as for finite state automata: a non-deterministic pda accepts a string iff there is *some* computation which leads to acceptance.

The natural question: are deterministic and non-deterministic pda's equivalent? Answer: NO. The proof is not simple and it isn't included in PtMW. Intuitively, in the given case, we can see that there is no deterministic way for a pda to know when it is at the middle of the string in Example 2, and it needs to know that in order to know when to stop adding symbols and start matching and erasing.

By contrast, the language $\{xcx^R \mid x \in \{a, b\}^* \}$, in which the center is marked by a distinctive symbol c, *is* acceptable by a deterministic pda.

Example 3: A pda for the language of exercise 1c, p. 503: $L_3 = \{x \mid x \in \{a, b\}^*$ and x contains twice as many b's as a's $\}$

Note: both example 3 and example 1 can be done with a 1-symbol pushdown alphabet; these are sometimes called "counter machines", since they are not requiring the full power of a pushdown store with arbitrary alphabet. The mirror-image language, by contrast, requires as many distinct symbols in the pushdown alphabet as there are in the input alphabet.

2. Context-free grammars and languages

Non-deterministic pda's accept exactly the languages generated by context-free (Type 2) grammars. The regular languages are a proper subset of the context-free languages.

The proof of equivalence is fairly long and complex. PtMW give just one part: an algorithm for constructing from any context-free grammar an equivalent non-deterministic pda (but not the formal proof that the constructed automaton is indeed equivalent.)

[blackboard. See pp. 490-1]

The automaton *M* just constructed works as a simple parser for the grammar *G. M* starts by loading the symbol *S* onto its stack and then simulating a derivation of the string it finds on its input tape by manipulations that correspond to the rules of *G. M* in general has to be non-deterministic, since the grammar may have more than one way of rewriting a given non-terminal symbol *A*.

3. Pumping theorem for context-free languages

The pumping theorem for cfl's is similar in form to that for fal's. It is used primarily for showing that a given language is not context free.

Key facts used in the theorem: a derivation by a cfg can be naturally associated with a parse tree, and the maximum width of any such parse tree is constrained by its height.

For discussion and pictures, see pp 492-494.

Theorem 18.1: If L is an infinite context free language, then there is some constant K such that any string w in L longer than K can be factored into substrings $w = uvxyz$ such that v and y are not both empty and $uv^ixy^iz \in L$ for all $i \geq 0$.

Note that this pumping theorem is a conditional but not a biconditional.

Examples: Show that $a^nb^nc^n$ is not context-free.
Show that xx is not context-free (assuming an alphabet of more than one letter).

Interesting example: The language of first-order logic, with and without constraints on vacuous quantifiers or unbound variables. (See (Marsh and Partee 1984)

4. Closure properties of context free languages.

Recall that the finite-state languages have very nice closure properties: they are closed under union, concatenation, Kleene star, complementation, and intersection. Also recall that the equivalence of deterministic and non-deterministic fsa was crucial in some of the proofs of those closure properties: the union algorithm, for instance, in general gave a non-deterministic machine as output, but the complement algorithm required a deterministic machine as input, so constructing an intersection machine required intermediate steps of converting a non-deterministic fsa into a deterministic fsa. So given that we already know that deterministic and non-deterministic PDAs are not equivalent classes, it may not be surprising that we don't have as full an array of closure properties for context-free languages (which correspond to non-deterministic PDAs.)

Constructions for the operations under which the class of CF languages *is* closed can be found in PtMW, pp 495-497. In this case the constructions involve showing how to make a CFG for the resulting language, given CFGs for the original languages.

The class of CF languages is closed under:
- Union: Key to construction: a new pair of rules $S \rightarrow S_1$ and $S \rightarrow S_2$.
- Concatenation: Key to construction: a new rule $S \rightarrow S_1 S_2$.
- Kleene star: Key to construction: a new pair of rules $S \rightarrow e$ and $S \rightarrow S S_1$.

It is *not* closed under:
- Intersection

  Proof: Consider the intersection of the two CF languages $\{a^i b^i c^j \mid i, j \geq 0\}$ and $\{a^k b^l c^l \mid k, l \geq 0\}$. The intersection language is $\{a^n b^n c^n \mid n \geq 0\}$, which can be shown to be non-context-free using the pumping lemma.

- Complementation (this fact follows from the previous one.)

But it *is* closed under:
- Intersection with a regular language.

This last fact can be very useful in proofs that a certain language is not context-free. For instance, the language $L = \{ x \in \{a, b, c\}^* \mid x$ contains equal numbers of a's, b's, and c's $\}$ is not context-free, but the pumping theorem can't be applied to it directly. However, we know that if L is CF, its intersection with any regular language must be CF. Let's intersect it with the regular language a*b*c*. The result is the language $\{a^n b^n c^n \mid n \geq 0\}$, which we already know is non-CF. Hence L is also not CF.

5. Decidability properties of context free languages.

CF languages similarly differ from finite-state languages in that various questions that are decidable for fs languages (see Lectures 11-12, Section 6) are not in general decidable for CF languages.

Is there an algorithm for determining … ?
- the membership question: Given an arbitrary cfg G and an arbitrary string x, is x generated by G? Yes, that can always be answered.
- the emptiness question: Yes. Key observation: If G generates any terminal string at all, it generates some terminal string with a tree which has no non-terminals repeated along any path. This makes the number of possible derivations that has to be searched finite.
- whether $L(G) = V_T^*$: NO
- whether the complement of L(G), i.e. $V_T^* - L(G)$, is empty, infinite, regular, or context-free. NO

Other undecidable questions for CFGs: Given two arbitrary CFGs G_1 and G_2, there is no algorithm for determining:

- whether $L(G_1) \subseteq L(G_2)$;
- whether $L(G_1) = L(G_2)$;
- whether $L(G_1) \cap L(G_2) = \varnothing$;
- whether $L(G_1) \cap L(G_2)$ is infinite, regular, or context-free.

Recall that the absence of an algorithm for the general case doesn't mean that we can't sometimes answer these questions in specific cases. This is easy to remember if you remember that the regular languages are all context-free languages, and we know how to answer all of these questions for regular languages. In particular cases we can also answer various of them for context-free languages that are not regular languages; but not always, and not with a general algorithm.

6. Are natural languages context-free?

When BHP was a graduate student, it was considered that Chomsky had clearly established that natural languages are not CF. Gil Harman was the first to argue that one could mimic transformational grammars with large but systematic CFGs in (Harman 1963). His construction was not entirely convincing, as I recall (I need to check this), but it opened up the idea that if one were willing to consider a very big CFG, one could do at least a great deal of what might be more "elegantly" done with transformations. A more satisfying and more influential approach was developed first by Gerald Gazdar (Gazdar 1982, Gazdar 1983a, Gazdar 1983b) and then in further collaboration with Geoffrey Pullum (Pullum and Gazdar 1982), Ivan Sag, and Ewan Klein, who collectively developed GPSG (Gazdar et al. 1985). They not only showed how to do with CF rules everything that transformations were being used to do, but got around the "elegance" problem by providing a finite set of 'meta-rules' that would specify the large set of needed CF rules.

Then Stuart Shieber found a construction in Swiss German which was not amenable to GPSG treatment (Shieber 1985) (see section below), which convinced everyone that natural languages are after all not context-free. Much work since then has focused on exploring "mildly context-sensitive" languages; Aravind Joshi is one of the leaders in this area (Joshi 1985, Joshi et al. 1989).

Geoffrey Pullum has written good overviews of the state of the controversies both before and after Shieber's discovery (Pullum 1984, Pullum 1986), including careful (and sharp) critiques of fallacious arguments that can be found in the literature.

============

A summary of Shieber's argument, from a University of Georgia Master's Thesis on GPSG, found at http://www.ifi.unizh.ch/CL/volk/SyntaxVorl/GPSG.thesis.html :

As intriguing as GPSG might seem on first sight some caveats are appropriate. From the very start GPSG worked under the assumption that natural languages are CF. The theory was thus designed to accommodate CF languages and as we have seen is even restricted to ID/LP grammars which form a subset of CF grammars. Recently, evidence has been brought forth that suggests that human languages exceed CF language restrictions. Shieber (1985), for example, claims that Swiss German is non-CF. His argument, informally, goes as follows: Swiss German subordinate clauses can have a structure where all verbs follow all NPs (which is true, by the way, for High German subordinate clauses as well). Among such sentences there are some with more than one accusative and dative NP. The number of verbs that subcategorize for these NPs must be equal to the number of these NPs. That is, formally we get a language like:

```
(12) L = w a^m b^n x c^m d^n y
```

'a' and 'b' can be seen as the NPs in dative or accusative respectively and 'c' and 'd' are the verbs that subcategorize for these NPs. L is a classical example of a non-CF language if 'm' and 'n' range over N, the set of integers. This is the point where Shieber's proof breaks down. Although sentences in natural languages are theoretically of infinite length, in practice they never are. This means that we can find an integer i as an upper bound for 'm' and 'n' and then L is CF again.

=============

7. Issues concerning the difficulty of processing nested and serial dependencies.

There are some constructions in Dutch that are similar to Shieber's Swiss German examples, except that the verbs don't vary in what case noun they take, so that from a syntactic point of view one just gets $a^n b^n$, where you can't "see" whether the embedding is nested (mirror-image) or serial. But supposing that wanting to get the semantics right were to force us to associate the right NP with the right verb, then Dutch, like Swiss German, has something like an "xx" language, where standard German has "x x^R". The situation looks like this.

Dutch:

(1)	dat	de	inspectie	de	bewaker	de	gevangenen	zag	helpen	ontsnappen
	that	the	inspectors	the	guard	the	prisoners	saw	help	escape

‘that the inspectors saw the guard help the prisoners escape’

Pattern: N1 N2 N3 V1 V2 V3

German:

(2) (help)

Pattern: N1 N2 N3 V3 V2 V1

Emmon and some Dutch and German colleagues did some experiments at the Max Planck Institute (Bach et al. 1986) to try to find out which was easier to process, Dutch or German. I don't remember the results. But it's clear, as often noted, that both constructions are extremely difficult to process when there are more than two or three layers of embedding. For this reason there continue to be debates about the significance of the evidence that natural languages are non-CF, and even that they are non-finite-state. See, for instance, Van Noord's page http://odur.let.rug.nl/~vannoord/alp/proposal/node10.html titled "Arguments for Finite-State Language Processing", which includes discussion of Dutch examples like that cited above.

===============

[No homework. If there were, it would be to do some or all of the questions at the end of Chapter 18.]

References

Bach, Emmon, Brown, Colin, and Marslen-Wilson, William. 1986. Crossed and nested dependencies in Dutch and German. *Language and Cognitive Processes* 1:249--262.

Gazdar, Gerald. 1982. Phrase structure grammar. In *The Nature of Syntactic Representation*, eds. Pauline Jacobson and Geoffrey Pullum, 131-186. Dordrecht: D.Reidel.

Gazdar, Gerald. 1983a. Phrase structure grammars and natural languages. In *IJCAI-83*, 556-565.

Gazdar, Gerald. 1983b. *NLs, CFLs and CF-PSGs*. Chichester/New York: Ellis Horwood/Wiley.

Gazdar, Gerald, Klein, Ewan, Pullum, Geoffrey, and Sag, Ivan. 1985. *Generalized Phrase Structure Grammar*. Oxford: Basil Blackwell.

Harman, Gilbert. 1963. Generative grammars without transformation rules: a defense of phrase structure. *Language* 39:597- 616. Reprinted in Walter J. Savitch, Emmon Bach, William Marsh and Gila Safran-Naveh, editors, *The Formal Complexity of Natural Language* (Dordrecht, Holland; D. Reidel: 1987) pp. 87-116.

Joshi, Aravind K. 1985. Tree adjoining grammars: how much context-sensitivity is required to provide reasonable structural descriptions? In *Natural Language Parsing*, ed. Lauri Karttunen and Arnold M. Zwicky David R. Dowty, 206-250. Cambridge: Cambridge University Press.

Joshi, Aravind K., Vijay-Shanker, K., and Weir, D.J. 1989. The convergence of Mildly Context-Sensitive Grammar formalisms. Report MS-CIS-89-14, LINC LAB 144. Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA.

Marsh, William, and Partee, Barbara. 1984. How non-context-free is variable binding? In *Proceedings of the West Coast Conference on Formal Linguistics III*, eds. M. Cobler, S. MacKaye and M. Wescoat, 179-190. Stanford, CA.

Pullum, Geoffrey, and Gazdar, Gerald. 1982. Natural languages and context free languages. *Linguistics and Philosophy* 4:471-504.

Pullum, Geoffrey K. 1984. On two recent attempts to show that English is not a CFL. *Computational Linguistics* 10:182-186.

Pullum, Geoffrey K. 1986. Topic ... Comment: Footloose and context-free. *Natural Langauge and Linguistic Theory* 4:409-414. reprinted in Pullum (1991), 131-8.

Shieber, Stuart M. 1985. Evidence Against the Context-Freeness of Natural Language. *Linguistics and Philosophy* 8:333-344.