Context Sensitive Grammar and LBA

Dr. Mousumi Dutt

Module 3

Introduction

- This leads to another important grammar based class of languages: context-sensitive languages. As it turns out, CSL are plenty strong enough to describe programming languages - but in the real world it does not matter, it is better to think of programming language as being context-free, plus a few extra constraints.

- As we will see shortly, though, the associated machines (linear bounded automata) are quite important in complexity theory.

Introduction

- Hewlett-Packard figured out 40 years ago the reverse Polish notation is by far the best way to perform lengthy arithmetic calculations. Very easy to implement with a stack.

- Example: we would want the compiler to address this kind of situation

- To deal with problems like this one we need to strengthen our grammars. The key is to remove the constraint of being “context-free.”

Context Sensitivity in Programming Language

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.

Context Sensitivity in Programming Language

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 can be shown that 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 Programming Language

Context-sensitive languages are a big step up from context-free languages in terms of their power and generality.

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.

Programming languages contain non-context-free features (typing, scoping etc.), but all these fall comfortably within the realm of context-sensitive languages.

Next time: what kinds of machines are needed to recognize context-sensitive languages?

Context Sensitive Grammar (CSG)

Context Sensitive Grammar(Type1 Grammar)

- A context-sensitive grammar (CSG) is an unrestricted grammar in which every production has the form
  $\alpha \to \beta$ with $|\beta| \geq |\alpha|$ (where α and β are strings of nonterminals and terminals).
- The concept of context-sensitive grammar was introduced by Noam Chomsky in the 1950.
- In every derivation the length of the string never decreases.
- The term "context-sensitive" comes from a normal form for these grammars,where each production is of the form $\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2$,with $\beta \neq \epsilon$.
- They permit replacement of variable A by string β only in the "context " α_1 - α_2.

Formal Definition of Context Sensitive Grammar (CSG)

A Context Sensitive Grammar is a 4-tuple , G = (N,Σ, P, S) where:

- N=Set of non terminal symbols.
- Σ=Set of terminal symbols.
- S=Start symbol of the production.
- P=Finite set of productions.
  All rules in P are of the form $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$.
  - $A \in N$ (A is a single nonterminal)
  - $\alpha_1, \alpha_2, \beta \in (N \cup \Sigma)^+$.

The production S →ε is also allowed if S is the start symbol and it does not appear on the right side of any production.

Formal Definition of Context Sensitive Grammar (CSG)

A context-sensitive grammar (CSG) is a grammar where all productions are of the form

$$\alpha A \beta \to \alpha \gamma \beta \qquad \text{where } \gamma \neq \varepsilon$$

Some authors also allow $S \to \varepsilon$ in which case S may not appear on the righthand side of any production. A language is context-sensitive if it can be generated by a context-sensitive grammar.

Note the constraint that the replacement string $\gamma \neq \varepsilon$; as a consequence we have

$$\alpha \Rightarrow \beta \qquad \text{implies} \qquad |\alpha| \leq |\beta|$$

This should look familiar from our discussion of ε-free CFG.

Context Sensitive Language (CSL)

The language generated by the Context Sensitive Grammar is called context sensitive language.

If G is a Context Sensitive Grammar then

L(G)=$\left\{w \,\middle|\, (w \in \Sigma^*) \wedge \left(S \Rightarrow_G^+ w\right)\right\}$.

Eg 1 of a context sensitive grammar

$G = \{\{S, A, B, C, a, b, c\}, \{a, b, c\}, P, S\}$where P is the set of rules.

$$\begin{gathered} \text{S} \rightarrow \text{aSBC} \\ \text{S} \rightarrow \text{aBC} \\ \text{CB} \rightarrow \text{BC} \\ \text{aB} \rightarrow \text{ab} \\ \text{bB} \rightarrow \text{bb} \\ \text{bC} \rightarrow \text{bc} \\ \text{cC} \rightarrow \text{cc} \end{gathered}$$

The language generated by this grammar is $\{a^n b^n c^n | n \geq 1\}$.

Context Sensitive Language (CSL)

The derivation for the string aabbcc is

$$
\begin{aligned}
S &\Rightarrow aSBC \\
&\Rightarrow aaBCBC \\
&\Rightarrow aabCBC \\
&\Rightarrow aabBCC \\
&\Rightarrow aabbCC \\
&\Rightarrow aabbcC \\
&\Rightarrow aabbcc
\end{aligned}
$$

CSL, L={#a= #b = #c}

$$G2 = (\{S, A, B, C, a, b, c\}, \{a, b, c\}, P, S)$$

$$S \to ABC$$

$$S \to ABCS$$

$$AB \to BA$$

$$AC \to CA$$

$$BC \to CB$$

$$BA \to AB$$

$$CA \to AC$$

$$CB \to BC$$

$$A \to a$$

$$B \to b$$

$$C \to c$$

General and Noncontracting Grammar

In a general or unrestricted grammar, we allow productions of the form

$$\alpha \rightarrow \beta$$

where α, β are sequences of terminals and nonterminals, i.e., $\alpha, \beta \in (N \cup \Sigma)^*$, with α containing at least one nonterminal.

In a noncontracting grammar, we restrict productions to the form

$$\alpha \rightarrow \beta$$

with α, β as above, subject to the additional requirement that $|\alpha| \leq |\beta|$ (i.e., the sequence β is at least as long as α).
In a noncontracting grammar also permit the special production

$$S \rightarrow \epsilon$$

where S is the start symbol, as long as S does not appear on the right-hand-side of any productions.

General and Noncontracting Grammar

Consider the noncontracting grammar with start symbol S:

$$\begin{array}{rcl} S & \rightarrow & abc \\ S & \rightarrow & aSBc \\ cB & \rightarrow & Bc \\ bB & \rightarrow & bb \end{array}$$

Example derivation (underlining the sequence to be expanded):

$$\underline{S} \Rightarrow a\underline{S}Bc \Rightarrow aab\underline{cB}c \Rightarrow aa\underline{bB}cc \Rightarrow aabbcc$$

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

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

General and Noncontracting Grammar

Theorem. A language is context sensitive if and only if it can be generated by a noncontracting grammar.

That every context-sensitive language can be generated by a noncontracting grammar is immediate, since context-sensitive grammars are, by definition, noncontracting.

The proof that every noncontracting grammar can be turned into a context sensitive one is intricate, and beyond the scope of the course.

Sometimes (e.g., in Kozen) noncontracting grammars are called context sensitive grammars; but this terminology is not faithful to Chomsky's original definition.

Decidablity

Lemma

Every context-sensitive language is decidable.

Proof.

Suppose $w \in \Sigma^\star$ and $n = |w|$. In any potential derivation $(\alpha_i)_{i<N}$ we have $|\alpha_i| \leq n$.

So consider the digraph D with vertices $\Gamma^{\leq n}$ and edges $\alpha \Rightarrow^1 \beta$.

Then w is in L if w is reachable from S in D.

□

Of course, the size of D is exponential, so this method won't work in the real world.

Decidablity

Not all decidable languages are context-sensitive.

Here is a cute diagonalization argument for this claim.

Let $(x_i)_i$ be an effective enumeration of $\Sigma^\star$ and $(G_i)_i$ an effective enumeration of all CSG over Σ (say, both in length-lex order). Set

$$L = \{\, x_i \mid x_i \notin \mathcal{L}(G_i) \,\}$$

Clearly, L is decidable.

Closure Properties of CSL

Context Sensitive Languages are closed under

- Union
- Intersection
- Complement
- Concatenation
- Kleene closure
- Reversal

Every Context sensitive language is recursive

Union

L(CS) closed under union

- Let $G_1 = (N_1, T_1, P_1, S_1)$ and $G_2 = (N_2, T_2, P_2, S_2)$,
  s.t $L(G_1) = L_1$ and $L(G_2) = L_2$.
- Construct $G = (S \cup N_1 \cup N_2, T_1 \cup T_2, \{S \to S_1, S \to S_2\} \cup P_1 \cup P_2, S)$,
  s.t $N_1 \cap N_2 = \varnothing$ and $S \notin \{N_1 \cup N_2\}$.
- G also CSG and any derivation has the form
  $S \Rightarrow S_i \Rightarrow^*_{G_i} w \in L(G_i)$ for some $i \in \{1, 2\}$.
- We cannot merge the productions of P_1 and P_2.
- We can derive only words and all words of $L(G_1) \cup L(G_2) = L_1 \cup L_2$.
  Therefore $L_1 \cup L_2 = L(G) \in L(CS)$.

Concatenation

L(CS) closed under concatenation

- Let $G_1 = (N_1, T, P_1, S_1)$ and $G_2 = (N_2, T, P_2, S_2)$, s.t $L(G_1) = L_1$ and $L(G_2) = L_2$.
- Construct $G = (S \cup N_1 \cup N_2, T, \{S \to S_1 S_2\} \cup P_1 \cup P_2, S)$, s.t $N_1 \cap N_2 = \varnothing$ and $S \notin \{N_1 \cup N_2\}$.
- Any derivation in G has the form

  $$S \Rightarrow S_1 S_2 \Rightarrow^*_{G_1} w_1 S_2 \Rightarrow^*_{G_2} w_1 w_2$$

  for $i \in \{1, 2\}$, $S_i \Rightarrow w_i$ is a derivation in G_i. i.e. the derivation only uses rules of P_i.
- The derivations in G_1 and G_2 cannot be influenced by the contexts of the other part. So G is a context sensitive grammar, $L(G)$ is a CSL.

Theorem

Every context-sensitive language L is recursive.

For CSL L, CSG G, Derivation of w $S \Rightarrow x_1 \Rightarrow x_2 \Rightarrow x_3 \cdot \cdot \cdot \Rightarrow w$ has bound on no of steps.(Bound on possible derivations). We know that $|x_i| \leq |x_{i+1}|$ (G is non contracting). We can check whether w is in $L(G)$ as follows

- Construct a transition graph whose vertices are the strings of length $\leq |w|$.
- Paths correspond to derivation in grammars.
- Add edge from x to y if $x \Rightarrow y$
- $w \in L(G)$ iff thers is a path from S to w.
- Use path fining algorithm to find.

Theorem

There exists a recursive language that is not context sensitive.

Language L is recursive

- Create possible CSG $G_i = (N_i, \{0,1,2,3,4,5,6,7,8,9\}, S_i, P_i)$ which generates numbers.
- Now, define language L, which contains the numbers of the grammars which does not generate the number of its position in the list: $L = \{i \mid i \notin L(G_i)\}$.
- We can create a list of all context-sensitive generative grammars which generates numbers, and we can decide whether or not a context-sensitive grammar generates its position in the list.
- So language L is recursive.

Theorem

There exists a recursive language that is not context sensitive.

Language L is not context sensitive

- Assume, for contradiction, that L is a CSL
- So there is a CSG G_k, s.t $L(G_k) = L$ for some k.
- If $k \in L(G_k)$, by the definition of L, we have $k \notin L$, but $L = L(G_k)$. So a contradiction.
- If $k \notin L(G_k)$, then $k \in L$ is also a contradiction since $L = L(G_k)$.
- So language L is not context sensitive.

THANK YOU

More on next class…