Parsing CFLS: Memoization and Dynamic Programming

Polynomial time parsing

As an example of the application of memoization and dynamic programming, we consider the problem of deciding, given a context free grammar, a particular nonterminal, and a string of terminals, whether the string of terminals can be derived from the given nonterminal using the rules of the grammar. This is the parsing problem for context free grammars.

We will assume that the grammar is in Chomsky Normal Form, which means that every rule is in one of two forms: (1) $A \rightarrow a$, or (2) $A \rightarrow BC$. That is, the right hand side of every rule is either a single terminal symbol or exactly two nonterminal symbols. It is possible to transform any context free grammar (that doesn’t generate the empty string) into this form efficiently. Note that no nonterminal generates the empty string.

Here is an example of a grammar in Chomsky Normal Form:

\[
\begin{align*}
A & \rightarrow BC \mid CD \\
B & \rightarrow b \mid BC \\
C & \rightarrow c \mid BC \mid CC \\
D & \rightarrow b \mid d \mid CD
\end{align*}
\]

The terminals in this grammar are $b, c, d$, and the nonterminals are $A, B, C, D$. Consider the problem of deciding whether the string $bccd$ can be derived from the nonterminal symbol $A$ using the rules of this grammar.

We can approach this problem with a recursive strategy as follows. $A$ can derive the string $bccd$ if either $BC$ can derive this string or $CD$ can derive this string. $BC$ can derive this string if there is a way of dividing the string $bccd$ into two nonempty parts such that $B$ generates the first part, and $C$ generates the second part.

In more detail, $BC$ generates $bccd$ if and only if at least one of the following cases holds.

1. $B$ generates $b$ and $C$ generates $ccd$.
2. $B$ generates $bc$ and $C$ generates $cd$.
3. $B$ generates $bcc$ and $C$ generates $d$.

Thus, six recursive calls of our procedure suffice to determine whether $BC$ generates the string $bccd$. Similarly, six recursive calls of our procedure suffice to determine whether $CD$ generates the string $bccd$, because $CD$ generates $bccd$ if and only if at least one the following cases holds.

1. $C$ generates $b$ and $D$ generates $ccd$.
2. $C$ generates $bc$ and $D$ generates $cd$.
3. $C$ generates $bcc$ and $D$ generates $d$.

What are the base cases? When the string is one terminal symbol, then the question is just whether there is a rule whose lefthand side is the given nonterminal symbol, and whose righthand side is the single
terminal symbol. This gives a complete recursive solution to the problem of parsing a Chomsky Normal Form grammar.

Is this a case in which memoization would help? Yes, because overall there will not be so many different arguments for the recursive calls. The first argument can be any nonterminal in the grammar (in the example, one of \(A, B, C, D\)) and the second argument can be any nonempty substring of the original string (in the example, a nonempty substring of \(bccd\)).

A substring is a sequence of contiguous characters from the string. The number of substrings is bounded by choosing two places to cut the string – to the left of the leftmost character of the substring and to the right of the rightmost character of the substring. If the input string has \(n\) characters, there are \((n + 1)\) possible places to cut (including before the first character and after the last character), giving a total of at most \((n + 1)n/2\) distinct nonempty substrings of the input string. Thus, a memoization table has to hold \(O(mn^2)\) entries, where \(n\) is the length of the input string, and \(m\) is the number of nonterminal symbols in the grammar.

Converting this approach to a “bottom up” dynamic programming method, we get a version of the classical algorithm for parsing general Chomsky Normal Form grammars, called chart parsing or Cocke-Kasami-Younger (CKY) parsing. If we regard the grammar as fixed and bound the time to parse a string of length \(n\) using this method, the time grows as \(O(n^3)\). (Interestingly, this bound was improved by L. G. Valiant using a formal relationship between parsing and matrix multiplication.)

The setup for chart parsing is a two-dimensional array of boxes, where each box is uniquely associated with a nonempty substring of the string to be parsed. What we enter into the boxes is the set of all the nonterminals that can derive the associated substring. We do this systematically, from the bottom up as follows.

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\hline
1 & | & | & | \\
2 & | & | & | \\
3 & | & | & | \\
4 & | B,D & C & C & D \\
\end{array}
\]

Here in the bottom row of boxes (row 4) we’ve entered the nonterminals that can generate each of the terminal symbols according to the grammar rules \(B \rightarrow b\), \(C \rightarrow c\), \(D \rightarrow b\) and \(D \rightarrow d\).

Each of the row 4 boxes refers to the substring consisting of one character starting in that column. In general, for row \(j\), we consider only columns 1 through \(j\) and the box refers to the substring of length \((n - j) + 1\) starting in the column of the box. Thus, in row 3 we consider the boxes in columns 1, 2, and 3, referring to substrings \(bc, cc\), and \(cd\) of the input; in row 2 we consider the boxes in columns 1 and 2, referring to the substrings \(bcc\) and \(ccd\) of the input, and in row 1 we consider only the box in column 1, referring to the whole input string \(bccd\).

To fill in the box in row 3 and column 1 we consider the boxes in (row 4, column 1) and (row 4, column 2), which have, respectively \(B, D\) and \(C\). We consider all rules with right hand sides \(BC\) or \(DC\), which are \(A \rightarrow BC\), \(B \rightarrow BC\) and \(C \rightarrow BC\). Thus, we can put \(A, B, C\) in the box in (row 3, column 1). Similarly, we can put \(C\) in the box in (row 3, column 2), and \(A, D\) in the box in (row 3, column 3). The result is the
Considering the box in (row 2, column 1), we consider two pairs of boxes. The first pair of boxes is (row 3, column 1) and (row 4, column 3), corresponding to dividing the string $bcc$ into $bc$ and $c$. We have $A, B$ in the first box and $C$ in the second box, so we consider all rules with right hand sides $AC$ or $BC$, and find $A \rightarrow BC$, $B \rightarrow BC$ and $C \rightarrow BC$. Thus $A, B, C$ are to be put into the box in (row 2, column 1). The second pair of boxes is (row 4, column 1) and (row 3, column 2), corresponding to dividing the string $bcc$ into $b$ and $cc$. These boxes have, respectively, $B, D$ and $C$, so we consider all rules with right hand sides $BC$ or $DC$, which are $A \rightarrow BC$, $B \rightarrow BC$ and $C \rightarrow BC$. Thus we put $A, B, C$ into the box in (row 2, column 1).

A similar calculation for the box in (row 2, column 2) means that we consider the two pairs of boxes (row 3, column 2) (with $C$) and (row 4, column r) (with $D$) and (row 4, column 2) (with $C$) and (row 3, column 3) (with $A, D$). Thus, we consider rules with right hand sides $CD$ and $CA$, which are $A \rightarrow CD$ and $D \rightarrow CD$, so we put $A, D$ in the box in (row 2, column 2). The result is as follows.

Finally, to calculate what goes in the box in (row 1, column 1), we consider 3 pairs of boxes. The first pair is (row 2, column 1) and (row 4, column 4), giving possible right hand sides of $AD$, $BD$ and $CD$. The second pair is (row 3, column 1) and (row 3, column 3), giving possible right hand sides of $AA$, $AD$, $BA$, $BD$, $CA$, $CD$. The third pairs is (row 4, column 1) and (row 2, column 2), giving possible right hand sides of $BA$, $BD$, $DA$ and $DD$. All the rules with right hand sides in this set are $A \rightarrow CD$ and $D \rightarrow CD$, so we put $A, D$ into the (row 1, column 1) box, yielding the following result.
Thus the string $bccd$ is derivable from the nonterminal $A$, for example, via the following derivation:

$$
A \vdash CD \vdash BCD \vdash BCCD \vdash bCCD \vdash bcCD \vdash bccd.
$$

It is also derivable from the nonterminal $D$, because we have $D \vdash CD$.

Clearly there are $n(n + 1)/2$ boxes in the chart, and for each one we have to consider at most $(n - 1)$ pairs of boxes, for a total running time of $O(n^3)$, if we consider the grammar as fixed. This gives polynomial time parsing of general context free grammars. However, $n^3$ is not a very attractive polynomial in practice; computer languages are designed to have grammars that may be parsed more easily.