CS310 : Automata Theory 2019

Lecture 21: Applications of CFG

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2019-02-21

CFG for defining languages

- CFGs can be used to define practical languages
  - Natural languages
  - Program languages
- How do we know if a word is in the language?
  - Build parse tree

Backtracking based algorithm for for CFG parsing

1. Translate the grammar to PDA
2. Run the PDA
   2.1 At each nondeterministic choice, records the choices
   2.2 if a run gets stuck, backtrack to the recorded choices
   2.3 run again with one of the choices
3. If all runs get stuck, parsing fails

Top down parsing

May run into infinite loop if we have $A \stackrel{*}{\Rightarrow} A\beta$

Backtracking based algorithm for CFG parsing

Called look ahead in compilers

Optimization :

- For every nonterminal, compute the set of terminals that can occur in the first position
  - execute only those transitions that are compatible with the current input symbol
- Avoid performing duplicate runs with identical position and nonterminal head of stack: memoize previous successful run segments

Example 21.1

If we observe in some run

$$(q, xy, Z\alpha) \vdash^* (q, y, \alpha)$$

We record $(|xy|, Z, |x|)$ *and use it to shortcut running the PDA again*

Cocke-Younger-Kasami (CYK) algorithm for CFG parsing

We can do lot better with some dynamic programming

CYK algorithm assumes the grammar $G = (\{A_1, ..A_m\}, T, P, A_1)$ is in CNF.

Let $w = a_1...a_n$ be the input word.

CYK algorithm maintains a three dimensional bitvector $C[n][n][m]$.

$$C[i][j][k] = true \text{ means that } A_k \overset{*}{\Rightarrow} a_{i+1}....a_j$$

It is a bottom up parsing

CYK bottom up matching strategy

If we have

- $C[i][i+j][q]$, i.e., $A_q \stackrel{*}{\Rightarrow} a_{i+1}..a_{i+j}$
- $C[i+j][i+\ell][r]$, i.e., $A_r \stackrel{*}{\Rightarrow} a_{i+j+1} \dots a_{i+\ell}$
- $A_p \rightarrow A_q A_r$

$$\underbrace{\overbrace{a_{i+1}..a_{i+j}}^{C[i][i+j][q]}\;\overbrace{a_{i+j+1} \dots a_{i+\ell}}^{C[i+j][i+\ell][r]}}_{C[i][i+\ell][p]}$$

we conclude $C[i][i+\ell][p]$, i.e.,$A_p \stackrel{*}{\Rightarrow} a_{i+1} \dots a_{i+\ell}$

CYK Parsing

Algorithm 21.1: CYK($a_1, ...a_n$, $G = (\{A_1, ..A_m\}, T, P, A_1)$)

Output: Is $a_1..a_n \in L(G)$?

Boolean array $C[n][n][m]$, all enteries initialized to *false*;

for $i = 1$ *to* n **do**

> k be such that $A_k \rightarrow a_i$;
>
> $C[i-1][i][k] :=$ *true* // Initializing single length matching

for $\ell = 2$ *to* n **do**

> // consider ℓ long matches
>
> **for** $i = 0$ *to* $n - \ell$ **do**
>
> > // find matches at all the starting points
> >
> > **for** $j = 1$ *to* $\ell - 1$ **do**
> >
> > > // j is the length of the first part
> > >
> > > // Naturally, $\ell - j$ is the length of the second part
> > >
> > > **for** $A_p \rightarrow A_q A_r$ **do**
> > >
> > > > // Look for all applicable rules
> > > >
> > > > **if** $C[i][i+j][q]$ *and* $C[i+j][i+\ell][r]$ **then**
> > > >
> > > > > $C[i][i+\ell][p] :=$ *true*

return $C[0][n][1]$

Example: CYK run

Example 21.2

Instead of drawing 3D table, we write the set of symbols for which the $C[i][i+\ell]$ bits are true

Consider the following almost CNF grammar

1. $E \rightarrow 0 \mid 1$
2. $E \rightarrow EP$
3. $P \rightarrow +E$
4. $E \rightarrow EM$
5. $M \rightarrow \times E$

and word $w = 0 + 1 \times 0$

$i+\ell$	1	E				
	2		+			
	3	E	P	E		
	4				×	
	5	E	P	E	M	E
	i	0	1	2	3	4
	w	0	+	1	×	0

Exercise 21.1

a) Is there a cell filled because of application of two rules?

b) Where is the parse tree?

Supporting parse tree

Whenever we set $C[i][i+\ell][p]$ to true we need to keep the record of

- the split j and
- the used symbol indexes q and r

in another table.

From the information, we can construct the parse tree.

About CYK algorithm

- Bottom up process collects chunks of grammatically correct subwords
- CYK moves iteratively without caring about blank cells

Let us develop a worklist based algorithm to mitigate this problem.

Marked rules

Let $G = (N, T, P, S)$ be a grammar.

We define the following object that records the work done and to be done.

$$\text{Let} \quad MRules \triangleq N \cup T \cup (P \times \mathbb{N})$$

X : *Mrules* has the following possibilities

- $X \in T$
- $X \in N$
- $X = (A \rightarrow X_1...X_k\beta, k)$ or denoted $X = A \rightarrow X_1...X_k \bullet \beta$

k symbols in the rule has been matched and rest are yet to be matched.

Commentary: $\bullet$ is used to mark k

Worklist based CYK

Algorithm 21.2: WORKLISTCYK($a_1, ...a_n$, $G = (N, T, P, S)$)

worklist, *store* : $\mathfrak{p}(MRules \times \mathbb{N} \times \mathbb{N})$;

for $i = 1$ *to* n **do** add $(a_i, i-1, i)$ to *worklist* and *store*; ;

while *worklist* $\neq \emptyset$ **do**

  choose $(M, i, j) \in worklist$; $worklist := worklist - \{(M, i, j)\}$;

  > M is matched between i and j

  if $M = X \in T \cup N$ **then**

    for $A \to X\alpha \in P$ **do** CHECKANDADD($A \to X \bullet \alpha, i, j$) ;

    for $k = 0$ *to* $i-1$ **do**

      for $(A \to \beta \bullet X\alpha, k, i) \in store$ **do**

        CHECKANDADD($A \to \beta X \bullet \alpha, k, j$)

  else

    // $M = A \to \beta \bullet Y\alpha$

    for $k = j+1$ *to* n **do**

      if $(Y, j, k) \in store$ **then** CHECKANDADD($A \to \beta Y \bullet \alpha, i, k$) ;

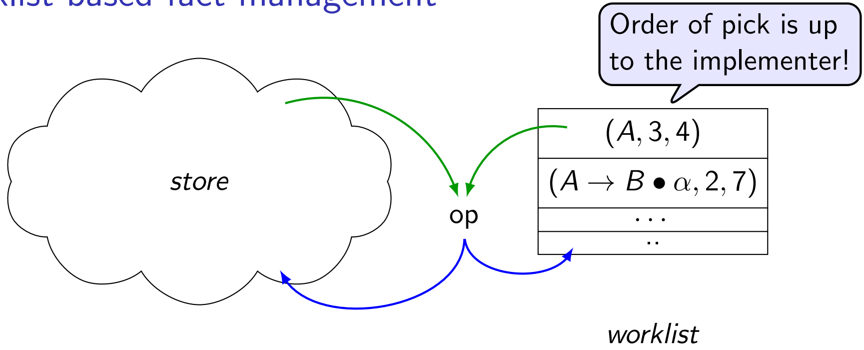
return $(S, 0, n) \in store$;

CHECKANDADD($A \to \beta \bullet \alpha, i, j$)

  if $\alpha = \epsilon$ *and* $(A, i, j) \notin store$ **then** add (A, i, j) to *worklist* and *store* ;

  if $(A \to \beta \bullet \alpha, i, j) \notin store$ **then** add $(A \to \beta \bullet \alpha, i, j)$ to *worklist* and *store* ;

Worklist based fact management



Let us suppose the top level algorithm has fact deduction rules *Ops*
Initialize *worklist* and *store* with initial facts

1. Pick a fact F from worklist and remove F from the worklist
2. For each rule $op \in Ops$ and fact $F' \in store$, deduce fact $op(F, F')$
3. If $op(F, F')$ is new, add to both *store* and *worklist*

Keep doing it until *worklist* is empty.

CYK deductions

CYK algorithm has the following two deductions

1. Upward prediction:
   (X, i, j) and $A \rightarrow X\alpha \in P$, then $(A \rightarrow X \bullet \alpha, i, j)$

2. Completion:
   If $(A \rightarrow \beta \bullet Y\alpha, i, j)$ and (Y, j, k), then $(A \rightarrow \beta Y \bullet \alpha, i, k)$

Rest of the description of the algorithm is the fact management

Example: Running WORKLISTCYK

Example 21.3

Consider the following grammar

1. $E \to 0 \mid 1$
2. $E \to EP$
3. $P \to +E$
4. $E \to EM$
5. $M \to \times E$

and word $w = 0 + 1 \times 0$

Collected facts in the run of WORKLISTCYK

Initialize
$(0, 0, 1)$, $(+, 1, 2)$, $(1, 2, 3)$, $(\times, 3, 4)$, $(0, 4, 5)$

$(E, 0, 1)$, $(E, 2, 3)$, $(E, 4, 5)$
$(E \to E \bullet P, 0, 1)$, $(E \to E \bullet M, 0, 1)$,$(E \to E \bullet M, 2, 3)$,
$(E \to E \bullet P, 2, 3)$, $(E \to E \bullet P, 4, 5)$, $(E \to E \bullet M, 4, 5)$,
$(P \to + \bullet E, 1, 2)$, $(M \to \times \bullet E, 3, 4)$

Lots of obviously useless facts generated!

$(P, 1, 3)$, $(M, 3, 5)$,

$(E, 0, 3)$, $(E, 2, 5)$,

$(P, 1, 5)$,

$(E, 0, 5)$,

$(E \to E \bullet P, 0, 3)$, $(E \to E \bullet M, 0, 3)$,
$(E \to E \bullet P, 2, 5)$, $(E \to E \bullet M, 2, 5)$,

Chart parsing

- Using dynamic programming, CYK keeps the record of the parts already parsed and yet to be parsed
- This class of algorithms are called chart parsing methods.
- Since CYK is bottom up, it is oblivious to the goal of matching the whole string to the start symbol S
- Earley Parsing is another chart parsing method that addresses this

Earley Parsing

Earley Parsing is top down parsing.

- Initialize: $(S \rightarrow \bullet\alpha, 0, 0)$ if $S \rightarrow \alpha \in P$
- Three deduction rules
  - Prediction:
    If $(A \rightarrow \gamma \bullet Y\alpha, i, j)$ and $Y \rightarrow \beta \in P$, then $(Y \rightarrow \bullet\beta, j, j)$.
  - Scanning:
    If $(A \rightarrow \beta \bullet a_{j+1}\alpha, i, j)$, then $(A \rightarrow \beta a_{j+1} \bullet \alpha, i, j+1)$.
  - Completion:
    If $(A \rightarrow \beta \bullet Y\alpha, i, j)$ and (Y, j, k), then $(A \rightarrow \beta Y \bullet \alpha, i, k)$.

Essentially, running the PDA with dynamic programming.

Example: Running WorklistCYK

Example 21.4

Consider the following grammar

1. $E \rightarrow 0 \mid 1$
2. $E \rightarrow EP$
3. $P \rightarrow +E$
4. $E \rightarrow EM$
5. $M \rightarrow \times E$

and word $w = 0 + 1 \times 0$

Collected facts in the run of WorklistCYK

Initialize
$(E \rightarrow \bullet 0, 0, 0)$, $(E \rightarrow \bullet 1, 0, 0)$,
$(E \rightarrow \bullet EP, 0, 0)$, $(E \rightarrow \bullet EM, 0, 0)$,

$(E \rightarrow 0, 0, 1)$,
$(E \rightarrow E \bullet P, 0, 1)$, $(E \rightarrow E \bullet M, 0, 1)$,

$(P \rightarrow \bullet + E, 1, 1)$, $(M \rightarrow \bullet \times E, 1, 1)$,
$(P \rightarrow + \bullet E, 1, 2)$,

$(E \rightarrow \bullet 0, 2, 2)$, $(E \rightarrow \bullet 1, 2, 2)$,
$(E \rightarrow \bullet EP, 2, 2)$, $(E \rightarrow \bullet EM, 2, 2)$,

.....

An optimization on Earley parsing: left-corner parsing

- Further restriction to only predict plausible choices
- Left-corner relation
  - $A > B \triangleq \exists \beta A \rightarrow B\beta$
  - $A >^* B$ is transitive closure of $A > B$
- Updated prediction step
  If $(A \rightarrow \gamma \bullet Y\alpha, i, j)$, $Y >^* C$, $C \rightarrow B\beta \in P$, and (B, j, k), then $(C \rightarrow B \bullet \beta, j, k)$.
- More informed predictions
- Mixes top down and bottom up parsing

End of Lecture 21