Theory of Bottom Up Parsing

Bottom Up Parsing

We study the theory of bottom up parsing. Bottom up parsing means that we start with a word that we hope to be in the language, and try to rewrite it into the start symbol by applying the rewrite rules backwards, i.e. from right to left.

We call backward replacements reductions. During a reduction, one can also compute the attribute of the left hand side.

Bottom Up Parsing Using a Pebble

Assume that the end marker is $\#$, and we try to parse the word w.

During parsing, we will insert a pebble $\bullet$ into the word.

We call a word, followed by $\#$ and with $\bullet$ inserted somewhere a configuration.

Note that $\bullet$ cannot be put after $\#$.

Bottom Up Parsing Using a Pebble (2)

The parser starts with configuration $\bullet w\#$.

At each configuration, the parser has two choices:

- If the $\bullet$ is not in front of $\#$, it can move the $\bullet$ one position to the right. This is called shift.
- If there is a rewrite rule whose right hand side fits, the parser can make a reduction just in front of the pebble.

Parsing ends succesfully, when the configuration $S \bullet \#$ is reached.

We call this configuration the accepting configuration.

Definition of $\vdash$

We use the symbol $\vdash$ to express one possible step of the parser.

Shift can be expressed as: For every word $w \in W^*$, for every $\sigma \in \Sigma$, for every $z \in \Sigma^*$,

$$w \bullet \sigma\, z\, \# \;\vdash\; w\sigma \bullet z\, \#.$$

Reductions can be expressed as: For every grammar rule $A \rightarrow v$, for every $w \in W^*$, and $z \in \Sigma^*$,

$$w\, v \bullet z\, \# \vdash wA \bullet z\, \#.$$

We write $\vdash^*$ for an arbitrary number of steps.

A word $z \in \Sigma^*$ can be succesfully parsed if $\bullet z\, \# \vdash^* S \bullet \#$.

Strictly from Left to Right

The pebble restricts the order in which reductions can be made. For example, consider grammar

$$\begin{array}{lcl} S & \to & AAA \\ A & \to & a \end{array}$$

In principle, one can reduce $aaa \Rightarrow aAa \Rightarrow AAa \Rightarrow AAA \Rightarrow S$.

Using the pebble, this is forbidden, and we can only construct the following parse:

$$\bullet aaa\,\# \;\vdash\; a \bullet aa\,\# \;\vdash\; A \bullet aa\,\# \;\vdash\; Aa \bullet a\,\# \;\vdash$$

$$AA \bullet a\,\# \;\vdash\; AAa \bullet \# \;\vdash\; AAA \bullet \# \;\vdash\; S \bullet \#$$

If we start by moving the $\bullet$ two positions forward, we get $aa \bullet a\#$, and we will not be able to reach the accepting configuration any more.

Regularity of the Prefix Language

Let R be a set of rewrite rules. The prefix language of R is defined as the set of words $z \in W^*$ for which there exist $w_1, w_2 \in \Sigma^*$, such that

$$\bullet\, w_1\#\ \vdash^* z \bullet w_2\,\# \vdash^* S \bullet \#.$$

Intuitively: The prefix language is the set of words z that can occur in front of the $\bullet$ in a succesful parse based on rules from R.

The main theoretical insight, which makes bottom up parsing possible, is:

For every R set of rewrite rules, for every start symbol S, the prefix language is regular.

Note that 'regular' implies 'totally easy to recognize once we have the DFA'.

Prefix Language $\neq$ All Prefixes of Correct Words

Consider the following grammar:

$$\begin{array}{lcl} S & \rightarrow & aSb \\ S & \rightarrow & \epsilon \end{array}$$

It generates $\mathcal{L} = \{\ a^i b^i \mid i \geq 0\ \}$.

What is the prefix language?

Warning: $a^i b^j$ with $i \geq j > 1$ is not in the prefix language.

Prefix Language

Assume that $i > 1$. In order to succesfully parse $a^i b^i$, first shift the a-s:

$$\bullet\, a^i b^i \vdash^* a^i \bullet b^i \#.$$

If we continue with $a^i b \bullet b^{i-1}\#$, we will never be able to reduce $S \rightarrow aSb$.

So we must reduce $S \rightarrow \epsilon$. The result is $a^i S \bullet b^i \#$. Now we can shift to $a^i Sb \bullet b^{i-1}\#$ and reduce into $a^{i-1} S \bullet b^{i-1}\#$.

We see that the prefix language is

$$\{\ a^i \mid i \geq 0\ \} \cup \{\ a^i S \mid i \geq 0\ \} \cup \{\ a^i Sb \mid i \geq 0\ \}.$$

This language is regular, even though the language $\{\ a^i b^j \mid i \geq j\ \}$ of prefixes of $\mathcal{L}$ is not regular.

The difference is caused by the fact that in a succesful parse, there is at most one b left of $\bullet$.

Getting the DFA

We show how to obtain the unique mininmal DFA for the prefix language. It is called the prefix automaton.

Items

Let $A \to w$ be a grammar rule. We partition w into two parts w_1 and w_2 (possibly empty), and write $A \to w_1 \bullet w_2$.

The result is called an item.

Grammar $S \to aSB$, $S \to \epsilon$ has 5 items:

$$S \to \bullet aSb$$

$$S \to a \bullet Sb$$

$$S \to aS \bullet b$$

$$S \to aSb\bullet$$

$$S \to \bullet$$

Item Sets

A set of items is called an item set.

The prefix automaton is constructed using item sets. The items sets are the states of the prefix automaton.

In order to obtain the prefix automaton, one needs two operations on item sets, closure and shift.

Closure

Let I be an itemset. The closure $\text{CLOS}_R(I)$ of I using grammar R is obtained by making the following insertions as long as possible:

If I contains an item of form $A \to w_1 \bullet B w_2$, and R contains a rule of form $B \to v$, then add $B \to \bullet v$ to I.

Shift

Let I be an itemset, let $s \in W$ be a symbol. The shift $\text{SHIFT}(I, s)$ of I under s is the set of items of form $A \rightarrow w_1 s \bullet w_2$ for which $A \rightarrow w_1 \bullet s\, w_2$ occurs in I.

In words: Check I for those items where the pebble stands right in front of symbol s. In those, put the pebble after s. Remove the other items.

The Prefix Automaton

In order to obtain the complete prefix automaton, start with the set of all items of form $S \rightarrow \bullet w$, where S is the start symbol, and close it.

This is the start state of the prefix automaton. After that, do the following as long as possible:

Pick an itemset I, a symbol s, and compute $I' = \mathrm{CLOS}_R(\ \mathrm{SHIFT}(I, s)\ )$. If $I' = \emptyset$, then nothing needs to be done. Otherwise, if I' is not yet a state of the automaton, add I' to the automaton.

If it is not yet present, add the transition $I \xrightarrow{s} I'$ to the automaton.

Keep on doing this until no more states and transitions can be added.

All states of the prefix automaton are accepting.

The Prefix Automaton (2)

For a realistic programming language, the prefix automaton has a couple of hundreds of states. Too big for a human to compute, but easy for a computer.

For rules $S \to aSb, \quad S \to \epsilon$, the initial state is $\{ \ S \to \bullet\, aSb, \quad S \to \bullet \ \}$. The state is already closed. There is one shift possible using a. All other shifts result in $\emptyset$. The result is $\{ \ S \to a \bullet Sb \ \}$. After closure we obtain

$$\{ \ S \to a \bullet Sb, \ \ S \to \bullet aSb, \ \ S \to \bullet \ \}.$$

The complete automaton is on the next slide.

Prefix Automaton for $S \to aSb, \quad S \to \epsilon$

$$\begin{tikzcd}
q_1 \arrow[r, "a"] & q_2 \arrow[r, "S"] \arrow[loop below, "a"] & q_3 \arrow[r, "b"] & q_4
\end{tikzcd}$$

$$q_1 = \left\{ \begin{array}{l} S \to \bullet aSb \\ S \to \bullet \end{array} \right\}, \quad q_2 = \left\{ \begin{array}{l} S \to a \bullet Sb \\ S \to \bullet aSb \\ S \to \bullet \end{array} \right\},$$

$$q_3 = \{\ S \to aS \bullet b\ \}, \quad q_4 = \{\ S \to aSb \bullet\ \}.$$

Making the Decisions

Using the prefix automaton, the parser can decide what should be done.

In general, the parser has the following options:

- If the symbol after $\bullet$ is not #, it can shift.
- If the configuration just before $\bullet$ contains the right side of a rule, it can reduce. It is possible that more than one rule can be reduced.

If the symbol right after $\bullet$ is not #, we eventually will have to shift it. So, we have zero or more reductions, followed by one shift. We check which option will be be accepted by the prefix automaton.

If there is no option, report 'syntax error'. If there is more than one option, the grammar is too hard for us.

Making the Decisions (2)

Suppose we try to parse $a^3b^3\#$. We start with $\bullet a^3b^3\#$. We have the choice between shifting a and reducing $S \rightarrow \epsilon$. If we reduce, we get $S \bullet a^3b^3$, but S is not accepted by the prefix automaton. So we have to shift.

This continues until we reach $a^3 \bullet b^3\#$. If we shift b, we get $a^3b \bullet b^2$, but a^3b will not be accepted by the prefix automaton. If we reduce $S \rightarrow \epsilon$, we get $a^3S \bullet b^3$. The word a^3S is accepted by the prefix automaton, so we reduce.

We again have to choose between reducing $S \rightarrow \epsilon$ or shifting b. Reducing $S \rightarrow \epsilon$ results in $a^3SS \bullet b^3$ which will be rejected by the prefix automaton. Shifting results in $a^3Sb \bullet b^2\#$, which is possible.

Making the Decisions (3)

In configuration $a^3Sb \bullet b^2\#$, we have three options:

1. Reduce $S \to \epsilon$. This results in prefix a^3SbS, which is not accepted by the prefix automaton.
2. Reduce $S \to aSb$, which results in prefix a^2S, which is accepted by the prefix automaton.
3. Shift b, which results in prefix a^3Sbb, which will be rejected by the prefix automaton.

So (2) is our only option.

L(1)

Languages that can be parsed in the way that we described on the previous slides are called L(1).

$L(1)$ stands for look ahead 1. This means that we need to see only one symbol in front of the $\bullet$ in order to decide what to do.

The following language is not L(1): $S \to aSa, \quad S \to \epsilon$. In order to decide when to reduce with $S \to \epsilon$, we need to know if we are half way. This can be done only by looking ahead all the way to the end.

On the following slides, we compare two languages.

$$\mathcal{L}_1 = \{\ E \rightarrow E + F, \quad E \rightarrow F, \quad F \rightarrow \mathrm{id}\ \}.$$

$$\mathcal{L}_2 = \{\ E \rightarrow F + E, \quad E \rightarrow F, \quad F \rightarrow \mathrm{id}\ \}.$$

The languages differ in how $\mathrm{id} + \mathrm{id} + \mathrm{id}$ is parsed. $\mathcal{L}_1$ will give priority to the first $+$, while $\mathcal{L}_2$ gives priority to the second.

Prefix Automaton of $E \to E + F, \quad E \to F, \quad F \to \mathrm{id}$

$$\begin{tikzcd}
q_1 \arrow[r, "E"] \arrow[d, "F"] \arrow[dr, "\mathrm{id}"] & q_4 \arrow[r, "+"] & q_5 \arrow[d, "F"] \arrow[dl, "\mathrm{id}"'] \\
q_2 & q_3 & q_6
\end{tikzcd}$$

$$q_1 = \left\{ \begin{array}{l} E \to \bullet E + F \\ E \to \bullet F \\ F \to \bullet\, \mathrm{id} \end{array} \right\}, \quad q_2 = \{\ E \to F\bullet\ \}, \quad q_3 = \{\ F \to \mathbf{id}\bullet\ \},$$

$$q_4 = \{\ E \to E \bullet +F\ \}, \quad q_5 = \left\{ \begin{array}{l} E \to E + \bullet F \\ F \to \bullet\, \mathrm{id} \end{array} \right\}, \quad q_6 = \{E \to E + F\bullet\}.$$

Parsing id + id + id

The parser for on the previous slide will proceed as follows: After shifting id, the configuration is $\mathrm{id} \bullet +\mathrm{id} + \mathrm{id}\,\#$. Shifting + would result in $\mathrm{id} + \bullet\mathrm{id} + \mathrm{id}\#$, which not be accepted by the prefix automaton, so we reduce and get $F \bullet +\mathrm{id} + \mathrm{id}\,\#$. The automaton is in state q_2, and shifting is not possible. So we reduce and get $E \bullet +\mathrm{id} + \mathrm{id}\,\#$. There is nothing to reduce, so we shift + and get $E + \bullet\mathrm{id} + \mathrm{id}\,\#$. No reduction is possible, so we shift id with result $E + \mathrm{id} \bullet +\mathrm{id}\#$. The prefix automaton is in state q_3, so we have to reduce. The result is $E + F \bullet +\mathrm{id}\#$. Since the prefix automaton is in state q_6, no shift is possible so we have to reduce $E \rightarrow E + F$. At this point, we have given priority to the first +.

Prefix Automaton of $E \to F + E, \quad E \to F, \quad F \to \text{id}$

$$\begin{tikzcd}
q_1 \arrow[r, "F"] \arrow[dr, "\text{id}"'] & q_2 \arrow[r, bend left, "+"] & q_4 \arrow[l, bend left, "F"] \arrow[dl, "\text{id}"] \arrow[r, "E"] & q_5 \\
 & q_3 & &
\end{tikzcd}$$

$$q_1 = \left\{ \begin{array}{l} E \to \bullet F + E \\ E \to \bullet F \\ F \to \bullet\,\text{id} \end{array} \right\}, \quad q_2 = \left\{ \begin{array}{l} E \to F \bullet + E \\ E \to F \bullet \end{array} \right\}, \quad q_3 = \{\, F \to \text{id}\,\bullet \,\},$$

$$q_4 = \left\{ \begin{array}{l} E \to F + \bullet E \\ E \to \bullet F + E \\ E \to \bullet F \\ F \to \bullet\,\text{id} \end{array} \right\}, \quad q_5 = \{\, E \to F + E \bullet \,\}.$$

Parsing id + id + id

The parser for $\mathcal{L}_2$ will proceed as follows: After shifting id, the configuration is $\text{id} \bullet +\text{id} + \text{id}\,\#$. Shifting $+$ would be not be accepted by the prefix automaton, so we reduce and get $F \bullet +\text{id} + \text{id}\,\#$.

At this point, we have the choice between reducing $E \to F$, or shifting $+$. If we reduce, we get configuration $E \bullet +\text{id} + \text{id}\,\#$, but E is not accepted by the prefix automaton. So we have to shift.

Reduction will happen only at the end, from configuration $F + F + F \bullet \#$. Since shifting is not possible, we reduce $E \to F$, and get $F + F + E \bullet \#$.

The parser reduces two times with $E \to F + E$, which results that the right $+$ is given priority.

LALR parsing

In reality, the analysis of when to shift or reduce is made in advance and stored in the state.

The resulting parsers are called LALR, which means look ahead left right.

Often, it can make the decisions without using a look ahead. Sometimes, a look ahead of 1 is required. (This matters for interactive applications).

Most parser generators (and in particular Bison, Yacc, CUP or Maphoon) construct LALR parsers.

Nearly all programming languages can be parsed by LALR parsers. The others can be parsed with small adaptations.