Computing Science, University of Aberdeen 1

CS4025: Parsing

- Top-down parsing
- Bottom-up parsing (shift-reduce/left corner)

See J&M Chapter 10 is 1st ed, 13 in 2nd

Computing Science, University of Aberdeen 2

Definitions

- A grammar is a description of what it takes to be a sentence of a language
- A recogniser is a program that takes a grammar and a string and indicates whether the string is a sentence according to the grammar
- A parser is a program that takes a grammar and a string and returns all possible analyses of the string according to the grammar (as trees, nested lists etc.)

It is possible to build a parser/recogniser for a single grammar, but in practice they are generic.

Computing Science, University of Aberdeen 3

Operation of a parser

Rules:
S = NP VP

Text: Sam sleeps

Parser

Parse:
s
np
name:: Sam
vp
intranV:: sleep

Computing Science, University of Aberdeen 4

Recogniser vs Parser

- A parser is basically a recogniser that retains a memory of how it has recognised the sentence (in all possible ways)
- For simplicity, we present parsing algorithms using recognisers
- It is fairly easy to change a recogniser into a parser by having it keep extra records.

Computing Science, University of Aberdeen 5

Top down recognition - Recursive Descent

- Start with initial symbol (eg, S), guess at an appropriate rewrite rule
- If first symbol is a non-terminal, rewrite it with another guessed rule. Keep doing until first symbol is a terminal.
  - leads to infinite loop in some cases!
- If terminal matches first word of real sentence, continue with second word.
- (As with any approach to recognition) need to be able to recover if the wrong choice is made (search)

Computing Science, University of Aberdeen 6

States of a top-down recogniser

The recogniser manipulates the combination of:

<table>
<thead>
<tr>
<th>Goals to find</th>
<th>Remaining words</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>All the words</td>
</tr>
<tr>
<td>We finish with</td>
<td>S</td>
</tr>
<tr>
<td>At any point, 2 actions may be possible:</td>
<td></td>
</tr>
</tbody>
</table>
1. S, OtherGoals | Words     New, OtherGoals | Words |
   if S \rightarrow New is a rule in the grammar.
2. S, OtherGoals | 6, OtherWords | OtherGoals OtherWords
Example grammar

- $S = NP \ VP$
- $VP = V \ NP$
- $NP = \text{Det} \ \text{Noun}$
- $NP = \text{Name}$
- $\text{Det}: \ a, \ the$
- $\text{Noun}: \ dog, \ cat$
- $\text{Name}: \ Fido, \ Misty, \ Phoebe, \ Mars$
- $V: \ chases, \ sees, \ likes$

Fido chases the cat
A cat sees Misty

Example recognition

<table>
<thead>
<tr>
<th>Goals</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>Phoebe likes Mars</td>
</tr>
<tr>
<td>$NP \ VP$</td>
<td>Phoebe likes Mars</td>
</tr>
<tr>
<td>$\text{Name} \ VP$</td>
<td>Phoebe likes Mars</td>
</tr>
<tr>
<td>$\text{Phoebe} \ VP$</td>
<td>Phoebe likes Mars</td>
</tr>
<tr>
<td>$VP$</td>
<td>likes Mars</td>
</tr>
<tr>
<td>$V \ NP$</td>
<td>likes Mars</td>
</tr>
<tr>
<td>likes $NP$</td>
<td>likes Mars</td>
</tr>
<tr>
<td>$NP$</td>
<td>Mars</td>
</tr>
<tr>
<td>$\text{Name}$</td>
<td>Mars</td>
</tr>
<tr>
<td>$\text{Mars}$</td>
<td>Mars</td>
</tr>
</tbody>
</table>

Problems

- Naive lexical goals (e.g. Mars) - in practice, look ahead
- Left recursive grammar, e.g.
  - distinguished(np).
  - $np \rightarrow \text{det}, \ ap, \ n$.  
  - $ap \rightarrow \text{ap}, \ a$.  
  - $\text{det} \rightarrow \{\text{the}\}$.  
  - $\text{a} \rightarrow \{\text{little}\}$.  
  - $\text{n} \rightarrow \{\text{old}\}$.  
- Indirect left recursion, e.g.
  - $np \rightarrow \text{det}, \ n$.  
  - $\text{det} \rightarrow \text{np}, \ \{s\}$.

Left Recursion

<table>
<thead>
<tr>
<th>Goals</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>$NP$</td>
<td>the little old lady</td>
</tr>
<tr>
<td>$\text{Det} \ \text{AP} \ N$</td>
<td>the little old lady</td>
</tr>
<tr>
<td>$\text{the} \ \text{AP} \ N$</td>
<td>the little old lady</td>
</tr>
<tr>
<td>$\text{AP} \ N$</td>
<td>little old lady</td>
</tr>
<tr>
<td>$\text{AP} \ A \ N$</td>
<td>little old lady</td>
</tr>
<tr>
<td>$\text{AP} \ A \ A \ N$</td>
<td>little old lady</td>
</tr>
<tr>
<td>$\text{AP} \ A \ A \ A \ N$</td>
<td>little old lady</td>
</tr>
</tbody>
</table>

Handling left-recursion

- Don't write left-recursive grammars. E.g. instead of:
  - $NP \rightarrow \text{Det} \ \text{Noun}$
  - $NP \rightarrow \text{NP} \ PP$

  write:
  - $NP1 \rightarrow \text{Det} \ \text{Noun}$
  - $NP \rightarrow NP1 \ PP$s
  - $PPs \rightarrow PP \ PP$s

- Left recursive rules can be automatically translated into rules that are not left recursive (accepting the same strings, but the structures returned will be different).
- Impose a limit on the depth of recursion attempted by the parser (number of goals allowed in the list).
- Use a different type of parser (e.g. bottom-up)

Bottom up - shift reduce

- Look for combinations of items that correspond to the right hand side of a rule, and replace them with the left hand side of the rule
- Keep doing this
- As for top-down, there may be alternatives, and picking the wrong one may lead to a blind alley
States of a shift-reduce recogniser

The recogniser manipulates the combination of

- Complete Phrases
- Remaining Words

- We start with Remaining Words
- We finish with Complete Phrases

- At any point, 2 actions may be possible (shift & reduce):
  1. Complete Phrases Remaining Words
     if Cat \( \rightarrow \) Word is a rule in the grammar.
  2. Complete1, Complete2 Remaining Words \( \rightarrow \) Complete1, Cat Remaining Words
     if Cat \( \rightarrow \) Complete2 is a rule in the grammar.

Example recognition

<table>
<thead>
<tr>
<th>Complete Phrases</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>likes Mars</td>
</tr>
<tr>
<td>NP</td>
<td>likes Mars</td>
</tr>
<tr>
<td>NP V Name</td>
<td>Mars</td>
</tr>
<tr>
<td>NP V NP</td>
<td></td>
</tr>
<tr>
<td>NP VP</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
</tr>
</tbody>
</table>

Problem: Empty rules

If we allow "empty" rules, where there are no symbols on the RHS (permitted in some variants of Context Free Grammar) then pure bottom-up parsing is unsuitable.

NP \( \rightarrow \) NP1 PPs
PPs \( \rightarrow \) PP PPs
PPs \( \rightarrow \)

When to "reduce" using the last rule?

Shift-reduce parsing only makes sense if all rules have at least one symbol on the RHS.

Solutions

- Don't write rules with empty RHS's.
- Grammars with empty rules can be automatically translated into grammars without them (accepting the same strings, but, of course, the structures returned will be different).
- Impose a limit on the number of empty phrases hypothesised by the parser?
- Use a different type of parser (e.g. top-down)

Left corner recognition

- A variety of bottom-up recognition that has some elements of top-down
- Find something that is the first element (only) of the RHS of a rule
- Hypothesise that this rule applies
- Recognise following constituents bottom-up
- Notice when they correspond to what is expected from the hypothesised rule
- The hypothesised rule allows some reduction in the bottom-up possibilities considered
Most recognition algorithms can be improved by simple pruning of alternatives that, given the context, can't work out:

- Top-down recognition: Don't hypothesise a goal unless the next word is of a category that could start that goal.
- Left-corner recognition: When a word is used bottom-up to suggest a rule, check that the next word is indeed in a category that can start the next element of the RHS.

Information about what categories can possibly start another category can be pre-computed and stored in a table.

Which rules should we consider?

| S = NP VP              | (e.g. Phoebe likes Mars) |
| S = VP                | (e.g. Open the door) |

Name can start NP but not VP

Verb can start a VP, but Noun cannot

All approaches to recognition involve search:

- Top down – when there is more than one rule with the same LHS
- Bottom up – when there is more than one rule with the same RHS (includes lexical rules)

Sometimes one can choose one or other approach according to the characteristics of one’s grammar (also left-recursion, empty rules)

Default parsing strategy is top-down recursive descent, but left corner is also provided

- Use `set_parser_type(lc)` to switch to “left corner” parsing
- Search strategy is depth-first backtracking search (inherited from Prolog)
Postgraduate Open Day
Tuesday 11th November

Open 1pm—6pm
Elphinstone Hall, King’s Campus

Talks throughout the afternoon on:
- Benefits of PhD Study
- Postgraduate Funding
- Admissions Unwelled
- What is a PhD?
- Studying as a PG at Aberdeen (GSA)

Subject Fair in Elphinstone Hall all afternoon—come and find about the possibilities of Postgraduate study (both taught and research)