Ambiguous Grammars and Languages

- A BNF grammar is ambiguous if its language contains strings for which there is more than one parse tree.
- If all BNF's for a language are ambiguous then the language is inherently ambiguous.

Example: Ambiguous Grammar

0 + 1 + 0

```
<Sum> + <Sum>
  <Sum> + <Sum>
    <Sum>   0
    <Sum>   0
      0      1
      1      0
```

Example

What is the result for:

3 + 4 * 5 + 6

Possible answers:

- 41 = ((3 + 4) * 5) + 6
- 47 = 3 + (4 * (5 + 6))
- 29 = (3 + (4 * 5)) + 6 = 3 + ((4 * 5) + 6)
- 77 = (3 + 4) * (5 + 6)

Example

What is the value of:

7 – 5 – 2
Example

What is the value of:

\[ 7 - 5 - 2 \]

Possible answers:
- In Pascal, C++, SML assoc. left
  \[ 7 - 5 - 2 = (7 - 5) - 2 = 0 \]
- In APL, associate to right
  \[ 7 - 5 - 2 = 7 - (5 - 2) = 4 \]

Example

Ambiguous grammar:

\[
\text{<exp>} ::= 0 \mid 1 \mid ( \text{<exp>} ) \\
| \text{<exp>} + \text{<exp>} \\
| \text{<exp>} * \text{<exp>}
\]

Strings with more then one parse:
- \[ 0 + 1 + 0 \]
- \[ 1 * 1 + 1 \]

Sources of ambiguity here: associativity and precedence

Two Major Sources of Ambiguity

- Lack of determination of operator precedence
- Lack of determination of operator associativity
- Not the only sources of ambiguity

Operator Precedence

- Operators of highest precedence get arguments first (bind more tightly).
  - This generally means evaluated first

Precedence for infix binary operators given in following table

Needs to be reflected in grammar

Precedence Table - Sample

<table>
<thead>
<tr>
<th></th>
<th>Fortan</th>
<th>Pascal</th>
<th>C/C++</th>
<th>Ada</th>
<th>SML</th>
</tr>
</thead>
<tbody>
<tr>
<td>highest</td>
<td>**</td>
<td>*, /</td>
<td>**</td>
<td>**</td>
<td>div, mod, /, *</td>
</tr>
<tr>
<td></td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>div, mod, /, *</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*, /, mod</td>
<td></td>
<td></td>
<td>* /, mod</td>
<td>* +, -</td>
</tr>
<tr>
<td></td>
<td>+, -</td>
<td>+, -</td>
<td>+, -</td>
<td>+, -</td>
<td>+, -</td>
</tr>
</tbody>
</table>

Disambiguating a Grammar

- Given ambiguous grammar G, with start symbol S, find a grammar G’ with same start symbol, such that
  \[ \text{language of G} = \text{language of G’} \]
- Not always possible
- No algorithm in general
Disambiguating a Grammar

- Idea: Each non-terminal represents all strings having some property, its language
  - Each rule describes a sublanguage
  - Identify these properties (often in terms of things that can't happen)
  - Use these properties to inductively guarantee every string in language has a unique parse

Steps to Grammar Disambiguation

- Identify the rules and a smallest use that display ambiguity
- Decide which parse to keep; why should others be thrown out?
- What syntactic restrictions on subexpressions are needed to throw out the bad (while keeping the good)?
- Add a new non-terminal and rules to describe this set of restricted subexpressions (called stratifying, or refactoring)
- Characterize each non-terminal by a language invariant
- Replace old rules to use new non-terminals
- Rinse and repeat

How to Enforce Associativity

- Have at most one recursive call per production

  - When two or more recursive calls would be natural, leave right-most one for right associativity, left-most one for left associativity

Example

- `<Sum> ::= 0 | 1 | <Sum> + <Sum> | (<Sum>)`
- Becomes
  - `<Sum> ::= <Num> | <Num> + <Sum>`
  - `<Num> ::= 0 | 1 | (<Sum>)`
- `<Sum> + <Sum> + <Sum>`

Precedence in Grammar

- Higher precedence translates to longer derivation chain
- Example: * higher than +, both assoc left
  - `<exp> ::= 0 | 1 | ( exp )`
  - `<exp> + <exp> | <exp> * <exp>`
- Becomes
  - `<exp> ::= <mult_exp>`
  - `<exp> + <mult_exp>`
  - `<mult_exp> ::= <id> | <mult_exp> * <id>`
  - `<id> ::= 0 | 1 | ( <exp> )`

Many other sources

- Many other sources
- Can apply same general approach
- Need insights into cause
- Need insights into restrictions to solve
- No general algorithm
- Process:
  - Stratify
  - Prove sublanguages disjoint
  - Prove union of new sublanguages give old language
- Method: Invariants and Induction