$LL(k)$: Parsing With More Than A Single Symbol of Lookahead

1. Review of LL, top-down parsing
2. Example $LL(1)$ conflict
3. Left-factoring
4. Resolving $LL(k)$ conflicts with LL(k+1)
5. $LL(k)$ parsing in (...)+, (...)* loops
6. Minimal use of lookahead
7. Why does ANTLR go "off to lunch" sometimes?

1. Review of LL, top-down parsing

- LL is goal oriented.
- 1-to-1 relationship between grammar, parser state.
- Recursive-descent is preferred over automaton.

Example:

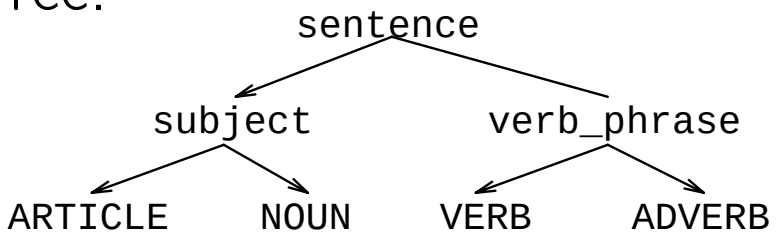
```
sentence    : subject verb_phrase ;

subject     : {ARTICLE} NOUN ;

verb_phrase : VERB {ADVERB} ;
```

Input: `The dog ran quickly`

Parse Tree:



2. Example $LL(1)$ conflict

Simple:

```
stat : ID ":" stat
     | ID ":=" expr
     ;
```

Less obvious:

```
a : {ID} ID ;
```

3. Left-factoring

```
stat : ID ":" stat
     | ID ":=" expr

     ;
```

becomes

```
stat : ID
       (   ":" stat
       |   ":=" expr
       )

     ;
```

How to left-factor this?

```
stat:   ID ":" stat          /* statement label */
    |   expr ";"             /* assignment stat */
    ;
expr:   ID ":=" expr
    |   INT

    ;
```

4. Resolving $LL(k)$ conflicts with LL(k+1)

```
operands
    :   ID
    |   REGISTER
    |   REGISTER "," NUM
    |   REGISTER "," REGISTER "," REGISTER

    ;
```

This is $LL(3)$, but not $LL(2)$.

5. $LL(k)$ parsing in (...)+, (...)* loops

Loop termination: examines what follows loop.

```
a : (A B)* A C ;
```

Will use $LL(2)$ termination decision:

```
while ( LA(1)==A && LA(2)==B ) { ... }
```

6. Minimal use of lookahead

```
operands
    :   ID
    |   REGISTER
    |   REGISTER "," NUM
    |   REGISTER "," REGISTER "," REGISTER
    ;
```

- To distinguish between alts 1 and 2,3,4: $LL(1)$
- To distinguish between alts 2 and 3,4: $LL(2)$
- To distinguish between alts 3 and 4: $LL(3)$

7. Why does ANTLR go "off to lunch?"

- Computing $LL(k > 1)$ lookahead sets is exponentially complex in k and the size of the grammar; size of one lookahead set in the worst case is $O(|T|^k)$.
- ANTLR goes to great lengths to avoid computing full $LL(k)$ lookahead sets; e.g., it uses linear approximate lookahead—$LL^1(k)$—whenever possible.
- The larger the grammar the higher the likelihood that you'll hit a "landmine".
- Ways around it: use a syntactic predicate to turn off the analysis; try lowering lookahead requirements.
- Those with iron stomachs can read my dissertation for more info.