Lookahead $k > 1$ in *LL* and *LR* Translators

Petr Zemek

Brno University of Technology
Faculty of Information Technology

December 17, 2009

LL Parsers and Grammars

LL parser

- **Top-down** parser.
- It parses the input from **L**eft to right, and constructs a **L**eftmost derivation of the sentence.

LL grammar

- Grammar, on which some LL parser can be based.

LR Parsers and Grammars

LR parser

- **Bottom-up** parser.
- It parses the input from **L**eft to right, and constructs a (reverse of) **R**ightmost derivation of the sentence.

LR grammar

- Grammar, on which some LR parser can be based.

Lookahead

Lookahead

- The number of input tokens, which a parser use to decide which rule it should use.
- Normally, we use lookahead of size 1.

	id +	id *	id id	(id	...
A	2	3		7	
...					

Figure: Example of an $LL(2)$ table.

An LL (LR) parser is called an $LL(k)$ ($LR(k)$) parser if it uses lookahead of size k when parsing a sentence.

Recognizers $\times$ Translators

Recognizer

- Given a source code and a grammar, can this code be generated by this grammar?
- Answer: Yes or No.

Translator

- Translates source code defined by some grammar into an equivalent target code.
- More than just a recognizer.

Recognition Power

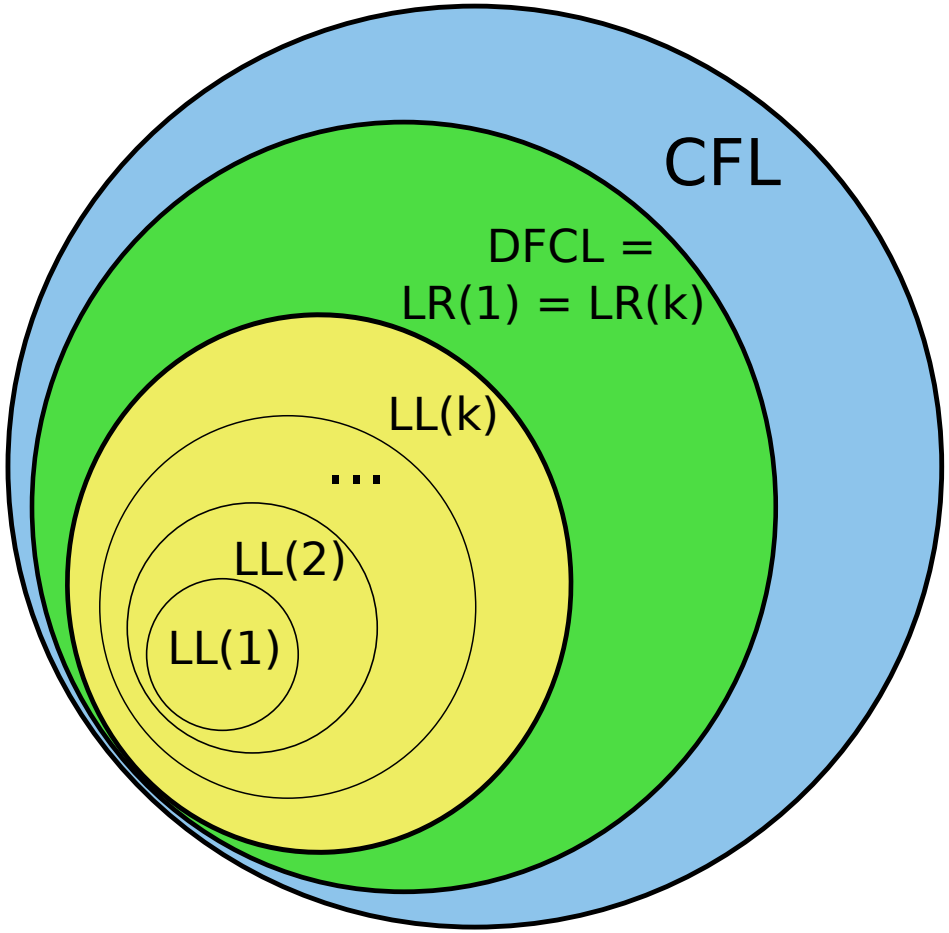

Figure: Hierarchy of language families.

Why Use $k > 1$?

LL and *LR* translators with lookahead $k = 1$ has been almost exclusively used because of the following claims:

- Transformation techniques (e.g. factorization) can be used.
- $LR(1)$ equals $LR(k > 1)$ in recognition power.
- $k > 1$ is not plausible (space and time requirements).

Problems:

- The first claim is often impractical.
- The second claim is not true in case of translators.
- The third claim is outdated.

Claim 1: Grammar Transformation

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

Figure: $LL(2)$ grammar for a fragment of the C language.

It could be transformed into an equivalent $LL(1)$ grammar using factorization, but:

- $LL(2)$ grammar is more convenient – where to put semantic actions in the transformed grammar?
- It can be practically implausible, because *expr* occurs throughout the grammar.

Claim 2: $LR(1)$ Equals $LR(k > 1)$

Semantic actions can decrease the power of translators based on LR parsers.

```
start:   {printf("X ahead");} A X
       |   A Y
       ;
```

Figure: $LR(2)$ grammar, which is not $LR(1)$ (due to actions).

In worst case:

$$LL(1) = LR(1) \subset LL(2) = LR(2) \subset \cdots \subset LL(k) = LR(k)$$

However, this do not often happen in practice.

Claim 3: Space and Time Requirements

Is lookahead $k > 1$ plausible in practice?

- In theory, storing full lookahead information for one decision requires $O(|T|^k)$ space, where $|T|$ is the number of token types.

It was not plausible earlier, but it can be today:

- More available memory, faster processors.
- Various techniques and heuristics were developed:
  - Linear-approximate lookahead – $O(k|T|)$

Conclusion

- Recognizers $\times$ Translators
- There are practical needs for $k > 1$ lookahead:
  - Transformation techniques might be impractical.
  - The presence of actions reduces the strength of $LR(k)$ translators.
  - With current computers and heuristic approaches, use of $k > 1$ is feasible.

References

- Terence J. Parr, Russell W. Quong
  LL and LR Translators Need $k > 1$ Lookahead
  ACM SIGPLAN Notices, v.31 n.2, p.27-34, February 1996

- Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman
  Compilers: Principles, Techniques, and Tools SE
  Addison-Wesley, 2006