Parsing considerations

- Other grammar types (SLR, LALR)
- Which parsing approach should we choose, when?
- What about error handling/recovery by the parser?
- What happens if context-sensitive information is needed to eliminate syntax ambiguities?
- Comparisons of left and right recursion in grammars
- What optimizations can be performed?

SLR and LALR grammars

- These produce smaller tables than LR(1), and can be generated for same sets of languages
- SLR make use of restricted grammars that eliminate LR(1) need for lookahead, with resulting reduction in table size
- LALR(1) categorizes some items in a state's set are critical, while others can be derived from the critical ones, so table construction only represents the critical ones

Picking an approach

- Handcrafted vs table driven?
  - Better error handling/optimization vs reliability/easy generation
- Top-down vs bottomup? LL(1), LR(1), LR(k), SLR, LALR?
  - Top-down, recursive descent a better fit for hand coding
  - Grammar choice might be dictated by available skill/toolsets

Error checking and recovery

- Providing good error messages is key role of most compilers
- Ideally, even if an error is found in one statement, the compiler can generate an error message and still proceed to later statements
- FOLLOW is useful here, to identify potential points to resume processing if an error is found in current statement
- Hand-coded recursive descent parsers provide the compiler writer with good opportunities to customize error messages and handling to the current context

Context sensitive ambiguities

- Sometimes a keyword can have two or more different meanings in different contexts:
  - E.g. Suppose in our language the syntax s(i,j) can mean either a function call or an index into a two-dimensional array
- The parser needs context-sensitive information to identify which use is applicable
- Alternatively, the parser can identify the dual form, and wait until context-sensitive analysis to identify which specifically applies

Left vs right recursion

- Top-down parsers rely on right-recursive grammars
- Bottom-up parsers can work with either left or right
- Compiler writer must consider implications when writing the grammar to be used by the compiler
- Left recursion naturally supports left associativity, right recursion naturally supports right associativity
- Rule of thumb: left recursion can give smaller stack depth

Grammar optimization

- Language and implementation-dependent issues to consider
- Can we optimize by reducing the grammar itself?
- Can we optimize by collapsing equivalent rows or equivalent columns in the table?
- Is it more effective to retain the (optimized) tables, or to use them as the basis for a direct-coded approach?