outline

Vincent Foley

February 26, 2015

1 Compiler overview

What is a compiler? An *automated* tool that *translates* programs written in language A into *equivalent* programs written in language B.

Students should be able to describe the general phases of a non-optimizing compiler (use Minilang as a reference).

- Scanner
- Parser
- AST
- Pretty printing
- Type checker + symbol table
- Code generation

2 Scanner

- What is the input of the scanner? **Characters**
- What is the output of the scanner? **Tokens**
- What formalism did we use to specify scanners? **Regular expressions**
- What are the five fundamental regular expressions?
  - Single character: **c**
  - Empty string: ϵ
  - Concatenation: **AB**
  - Alternation: **A|B**
  - Iteration: **A***

- What other operators can we build?
  - Option: **A? = A|ϵ**
  - One or more: **A+ = A(A$\star$)**
- How does a tool like flex match tokens? **Try all the regexes, return the one that succeeds**
- How does flex handle multiple matches?
  - Longest match rule (e.g. var vs variance)
  - First match rule (e.g. keywords come before identifiers)
- How do we make regular expressions executable? **Regex -> NFA -> DFA**

3 Parser

- What is the input of the parser? **Tokens**
- What is the output of the parser? **Syntax tree (concrete or abstract)**
- What formalism did we use to specify parsers? **Context-free grammars**
- What are the four parts of context-free grammars?
  - Terminals (i.e. tokens)
  - Non-terminals (e.g. stmt or expr)
  - Productions (non-terminal -> (terminal | non-terminal)*)
  - Start symbol
- When is a grammar ambiguous? **When at least one sentence has more than one derivation**
- What are two ways we saw to fix ambiguities? **%left, %right, etc. and factoring the grammar**
- What are the two types of parsers we saw in class? **Top-down, bottom-up**
- What is the difference between the two?
  - Top-down: start with the start symbol and work down to the leaves
  - Bottom-up: start with the leaves and build-up to the start symbol
- What do LL(1) and LR(1) mean?
  - Left-to-right parsing, left-most derivation, 1 token of look-ahead
  - Left-to-right parsing, right-most derivation, 1 token of look-ahead
- What's a left-most derivation? Right-most derivation?
  - LM: `IF expr THEN stmt` -> expand `expr` first
  - RM: `IF expr THEN stmt` -> expand `stmt` first

- Show function from a recursive-descent parser

```
def stmt():
    next_tok = peek()
    if next_tok == TOK_ID:
        id = consume(TOK_ID)
        consume(TOK_EQ)
        e = expr()
        consume(TOK_SEMI)
        return astnode(AST_ASSIGN, lhs=id, rhs=e)
    elif next_tok == TOK_PRINT:
        consume(TOK_PRINT)
        e = expr()
        consume(TOK_SEMI)
        return astnode(AST_PRINT, expr=e)
    elif ...
```

- What are the three actions of a bottom-up parser?
  - Shift (move a token from the input to the stack)
  - Reduce (replace the rhs of a production that's on top of the stack with its lhs)
  - Accept
- What sort of conflict in this grammar? **reduce/reduce**

```
%{
%}

%token ID
%start start

%%

start: rule1 | rule2
rule1: ID
rule2: ID

%%
```

- What sort of conflict in this grammar? **shift/reduce**

```
%{
%}

%token ID
%start start

%%
```

```
start: ID ID | rule1 ID
rule1: ID

%%
```

4 AST

- What is a concrete syntax tree? What are its nodes? What are its leaves?
  - CST: the tree that traces the derivation of a parser
  - Inner nodes: non-terminals
  - leaves: terminals
- Can we use a CST to do type checking? Code generation?
  - Yes, but many more node types to handle than may be necessary
- What is an AST?
  - A hierarchical representation of the program
  - Simplifies the analysis

5 Weeder

- What is the role of a weeder? **Reject invalid programs that the parser cannot**
- What kind of programs did our GoLite weeder reject? **Incorrect use of `break` and `continue`, multiple `default` in switch, etc.**
- If a check can be done in the parser and in the weeder, where should we do it? **Wherever is simplest for us**

6 Symbol tables

- What can be stored in a symbol table? **Identifiers and their related information (e.g. type, offset in stack frame, etc.)**
- What data structure is most often used for symbol tables? **Hash tables**
- How do we handle languages with multiple scopes where variables can be redeclared? **Stack of hash tables**
- How do we lookup an identifier? **Search hashtables from top of the stack to the bottom**

7 Type checking

- What is the role of the type checker? **Reject programs that are syntactically correct, but semantically wrong**
- What is the input of a type checker? **AST**
- What is the output of a type checker? **Annotated AST** (AST with type information for the expressions)
- Do declarations have types? **No**
- What do declarations do? **Add information to symbol table**
- Do statements have types? **No**
- Do expressions have types? **Yes**
- Where do we store the types of expressions?
  - Directly in the AST (add extra fields for expressions)
  - In a big hash table mapping expressions to types (SableCC)
- How do we (generally) type check a statement? **Type check the children, use their information to type check self**
- How do we type check an expression? **Type check the children, use their information to type check self *and* assign self a type**
- Example: in Minilang, how do we type check `if EXPR then STMTS1 else STMTS2 endif`?
  - Type check `EXPR`, make sure it is of type int
  - Type check `STMTS1`
  - Type check `STMTS2`
  - If all three parts are OK, the if statement is OK
- How do we type check a block in GoLite?
  - Open a new scope
  - Type check the statements
  - Close the scope

8 Code generation