Compiler Construction
CS 541

Mr
Mr
Prof

Raphael
Rafi

Finkel
Goldstein

www.cs.uky.edu/~raphael/
courses/CS541.html

MultiLab:

ssh pen.cs.uky.edu
cor.
faw.
use
________ specify
________ implement

________ Compiler
________ programmer language

Compiler outputs

pure machine code
augmented machine code
system calls
virtual machine code
interpreted machine JVM
+ portability
+ code size (register assignment)

Output representations

assembler, for later processing
relocatable binary (machine instructions)
+ modular compilation

absolute binary
Structure of a compiler

Program → scanner → parser

#1 symbol table → AST

#2 tokens → parser

#3 parser → semantic check

#4 tree

code generator

assembler code

#5 code generator → code

code: if (a < 39) 2

symbol table

integer literal

identifier

if statement

eexpr

id a

39 int
Special considerations

1) programming language
   pass by name (Algol 60)
   dynamic-sized arrays
   nested name scopes
   anonymous functions
   first-class functions
   iterators (Python generators)
   automatic object reclamation

2) computer architecture
   how many registers?
   cache principle: frequent
   operations should be fast.
   what operations are expensive?
   virtual method dispatch exceptions
Chapter 2: Adding calculator

- Types: integer, float
- Keywords: f, i, p
- Variables: lowercase Roman single letters.
  
  Syntax: Context-free grammar
  Backus-Naur Form (BNF)
Ambiguity in grammars leads to multiple parse trees

\[
\text{if (a)}
\]
\[
\quad \text{if (b)}
\]
\[
\quad \text{else}
\]
\[
S_1 ;
\]
\[
S_2 ;
\]

Scanner:

translate a stream of characters into a stream of tokens.

Stream:

item = peek (stream) : next item \[\text{no modify}\]

item = advance (stream) : next item \[\text{modifies}\]

match (stream, item) ; expects the given item, advances stream, error if not
Token: has a type (like id, sinum) has a semantic value (like 'a', 39)

Choices: reserved words: each one a different type (call them all id), and distinguish in the semantic value (call them all type = reserved)

Hard-wired scanner (for ac)
Alternative: generated scanner

Regular expressions:
Adequate for defining token syntax. Input to automatic scanner generators.

1) any string (including empty)
2) concatenation of 2 reg. exps.
3) alternation of 2 reg. exps.

Example: \( A^* b^* \) alternation
\( AS = \gamma 1 \times Ka AS \) concatenation
Formal-Language Hierarchy

Use
- Tokens (scanner)
- Syntax (parser)

Language Type
- Regular
- Context-free
- Context-sensitive
- Typed

Formalism
- Regular expressions
- BNF
- C S grammar
- Post productions

Automaton
- Finite-State automaton
- Push-down automaton
- Linear-bounded
- Turing machine

Parser

\[
\text{stream of tokens} \rightarrow \text{parser} \rightarrow \text{abstract syntax tree}
\]

Many approaches
- Recursive descent (LL(1))
- LL(3)

\[
f \ b \ i \ a \ c = 5 \ b = c + 3 \ a \ p \ b \ S
\]

Program needs:
- Dcl Dcls
  - *Motor* id
  - Dcl Dcls
  - *Initial* id
- Structs
resolve overloaded operator

\[(a + b) \cdot (c + d)\]

+ Float, String

package class Field class Field

disambiguate the meanings of some constructors

enforce type rules

Limitations of scanner + parser
Abstract Syntax Tree (AST)

Program

Decls

Stmts

class Decl

Assign

Print
class:

Stmt

Expr

Decl

type (for $i$)

name (like $b$)

Stmt list

Decls list

Program

stmts: Stmts

decls: Decls
Semantic Analysis

1) Symbol table to record types and name scopes. (ST)

   For arr, trivial: array indexed by 'a' ... 'z'
   element has type: unknown, int, float

2) Recursive walk through AST invoking check() on each node in tree.

   Modifying the ST during declarations
   Updating the AST as needed for type conversion
   Updating the type field in expressions.
   Checking that types are consistent.
   Checking other semantic constraints: reachability
   exception consistency.

Code generation

1) Recursive walk through AST invoking codeGen() on each node

Scanner: table-driven (as opposed to hard-coded)
Token specification: regular expressions

\[ \emptyset \] : no valid strings

\[ \Lambda \] : empty string

\[ \Sigma \] : a string containing the letters in alphabet \( \Sigma \)

Concatenation of 2 REs (no symbol)

Cross-product: one from each alternative of 2 REs (symbol \( \mid \) )

Resulting strings: union of two sets.

Closure operation:

\[ \ast \] : 0 or more concatenations

\[ + \] : 1 or more concatenation

Parentheses for grouping

Escape convention for metachars. Typically \( \backslash \)

Examples:

Alphabet \( \Sigma = \{ a, b, c \} \)

\( \emptyset \) : set of strings

\( \Lambda \) : empty

\( \Sigma \) : \{ "", " \}
Useful facts

- The set of strings specified by a RE is a regular set.
- Every finite set of strings is a regular set.
- Every regular set has multiple REs that describe it.

Example:

- $(ab)^* = \varepsilon^*, ab, abab, \ldots$
- $\exists \set{ab | abab(ab)^*}$

Notation:
- If $A$ is a set of characters, $\text{not}(A)$ refers to the other characters.
- If $S$ is a set of strings, $\text{not}(S)$ is also regular.

$S^k = SSSS\ldots S$
String: element of the language specified by a RE.

Examples

Java comment: //
RE: // (not \n)* \n\nmetachars

Decimal literal
D = 0123 ... 9
D+. D+ \equiv DD*. DD*

Integer literal, optionally signed
\n\nExample: \# hi \# there \#

Fortran-like real literals must have digits in at least one side.

Example: \( D+ . D* \) | (D*. D+) 4.7
\[(p^+) | (d^+) | (d^+ d^+)\]

\[(p^+) \lor (d^+) \lor (d^+ d^+)\]

Identifier, with - chars, but not adjacent - not final - not first char may not be a digit.

\[L \text{ Letter, D digit}\]

\[L (L | D)^* \lor (L | D)^*\]

Mickler: \[L ((-1)^2 L L D)^*\]

Hashing: \(O(1)\) searching and insertion

(actually \(O(\log n)\), have to look at entire key)

Java provides an interface \(Map<K, V>\)

\[
\text{Map}\langle\text{String, String}\rangle\text{ myHash} = \\
\text{new HashMap}\langle\text{String, String}\rangle();
\]

aside \[
\text{List}\langle\text{Integer}\rangle\text{ myList} = \\
\text{new LinkedList}\langle\text{Integer}\rangle();
\]
Hashing: given a key, apply a function \( h(k) \) returning an integer. Use that integer to index an array. Store/lookup that \( (k,v) \) pair at that location in the array.

Collisions: multiple keys can have same index.

Example: birthday paradox.

Resolution:

open addressing: find another place
external chaining: array points to a secondary structure, typically a list.

Finite-state automata (FSA)

Simple computer

Finite set of states: circles

one is start state.

one or more states are final (accepting)

transitions between states labeled by letters in \( \Sigma \)
\[
x_1 = x_0 \land y_0 \\
y_1 = x_1 \land y_0 = x_0 \land y_0 \land y_0 \\
x_2 = x_1 \land y_1 = x_0 \lor y_0 \lor y_0 
\]

grading assignments
look for lines: //grading:
file: grade
returned file is a ZIP file

Nice Java features

1. for (var : collection) {
2.   \[\text{instance of a class that implements the Collection interface.}\]
3. } switch can take string conditions.
switch (String expr) {
  case "foo": break;
  default: 
}

Short story: Stickers are described by REG.

1) REGs are encoded into non-deterministic finite-state automata (NDFSA).
2) NDFSA are converted to DFA (deterministic).
3) The DFA are described by tables.
4) A simple program can run the DFA.

Shorter story:
1) write a set of REGS describing tokens.
2) let a scanner generator (flex) build the tables.
3) let a pre-built scanner run the tables.

Complexities (if strange tokens):
1) escaped " within a string literal.
2) operator-eagerness
   Pascal: `..`
   Ada: `a..b`
   FORTRAN: `DO 200 I = 1..2`
built by an automatic process: lex, flex, jflex
alternative: automatically build a hard-wired scanner. Likely to be faster.
Transducer: recognizes strings in a regular language and outputs some semantic value.

Natural-language processing:

- **Noun** → lexeme (noun) → table → lexeme (plural)
- **Verb** → lexeme (verb) → table

3rd sg present

Each transition is associated with an action. (build a growing string, guess at meaning, construct value of a literal)

**Scanner generators** (*lex, flex, jflex*)
- **Input file**: defines tokens and actions
- **Output**: a table (embedded in code) and associated code:
  - `yylex()`: `jflex`...

Compile and link that output with a client (*P2jam*)
- **Driver, parser**
- **Input file** (*jflex*)
  - Subroutines
  - Declarations
  - 95%
  - 92%
  - REs and actions

Shared with client:
- Token codes (small integers)
Although:
\[ \text{list < ...>} \text{ var } = \text{LinkedList < } \uparrow \text{ diamond} \]

Sometimes:
\[ \text{LinkedList < ...>} \text{ var} \]
\[ \text{// need push/for when you need methods in implementation type not available in interface type} \]
\[ ((\text{LinkedList}) \text{ var}), \text{push}(\ldots) \]

\[ \text{lex} \]

\[ \text{declarations:} \]
\[ \text{short hands for RES,} \]
\[ \text{DIGIT } = 0123 \ldots 9 \]
\[ [0-9] \]
\[ [013-9] \]
\[ \text{VAR } = [a-e, gh, j, 08, 2] \]
\[ F=0 \rightarrow [aa \text{b}] \]
\[ [\wedge \text{a} \text{b}], \equiv \Sigma - [\text{a} \text{b}] \]
\[ \text{ALLCHARS} = \{ \wedge \} \in \Sigma \text{ equiv} \]
Foo = "abc"

abc \equiv \{ \text{equivalent} \}

"abc" \equiv \{ \text{equivalent} \}

"[abc]" \equiv \{ \text{equivalent} \}

Case is significant

ignorecase \equiv \text{makes all lower = upper.}

\{a\} \equiv \{aA\}

\{bB\} \equiv \{cC\} \equiv \{iI\} \equiv \{nN\}

\text{metacharacters}

Rules

- D \equiv { \text{return} (\text{NL1IT}). \}

processing code may invoke \text{yylex()} to a string value containing what was matched by RE.
$D \oplus \cdot D + \frac{3}{5} \leq$

\[
\begin{array}{l}
\} \\
\text{return \{FLOAT_LIT\}} \\
\} \\
\"\{\} \} \text{return \{OPENBRACE\}}
\end{array}
\]

JFlex creates: Jglex.jar

javac -classpath ./classes:Jglex.jar *java

Provided by JFlex

Advice:

Last rule

$\frac{3}{5} \text{ return \{ERROR\}}$

$\frac{3}{5} \text{ DF taken is automatic, (taken value = 0)}$

Subroutine section

introduce declarations of vars and functions, and classes.

"A diagnostic" : a compiler-generated error message.

Handle Pascal's 3.4

$$D \mid D/".\cdot" \frac{3}{5} \text{ int lit}$$

Even if looking ahead has 2 dots.
If there is an error in input (not less)
entered an error state.
backup until the scanner is in a final
if all the way back, discard 1 character,
diagnostic, continue,
example (5x_90) : N ^ ^

If not Len : GLA, readc : seq table-driven
scanner,

Identifiers
1) if not block-structured, scanner enter ids into ST.
2) if block-structured, copy the string in “string space”
   and return a pointer.

String space

    Hoobar

    ↑     ↑
simpler: duplicate String values.

3) scanner to enter in at ST that is block-structured
   controlled by parser.