CSE443 Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall
Team formation

- Please sit with your teams starting next week.

- C vs SML?
Phases of a compiler

Figure 1.6, page 5 of text
Formally, a grammar is defined by 4 items:

1. $N$, a set of non-terminals
2. $\Sigma$, a set of terminals
3. $P$, a set of productions
4. $S$, a start symbol

$G = (N, \Sigma, P, S)$
languages & grammars

N, a set of non-terminals
Σ, a set of terminals (alphabet)

\[ N \cap \Sigma = \{\} \]

P, a set of productions of the form (right linear)

- \( X \rightarrow a \)
- \( X \rightarrow aY \)
- \( X \rightarrow \varepsilon \)
- \( X \in N, \ Y \in N, \ a \in \Sigma, \ \varepsilon \text{ denotes the empty string} \)

S, a start symbol

\( S \in N \)
Lexical Analysis

- Lexical structure described by regular grammar
- Deterministic finite state machine performs analysis
If L and M are regular, so are:

- $L \cup M = \{ s | s \in L \text{ or } s \in M \}$ **union**
- $LM = \{ st | s \in L \text{ and } t \in M \}$ **concatenation**
- $L^* = \bigcup_{i=0,\infty} L_i$ **Kleene closure**

By definition, $L^0 = \{ \epsilon \}$
Given an alphabet $\Sigma$

**REGular EXpression (regex)**

**Inductive definition**

$\varepsilon$ is a regex

$L(\varepsilon) = \{\varepsilon\}$

For each $a \in \Sigma$, $a$ is a regex

$L(a) = \{a\}$
Regular expressions (regex)
Inductive definition

Assume $r$ and $s$ are regexes.

$r|s$ is a regex denoting $L(r) \cup L(s)$
$rs$ is a regex denoting $L(r)L(s)$
$r^*$ is a regex denoting $(L(r))^*$
$(r)$ is a regex denoting $L(r)$

Precedence: Kleene closure $>$ concatenation $>$ union
Associativity: all left-associative (minimize use of parentheses: $(r|s)|t = r|s|t$)
Assume \( r \) and \( s \) are regexes.

**Commutativity** \( r|s = s|r \)

**Associativity** \( r|(s|t) = (r|s)|t \) and \( r(st) = (rs)t \)

**Distributivity** \( r(s|t) = rs|rt \) and \( (s|t)r = sr|tr \)

**Identity** \( \varepsilon r = r \varepsilon = r \)

**Idempotency** \( r** = r* \)
We can describe a regular language using a regular expression
A regular expression can be recognized using a finite state machine.

Machines:
NFA
non-deterministic finite automaton
DFA
deterministic finite automaton
Process of building lexical analyzer

1) spell out the language
Process of building lexical analyzer

2) formulate a regular expression
Process of building lexical analyzer

3) build an NFA
Process of building lexical analyzer

4) transform NFA to DFA
Process of building lexical analyzer

5) transform DFA to a minimal DFA
Process of building lexical analyzer

5) The minimal DFA is our lexical analyzer

language → regex → NFA → DFA → DFA

character stream

token stream

lexical analyzer
Focus for today

regex  →  NFA
Nondeterministic Finite Automata (NFA)

- A finite set of states $S$
- An alphabet $\Sigma$, $\varepsilon \notin \Sigma$
- $\delta \subseteq S \times (\Sigma \cup \{\varepsilon\}) \times \mathcal{P}(S)$ (transition function)
- $s_0 \in S$ (a single start state)
- $F \subseteq S$ (a set of final or accepting states)
Deterministic Finite Automata (DFA)

- A finite set of states $S$
- An alphabet $\Sigma$, $\varepsilon \notin \Sigma$
- $\delta \subseteq S \times \Sigma \times S$ (transition function)
- $s_0 \in S$ (a single start state)
- $F \subseteq S$ (a set of final or accepting states)
NFA

Initial state: arrow from nowhere pointing in. Often labelled state 0.

Final state: drawn with a double circle

Arrows are labeled with $\varepsilon$ or $a \in \Sigma$.

for each $a \in \Sigma$
Regex $\rightarrow$ NFA

For each $a \in \Sigma$
Regex $\Rightarrow$ NFA

$\varepsilon s \Rightarrow (N(s) \cap N(\varepsilon)) 1$

$\varepsilon \Rightarrow (N(s)) 1$

$\varepsilon^* \Rightarrow (N(s)) 1$
Simple example

static
Simple example

static

0 1 2 3 4 5 6
Simple example

static
struct