Announcements

- HW-01 posted
- PR-01 posted
- Team formation: what is current status?
Phases of a compiler

Figure 1.6, page 5 of text
Bird's eye view

- `{ for, while, x, factorial, ... }`
- `G = (N, Σ, P, S)`

Language: a set of strings

Grammar: rules for generating language

Finite automaton: a machine for language

Regular expression: a form of grammar

C program: generated by FLEX
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

\[ N, \text{ a set of non-terminals} \]
\[ \Sigma, \text{ a set of terminals (alphabet)} \]
\[ N \cap \Sigma = \{\} \]
\[ P, \text{ a set of productions of the form (right linear)} \]
\[ X \rightarrow a \]
\[ X \rightarrow aY \]
\[ X \rightarrow \varepsilon \]

\[ X \in N, Y \in N, a \in \Sigma, \varepsilon \text{ denotes the empty string} \]

\[ S, \text{ a start symbol} \]
\[ S \in N \]
Lexical Analysis

- Lexical structure described by regular grammar
- Deterministic finite state machine performs analysis
LANGUAGE operations

base cases

- \{ \varepsilon \} is a regular language
- \forall a \in \Sigma, \{ a \} is a regular language

Recall, \varepsilon is the empty string
LANGUAGE operations

If $L$ and $M$ are regular, so are:

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

$L^i$ is $L$ concatenated with itself $i$ times:
- $L^0 = \{ \varepsilon \}$, by definition
- $L^1 = L$
- $L^2 = LL$
- $L^3 = LLL$, etc.
- $L^*$ is the union of all these sets!
Example of $L^*$

Suppose $L$ is \{a, bb\}

$L^0 = \{\epsilon\}$, by definition

$L^1 = L = \{a, bb\}$

$L^2 = LL = \{aa, abb, bba, bbbb\}$

$L^3 = LLL = \{aaa, aabb, abba, abbbb, bbbaa, bbbba, bbaa, bbabb, bbbba, bbbbbb, abbbb, bbabb, \ldots\}$

$L^4 = \ldots$ and so so...

$L^* = \bigcup_{i=0,\infty} L^i = \{\epsilon, a, bb, aa, abb, bba, bbbb, aaa, aabb, abba, abbbb, bbbaa, bbbbaa, bbbaa, bbabb, bbbbaa, bbbbbbb, abbbb, bbabb, \ldots\}$
Given an alphabet $\Sigma$

REGular EXpression (regex)

Inductive definition

$\epsilon$ is a regex

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

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: $(r|s)|t = r|s|t$)
Algebraic laws

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

language → regex → NFA
Process of building lexical analyzer

4) transform NFA to DFA

language ➔ regex ➔ NFA ➔ DFA
Process of building lexical analyzer

5) transform DFA to a minimal DFA
5) The minimal DFA is our 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)
A state is a circle with its state number written inside.
Initial state has an arrow from nowhere pointing in. State 0 is often the initial state.
A final state is drawn with a double circle.
Arrows are labeled with $\varepsilon$ ...

$\varepsilon$

... or $a \in \Sigma$.

$a$

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

For each $a \in \Sigma$

$S \mid t$
Regex -> NFA

$S^*$
Simple example

static

0 1 2 3 4 5 6
Simple example

static

struct
Process of building lexical analyzer

5) The minimal DFA is our lexical analyzer
Focus above:
build a non-deterministic recognizer
Next step:
make recognizer deterministic
first we construct an NFA
from this regular expression
(a|b)^*abb
\[(ab|b)^*abb\]
$(a|b)^*abb$
(a|b)*abb
(a|b)*abb
\((ab|b)^*abb\)
Operations

- $\varepsilon$-closure($t$) is the set of states reachable from state $t$ using only $\varepsilon$-transitions.

- $\varepsilon$-closure($T$) is the set of states reachable from any state $t \in T$ using only $\varepsilon$-transitions.

- move($T,a$) is the set of states reachable from any state $t \in T$ following a transition on symbol $a \in \Sigma$. 
INPUT: An NFA \( N = (S, \Sigma, \delta, s_0, F) \)

OUTPUT: A DFA \( D = (S', \Sigma, \delta', s_0', F') \) such that \( \mathcal{L}(D) = \mathcal{L}(N) \)

ALGORITHM:

Compute \( s_0' = \varepsilon\text{-closure}(s_0) \), an unmarked set of states

Set \( S' = \{ s_0' \} \)

while there is an unmarked \( T \in S' \)

mark \( T \)

for each symbol \( a \in \Sigma \)

let \( U = \varepsilon\text{-closure}(\text{move}(T,a)) \)

if \( U \not\in S' \), add unmarked \( U \) to \( S' \)

add transition: \( \delta'(T,a) = U \)

\( F' \) is the subset of \( S' \) all of whose members contain a state in \( F \).