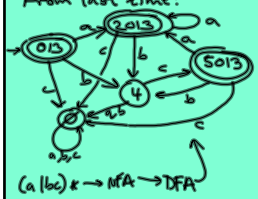
From last time:



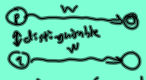
(a|bc)* → NFA → DFA

Optimize this DFA to have the fewest states possible (minimization).

Idea: merge indistinguishable states

Def: Let A be a DFA with transition function δ.

Two states p, q of A are distinguishable if there exist some string w such that reading w starting from p leads to a state with opposite acceptance status from the state led to by starting at q and reading w (one is accepting, the other rejecting).



Algorithmic (equivalent) definition of distinguishability:

States p and q are distinguishable iff either

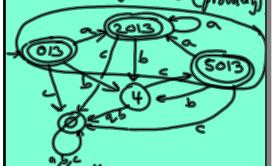
1) one of p & q is accepting and the other is rejecting (distinguished by ε), or

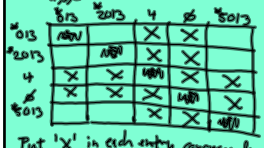
2) There exists a character a such that δ(p,a) and δ(q,a) are distinguishable.

[nothing else]

Apply those two rules as many times as possible to find all pairs of distinguishable states.

Pairs that are left over are indistinguishable (provably)

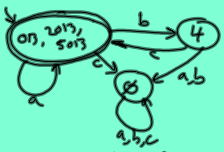




Put 'X' in each entry corresponding to a distinguishable pair.

013, 2013, and 5013 are indistinguishable, so merge them into one state.

Minimum DFA:



Theorem (Myhill, Nerode): this always leads to the unique equivalent DFA with the fewest possible states.

Flex does this for each regexp in the rules section.

How the lexical scanner works:

Rules section:

regexp1	action1
regexp2	action2
⋮	⋮
regexpn	actionn

Jan 31-10:58 AM

Get
DFA$_1$, DFA$_2$, ..., DFA$_n$ } DFA$_i$ is minimal DFA for regexp$_i$

Scanner produced by flex reads the input stream char by char, simulating all DFAs simultaneously, beginning in the start state of each.

Initially, all DFAs are alive.

As reading progresses, a DFA may enter its dead state. Mark this DFA as dead when this happens, and take it out of the simulation.

Also, as reading progresses, any of DFAs, say DFA$_i$, may enter an accepting state. Maintain a pointer p_i into the input stream for each DFA$_i$, marking the most recent spot where DFA$_i$ was in an accepting state. {Whenever DFA$_i$ enters an accept state, update p_i to the current position in the input.

Do this until either all DFAs are dead or EOF reached. When this happens, take the pointer, say p_i, that is furthest (greedy) along in the input. Then the match (yytext) runs from the beginning of the input up to p_i.

Execute corresponding action$_i$. Match is removed from input stream, so any further read starts with the first char following the match.

If action$_i$ has no return then this is repeated until EOF.

Ties among matching rules are resolved by executing the action of the rule that appears first.

Scanning C:

"else" } {return LEX_ELSE;}
"if" } {return LEX_IF;}
"while" } {return LEX_WHILE;}
⋮

[_a-zA-Z][_a-zA-Z0-9] { identifier}

What if nothing matches?

Lex adds a default rule:

.|\n {}

↑ matches any single char

↑ empty action

flex allows lots of optional behavior:
read from multiple files

FILE *yyin } streams that
FILE *yyout } lex uses
*yyrer }

by default, these are stdin, stdout, stderr, respectively. You can change these.

Scanner may be very inefficient (you should avoid situations like this):

Example:

%%
a*b { some action}
%%

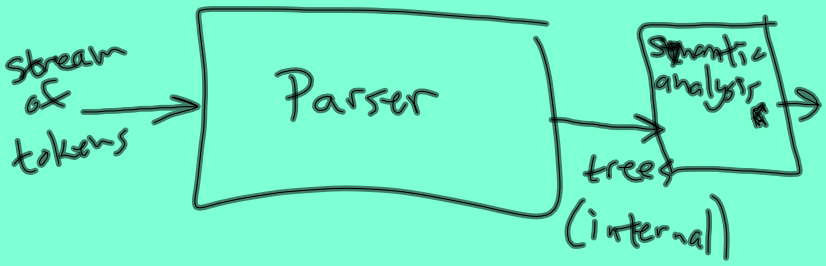
Suppose the input is a string of 10^6 many a's

---

hw1_test.pl

Jan 31-11:33 AM

Syntax Analysis (Parsing)



Parser recognizes hierarchical structures in the source:

syntactic structures nested inside of syntactic structures

(statements within statements
subexpressions within expressions
etc.)

Can't do this with regexp — not powerful enough to recognize nested things.

the set of all strings of balanced parentheses.

There is no regexp that recognizes this set.

Need more power: stack.

Jan 31-12:08 PM