A scanner must recognize various parts of the language's syntax.

Regular expressions (REs).

Legal tokens are usually specified by regular expressions.

Languages are sets of strings.

A scanner must recognize various parts of the language.

Operations include Kleene closure, concatenation, and union.

We assume closure, concatenation, union as the order of precedence.

Input is separated into tokens based on lexical analysis.

Legal tokens are usually specified by regular expressions.
Recognize

From a regular expression, we can construct a deterministic finite automaton (DFA).
This takes many fewer instructions per cycle.

1. Branch directly to next state
2. Test character class locally
3. Classify the input character

We can do better by "encoding" the state table in the scanner code.

To change languages, we can just change tables.

<table>
<thead>
<tr>
<th>char class</th>
<th>digit</th>
<th>letter</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>0</td>
<td>A-V(-)</td>
<td>Z-(_)</td>
</tr>
<tr>
<td>other</td>
<td>0</td>
<td>0</td>
<td>9</td>
</tr>
</tbody>
</table>

Two tables control the recognizer.

Table driven implementation is slow relative to Improved efficiency

Table for the recognizer
Nondeterministic Finite Automata

What about the regular expression $a | abb$?

Nondeterministic Finite Automata

...
What is the relationship between an NFA and a DFA?

- DFA is a special case of NFA.
- No transitions.
- DFA is a special case of an NFA.

Constructing a DFA from a regular expression:

1. No ε-transitions.

Possible exponential blowup.

Simulate sets of simultaneous states.

DFA can be simulated on a DFA.

DFA can be simulated on a NFA.

regular expression (RE) -> NFA

build NFA for each term

connect them with ε moves

nfa moves to nfa, ε moves

nfa -> dfa

construct the simulation

subset construction

the "subset" construction

construct the simulation

nfa -> dfa

construct regular expression (RE) -> nfa, ε moves

nfa moves

What is the relationship between an NFA and a DFA?
Converting regular expressions to NFA (cont.)

<table>
<thead>
<tr>
<th>Operation Description</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(L', a)$</td>
<td>Set of DFA states reachable from some state $s$ in $L'$ on input symbol $a$.</td>
</tr>
<tr>
<td>$(L', a)$</td>
<td>Set of DFA states reachable from some state $s$ in $L'$ on input symbol $a$.</td>
</tr>
<tr>
<td>$(L', a)$</td>
<td>Set of DFA states reachable from some state $s$ in $L'$ on input symbol $a$.</td>
</tr>
<tr>
<td>$(L', a)$</td>
<td>Set of DFA states reachable from some state $s$ in $L'$ on input symbol $a$.</td>
</tr>
</tbody>
</table>

Method: Let $s$ be a state in DFA and $T$ a set of accepting states. The language $T$ accepts the same language that output of a DFA with states and $T$ and same on input $N$. Apply subset construction algorithm.

Compose automata as follows:

- Concatenate
- Kleene closure
- Union

Build two-state automaton for atomic regular expression $a$ with $a$ as the edge.
Subset construction (cont)

state \( \text{Start} = \epsilon\text{-closure}(s_0) \)
add \( \text{Start} \) unmarked to \( D\text{states} \)
while 3 an unmarked state \( T \) in \( D\text{states} \)
mark \( T \)
for each input symbol \( a \) do
\[ U = \epsilon\text{-closure}(\text{move}(T,a)) \]
if \( U \) is not in \( D\text{states} \) then
add \( U \) to \( D\text{states} \) unmarked
\[ D\text{tran}[T,a] = U \]
endfor
endwhile

Each state in \( D \) corresponds to a set of states in \( N \).

Up to \( 2^{|N|} \) possible states in \( D \).

\( \epsilon\text{-closure}(s_0) \) is the start state of \( D \).

A state is an accepting state in \( D \), if one or more of the states it represents in \( N \) is accepting.

Example subset construction

\( \text{nfa for } a^*b \)

\[ e \]

\( 1 \quad 2 \quad 3 \quad 4 \quad 5 \)

\[ \epsilon\text{-closure}( ) = \{ \} = \]
\[ \text{MOVE}( , ) = \{ \} \]
\[ \epsilon\text{-closure}( ) = \{ \} = \]
\[ \text{MOVE}( , ) = \{ \} \]
\[ \epsilon\text{-closure}( ) = \{ \} = \]
\[ \text{MOVE}( , ) = \{ \} \]
\[ \epsilon\text{-closure}( ) = \{ \} = \]

C MSC 430 Lecture 2, Page 13
Building minimum-state dfas

Important theoretical result

*Every regular language is recognized by a minimum-state dfa that is unique up to state names.*

Look for states that can be distinguished from each other (i.e., end up in accepting/nonaccepting state for identical input).

dfa state minimization algorithm

- construct initial partition of states into accepting and non-accepting states
- successively refine partition by splitting a group $G$ into smaller groups if states in $G$ have transitions to different groups
  (two states $x$, $y$ are in same group iff for all input symbols $a$ $x$ and $y$ have transitions to same group)
  - update transition edges, remove dead states

See proof of theorem 3.10, pages 67–71 in Hopcroft and Ullman’s book
*Introduction to Automata Theory, Languages, and Computation*

---

Example minimal dfa construction

dfa for $a^*b$ from nfa

![Diagram](Image)

Initial partition

- accepting = {} =
- non-accepting = {} =

Split groups

\[
\begin{align*}
    \text{(state, input)} &= \text{group} \\
    (S, a) &= \\
    (T, a) &= \\
    (S, b) &= \\
    (T, b) &=
\end{align*}
\]

Minimal dfa
Convertin g

dfa
storegul arexpre
ssions

Method /:

For a dfa

\[ M \]

\[
\begin{align*}
&: /; /: /: /; s
\end{align*}
\]

Let

\[ R_{ik} \]

denote the set of all strings

\[ x \]

such that

\[
\{ s_i \mid x \} \subseteq \{ s_j \}
\]

and if \( y \) is a prefix of \( x \) then

\[
\{ s_i \mid y \} \subseteq \{ s_k \}
\]

for a dfa

\[ M \]

\[
\begin{align*}
&: /; /: /: /; s
\end{align*}
\]

Issues

"lazy" construction of \( dfa \)

"two stack simulation of \( dfa \)

Generate \( dfa \) directly from regular expression

Other approaches

For regular expression \( r \) and input \( x \)

Complexity Tradeoffs

\[
\begin{array}{c|c|c}
         & \text{Time} & \text{Space} \\
\hline
[|a|] & (|a|O) & (|a|O) \quad \text{vfp} \\
[|a| \ast |a|] & (|a|O) & (|a|O) \quad \text{vfu} \\
\hline
\end{array}
\]

See proof of Theorem 2.4, pages 33-34 in Hopcroft and Ullman's book

Introduction to Automata Theory, Languages, and Computation

C MSC 430

Lecture 2, Page 17

C MSC 430

Lecture 2, Page 18

Case Studies

Method:

Converting dfa to regular expressions

\[
\begin{align*}
\{ e \} \cap \{ s_i = (a_1 \cdots a_n) s_j \} &= f_{i,j} \quad \text{(a)} \\
\{ s_i = (a_1 \cdots a_n) s_j \} &= f_{i,j} \quad \text{(b)} \\
\{ f_{i,j} \cap (f_{i,j} \cup (f_{i,k} \cap f_{k,j})) \} &= f_{i,j} \quad \text{(c)} \\
\end{align*}
\]

More formally

\[
\begin{align*}
\left( \bigcap_{i,j} f_{i,j} \right) = (\bigcap_{i,j} f_{i,j})
\end{align*}
\]

Then, |

"thorough" means both entering and leaving \( s_i \) \n
\( q < 1 \)

state \( s_i \) before \( s_j \)

state \( s_j \) without going through a

state \( s_k \) where \( k \)

\leq 1 \)

\( s_k \) that take \( M \) from \( s_i \) to \( s_j \)

\( s_k \) such that that

(\( s_0, s_1, \ldots, s_n \) )

for a dfa

Method:

\[
\begin{align*}
( s_0, s_1, \ldots, s_n ) = M
\end{align*}
\]
What is a lexical error?

1. Reserved words in languages like FORTRAN and ALGOL ignore blanks
2. Illegal characters in strings
3. Do/10=1.25
4.reserved words
5. if then then else
6.非法字符

Language features that can cause problems:

Reserved words

Illegal characters in strings

Do/10=1.25

What is a lexical error?

Reserved words

Illegal characters in strings

Do/10=1.25

So what is hard?
Summary

Scanners

- Highly efficient in practice
- Constructed to recognize language
- Tokens specified by regular expressions

Scanner Generators

- Error recovery
- Lookahead
- Input buffering

Issues

- Difficulty affected by language design
- Catch lexical errors
- Break up input into tokens

Example due to Dr. D. F. Zadeck of IBM Corporation

```c
  14 END
  $ FILE(1)
  C
  13 END
  CONTINUE
  12 FORMAT(300,200)
  11 IF(X)H=1
  10 IF(X)=1
  9 D09E1=I,2
  8 D09E1=I
  7 D09E1=(3)
  6 200 FORMAT(4)=(3)
  5 100 FORMAT(4)=(3)
  4 INTEGER FORMAT(10),IF(10),D09E1
  3 IMPLEMENT CHARACTER(A-B)(A-B)
  2 INTEGER FUNCTION
  1 HOW BAD CAN IT GET?
```