

How flux converts
regexps to DFAs

Basic Idea

regexp R to start

$R \rightarrow N \rightarrow D$

↑
nondeterministic
FA

DFA

A nondet. FA (NFA)
is just like a DFA except
no determinism restriction

Also allow edges labeled
with ϵ (the empty string)

can
transition from s to t
without advancing the
input

Given an input string x
an NFA N accepts x iff

Feb 2-11:00 AM

... iff there is some
path from start state
to an accept state
reading x in its entirety

regexp $\rightarrow$ NFA

$R \mapsto N$

such that for any input
string x ,

x matches $R \iff N$ accepts x

Translation is defined
recursively on the
structure of R .

Recall regexps can
be built up from

$\emptyset$ ϵ a

atomic regexps

using $|$ (or)
concatenation, or
 $*$ (Kleene closure)

Feb 2-11:07 AM

Translate (R)

// R is some regexp
// output is an NFA
// equivalent to R

Basic cases:

1. $R = \emptyset$.
output $\rightarrow \emptyset$
2. $R = \epsilon$.
output $\rightarrow$ (NFA with two states, start and accept, connected by an ϵ transition)
better: $\rightarrow$ (NFA with one state, start and accept, self-loop on ϵ)
3. $R = a$ (some ASCII char a).
output $\rightarrow$ (NFA with two states, start and accept, connected by an a transition)

Recursive cases:

4. $R = S|T$ where
 S, T are regexps.
Recurse on S and on T .
Let N_S be output of Translate(S)
be output of Translate(S)

Feb 2-11:13 AM

Let N_S be output of Translate(S)
Let N_T be output of Translate(T)

$[R = S|T]$ output N

5. $R = ST$ where
 S, T are regexps.
 $N_S = \text{Translate}(S)$ $N_T = \text{Translate}(T)$

Insert ϵ -trans from all accept
states of N_S into start state of N_T
yielding output NFA N
also form accept states
of N_S into reject states

6. $R = S^*$ For some
regexp S .
 $N_S = \text{Translate}(S)$

Output is NFA (equivalently
make accept states of N_S into reject states)

Feb 2-11:25 AM

Concludes def of

$R \mapsto N$

Translate

Converting an NFA N
into an equivalent DFA D .

Convert(N) = D .

Idea: in N :

F is an NFA N
with state set Q ,
start state $q_0 \in Q$,
and accept state set $F \subseteq Q$

Def: Let $s \in Q$ be
some state. The
 ϵ -closure of s
($\epsilon\text{-cl}(s)$) is the
set of all states reachable
from s using zero or more
 ϵ -moves. (includes s)

Feb 2-11:39 AM

Def: If $T \subseteq Q$ is
a set of states, then
define
 $\epsilon\text{-cl}(T) := \bigcup_{s \in T} \epsilon\text{-cl}(s)$

Def: A set U of states
is ϵ -closed if
 $U = \epsilon\text{-cl}(T)$ for some
 $T \subseteq Q$.

equivalently, you cannot
go from inside U to
outside U by following
 ϵ -moves.

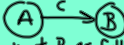
Construct(N) = D

// states of D will be
sets of states of N

start state of D :
 $\epsilon\text{-cl}(q_0)$

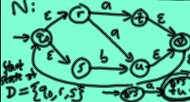
start state
without reading any of the input

Feb 2-11:50 AM

Call this state Q_0 .
 Insert Q_0 onto an empty queue.
 While queue is not empty, do
 {
 remove some $A \in Q$ from the queue
 // A is one of D 's states we've constructed
 for each ASCII char c , do
 
 construct B as follows:
 for all
 $B = \epsilon\text{-cl}\left(\bigcup_{s \in A} \delta(s, c)\right)$
 states of N reachable from s via a c -transition
 states reachable from A following a transition then some number of c 's
 = set of states reachable from states in A by reading c on input.
 }
 Note: all constructed states are ϵ -closed.

Feb 2-11:57 AM

add transition
 $A \xrightarrow{c} B$
 to D , where B is defined above.
 If B is a new state then insert B into the queue.
 end-for each
 end-while
 // constructed start state, all states, and all transitions of D .
 Say that a state T of D ($T \in Q$) is accepting iff
 $T \cap F \neq \emptyset$
 (T contains at least one accepting state of N)
 end construct.

N :

 that start of $D = \{q_0, q_1\}$

Feb 2-12:07 PM