Compiler Construction: Principle and Practice

Regex To DFA

Regex to NFA:

- Concatenation: ε-transition.
- Alternative: New start state, ε-transitions.
- Repetition: Direct ε-transition **to accepting state and backward**.

NFA to DFA (subset construction):

1. ε-closure of start state as start state.
2. For any set of states S and character a, compute set $T_a = \{t \mid s \xrightarrow{a} t, s \in S\}$, and add the transition from S to the ε-closure of T_a on a.

Better use tables instead of drawing the diagram directly.

State minimization:

1. Start with two sets, one of all the accepting, the other one of all non-accepting.
2. For every character a in alphabet or on newly-split partitions:
   - If the states in a partition all have a-transition to another partition, this defines a a-transition between the two partitions.
   - If only a group of the states in a partition has an a-transition to another partition, the partition should be split.

LL(1) Parsing

Parser action table:

Parsing stack	Input	Action
$ symbols (*reversed*)	input $	match or generate
$	$	accept

Left recursion removal and left factoring: ⋯

First set and follow set: ⋯

ε is never in a follow set.

Constructing parsing table:

1. If $A \to \alpha$ and $\alpha \Rightarrow^* a\beta$ where a is a token, then add $A \to \alpha$ to (A, a).
2. If $A \to \alpha$ and $\alpha \Rightarrow^* \varepsilon$, $S\$ \Rightarrow^* \beta A a \gamma$ where a is a token or $\$$, then add $A \to \alpha$ to (A, a).

Or:

1. For each rule $A \to \alpha$, for each token a in $First(\alpha)$, add $A \to \alpha$ to (A, a).
2. For each rule $A \to \alpha$, if $\varepsilon \in First(\alpha)$, for each token a (including $\$$) in $Follow(\alpha)$, add $A \to \alpha$ to (A, a).

By lookahead of 1.

Parsing table:

	terminals

non-terminals

LR Parsing

Parser action table:

Parsing stack	Input	Action
$ symbols	input $	shift or reduce
$S	$	accept

LR(0) Parsing

NFA of items:

- Shift: $(A \to \alpha . X\gamma) \xrightarrow{X} (A \to \alpha X . \gamma)$, where X is a terminal or a non-terminal.
- Beginning of reduce: $(A \to \alpha . X\gamma) \xrightarrow{\varepsilon} (X \to . \beta)$, where X is a non-terminal.
- $S' \to . S$ for initial state.

DFA of items:

NFA to DFA, or directly written.

Parser action table:

Parsing stack	Input	Action
$ (symbol, state number)s	input $	shift or reduce
$S	$	accept

Shift-reduce conflict, reduce-reduce conflict: ⋯

Parsing table:

State	Input	Goto
	terminals (including $)	non-terminals
state number	s(state number), r(rule), accept or empty	state number or empty

SLR(1) Parsing

LR(0) with lookahead of 1 in parsing algorithm.

LR(1) Parsing

NFA of items:

- Shift: $([A \to \alpha . X\gamma, a]) \xrightarrow{X} ([A \to \alpha X . \gamma, a])$, where X is a terminal or a non-terminal.
- Beginning of reduce: $([A \to \alpha . B\gamma, a]) \xrightarrow{\varepsilon} ([B \to . \beta, b])$, where B is a non-terminal, for every $b \in First(\gamma a)$
- $S' \to . S$ for initial state.

$First(\gamma a)$ can be a proper subset of $Follow(B)$, which is the power of LR(1) over SLR(1).

LALR(1) Parsing

NFA of items:

- Use $[rule, token/token]$.
- Construction:
  - Merge states of the same core from LR(1).

- Or by propagating lookaheads from LR(0).

Attribute Grammar

Attribute grammar:

Grammar Rule	Semantic Rules
rule	attribute equations

Dependency graph: Arrow means contributes to, with parse tree edges dashed. Dependency on parent or sibling can be drawn as dashed line if it can be done within siblings which can be programmatically implemented.

Stack-based Runtime Environment

arguments
access/static link
control/dynamic link
fp
return address
local variables
sp