

Theory of Computation and Compilers

Curated computer science study track (not a degree, no licensure claim). Original explanations and practice only, following the reviewed course handout.

Decide, then translate

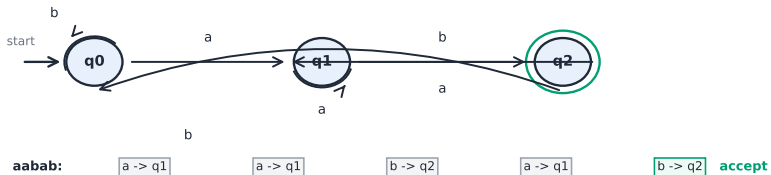
- 11.1-11.3 the theory: automata, **context-free languages**, computability.
- 11.4-11.6 the practice: lexing, parsing, semantics and code generation.

Automata: the table decides

- 1 **Trace** ends-in-ab DFA on aabab: $q_0 \ q_1 \ q_1 \ q_2 \ q_1 \ q_2$ - accept; aba ends q_1 - reject; the suite asserts every step.
- 2 **Pump** Any long accepted string splits xyz with pumpable y - $a^n b^n$ is not regular: pumping the a-block breaks the count.
- 3 **Guess** NFAs branch; subset construction tracks the branches at once - determinisation can square the state count.

State diagrams and acceptance trace

a DFA that accepts strings ending in ab



pumping ceiling: no finite machine matches $a^n b^n$ — pump the loop and the count breaks

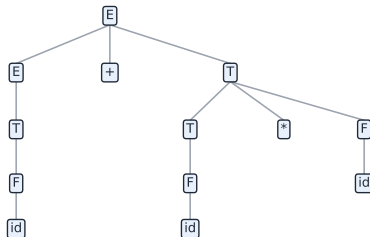
Three states; the trace of aabab accepts at q2.

Context-free: grammar nests

- 1 **Tree** $E \rightarrow E+T$, $T \rightarrow T*F$, $F \rightarrow \text{id}$: $\text{id} + \text{id} * \text{id}$ parses once - $*$ binds under T , deeper than $+$.
- 2 **Ambiguity** $E \rightarrow E+E \mid E*E \mid \text{id}$ parses the same string twice - the suite counts 2 trees unlayered, 1 layered.
- 3 **Ceiling** $a^n b^n c^n$ needs two counts: CFGs pump one - context-sensitive territory.

Parse trees

layered: one tree



x binds under T, deeper than + — precedence by layering

unlayered: two trees



same string,
two trees

$E \rightarrow E+E \mid E * E \mid id$ gives both parses

suite counts: 2 trees unlayered, 1 layered

derivation: $E \Rightarrow E+T \Rightarrow T+T \Rightarrow id+T \Rightarrow id+T * F$

$\Rightarrow id+id * F \Rightarrow id+id * id$ (six steps)

$a^n b^n c^n$ needs two counts: beyond context-free

Layered: one tree. Unlayered: two trees for the same string.

Computability: diagonalisation

- 1 **D** Assume halting decider H ; $D(M)$ runs $H(M,M)$ and loops exactly when H says halts - D on D contradicts.
- 2 **Rice** Every nontrivial semantic property is undecidable - does P print banana reduces halting to printing.
- 3 **Cousin** DFA emptiness IS decidable: finite reachability search - the infinite tape is exactly what breaks the argument.

Machine model

D flips H answers about itself — no H exists



table view: enumerate machines x inputs; D disagrees with the diagonal of every row

M3	1	1	0
M2	0	1	0
M1	1	0	1

diagonal

decidable cousins:

DFA emptiness: reachability search — finite, so decidable

Rice: every nontrivial semantic property undecidable

D flips H answers about itself - no H can exist.

Lexical analysis: longest match wins

- 1 **Line** `count = a1 + b2;` lexes to IDENT ASSIGN IDENT PLUS IDENT SEMI at columns 1 7 9 12 14 16.
- 2 **Match** `ifx` is one IDENT - greedy longest match beats the keyword prefix; if alone is a keyword.
- 3 **Policy** `12abc` is NUM then IDENT: some languages allow it, ours reports it - positions carried for every diagnostic.

Token pipeline

characters in, tokens out — longest match wins

```
count = a1 + b2;
```

1 4 7 10 13 16



DFA per token class (regex-compiled)



policy notes: ifx is one IDENT (longest match beats the keyword prefix); whitespace skipped; positions kept

error policy is part of the specification: 12abc is NUM then IDENT — our lexer reports it

Characters in, **tokens** out - longest match with priority.

Parsing: table plus stack

- 1 **Trace** $E \rightarrow T E_p$, $E_p \rightarrow + T E_p \mid \text{empty}$, $T \rightarrow \text{id}$: $\text{id} + \text{id}$ parses in eight stack snapshots, suite-asserted.
- 2 **Conflicts** A cell with two productions means one lookahead cannot choose - the grammar is not LL(1).
- 3 **Tail** $E \rightarrow E+T$ loops the stack; $E \rightarrow T E_p$ makes the loop a tail - associativity moves to the AST build.

Parse table and stack trace

the LL(1) table

nonterm	lookahead	production
E	id	T Ep
Ep	+	+ T Ep
Ep	\$	empty

FIRST/FOLLOW fill the rows;

conflicts reject the grammar

left recursion $E \rightarrow E+T$ loops the
stack; the tail trick fixes it:

$E \rightarrow T \text{ Ep}, \text{ Ep} \rightarrow + T \text{ Ep} \mid \text{empty}$

the stack trace of id + id

1. E
2. T Ep
3. id Ep
4. Ep
5. + T Ep
6. T Ep
7. id Ep
8. Ep
9. (empty)

eight moves, one tree, one symbol of lookahead

The LL(1) table and the eight-snapshot stack trace of id + id.

Semantic analysis: decorate, then emit

- 1 **Program** $\text{count} = a1 + b2 * 3$ with all int: $t1 = b2 * 3$; $t2 = a1 + t1$; $\text{count} = t2$ - three instructions, one temp.
- 2 **Error** b2: str rejects $b2 * 3$ with the lexer position - positions are the pipeline error currency.
- 3 **Fold** Literal-only: $2 * 3$ becomes 6; $b2 * 3$ stays - folding is a proof about literals.

Symbol table and IR

decorate with meaning, then emit IR

```
count = a1 + b2 * 3
```



symbol table

a1	int	global
b2	int	global
count	int	global

IR (three-address)

```
t1 = b2 * 3
t2 = a1 + t1
count = t2
```

type walker: $b2 * 3$ (int x literal) then $a1 + \dots$ (int + int) — all int, accepted

error case: b2: str rejects $b2 * 3$ with the lexer position

why IR: m languages x n targets = $m + n$, not $m \times n$; folding is literal-only ($2 * 3 \rightarrow 6$)

The **symbol table** binds a1, b2, count as int; the IR is three instructions.

Check yourself

- The course DFA on aba ends at...? (q1 — rejected.)
- $\text{id} + \text{id} * \text{id}$: leftmost **derivation** steps...? (Six.)
- ifx lexes as...? (One IDENT.)
- $\text{count} = \text{a1} + \text{b2} * 3$ emits...? (t1, t2, count = t2.)