CS4031 — Compiler Construction

Assignment 01 — Automata Design

Members: 22i-1271 Ahmed Ali Zahid - 22i-1120 Asad Mehdi

Token Types Covered:

1. Integer Literal [+-]?[0-9]+
2. Floating-Point Literal [+-]?[0-9]+\.[0-9]{1,6}([eE][+-]?[0-9]+)?
3. Identifier [A-Z][a-z0-9_]{0,30}
4. Single-line Comment ##[^\n]*
5. Keyword start|finish|loop|condition|...
6. String Literal "([^\"\\\n]|\\["\\ ntr])*"
7. Boolean Literal true|false

Legend: → Start State | Double ring = Accepting State | — = Dead/Trap State

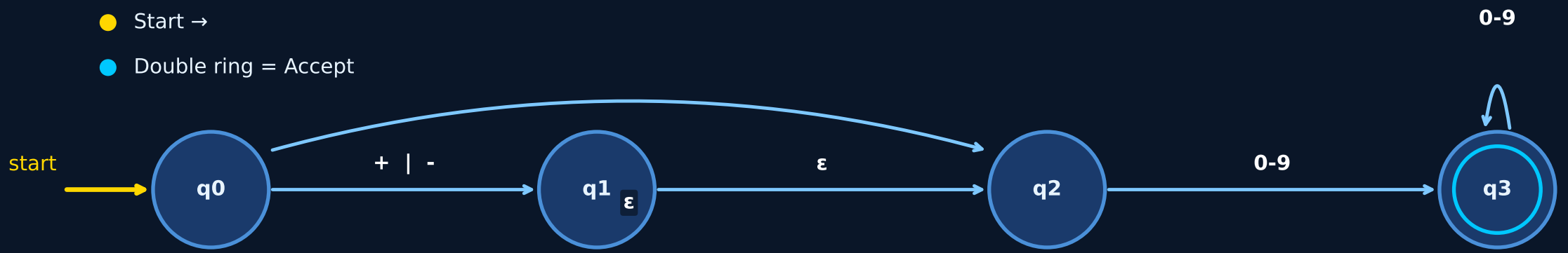
Regular Expression Specifications — All Token Types

Keywords	`(start|finish|loop|condition|declare|output|input|function|return|break|continue|else)`
Identifier	`[A-Z][a-z0-9_]{0,30}`
Integer Literal	`[+-]?[0-9]+`
Float Literal	`[+-]?[0-9]+\.[0-9]{1,6}([eE][+-]?[0-9]+)?`
String Literal	`"([^"\\\n]|\\["\\ntr])*"`
Character Literal	`'([^'\\\n]|\\['\\ntr])'`
Boolean Literal	`(true|false)`
Arithmetic Ops	`(\*\*|[+\-*/%])`
Relational Ops	`(==|!=|<=|>=|<|>)`
Logical Ops	`(&&||||!)`
Assignment Ops	`(\+=|-=|\*=|/=|=)`
Inc/Dec Ops	`(\+\+|--)`
Punctuators	`[(){}[\],;:]`
Single Comment	`##[^\n]*`
Multi-line Comment	`#\*([^*]|\*+[^*#])*\*+#`
Whitespace	`[ \t\r\n]+` (skip, track line/col)

PART A — NFA Diagrams

7 Non-deterministic Finite Automata

NFA — Integer Literal Regex: [+-]?[0-9]+



States: q0=start, q1=sign seen, q2=pre-digit, q3=digit(s) seen (accept)

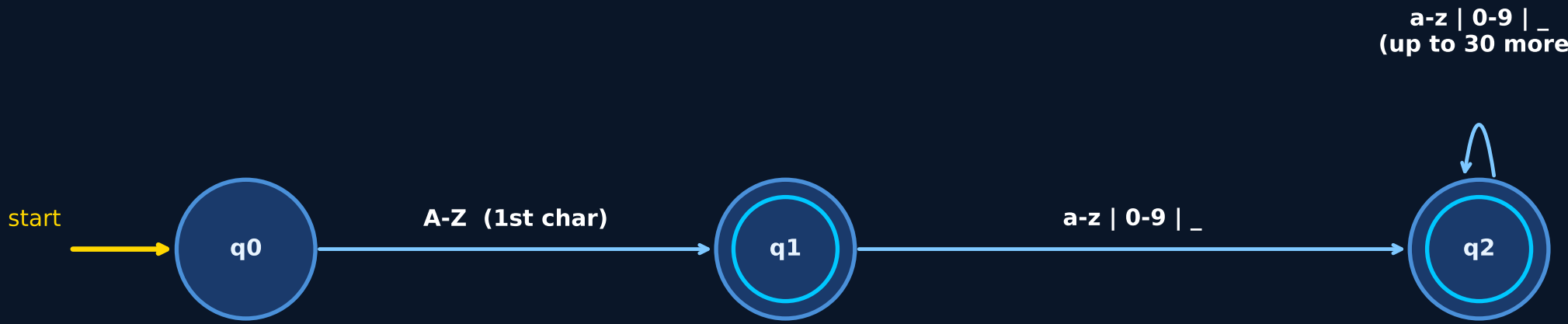
ε-transitions allow optional sign; q3 loops on any digit (0-9)

NFA — Floating-Point Literal Regex: [+-]?[0-9]+\.[0-9]{1,6}([eE][+-]?[0-9]+)?

q5: accept after 1-6 decimal digits | q9: accept after exponent digits

Exponent part (q6→q9) is optional via ε from q5 → exit automaton

NFA — Identifier Regex: [A-Z][a-z0-9_]{0,30} (max 31 chars)



q1 accepts single uppercase letter identifier | q2 accepts after additional chars

Counter enforced in implementation (≤30 subsequent chars → max 31 total)

NFA — Single-Line Comment Regex: ##[^\n]*

Two # characters start the comment; everything until newline is consumed

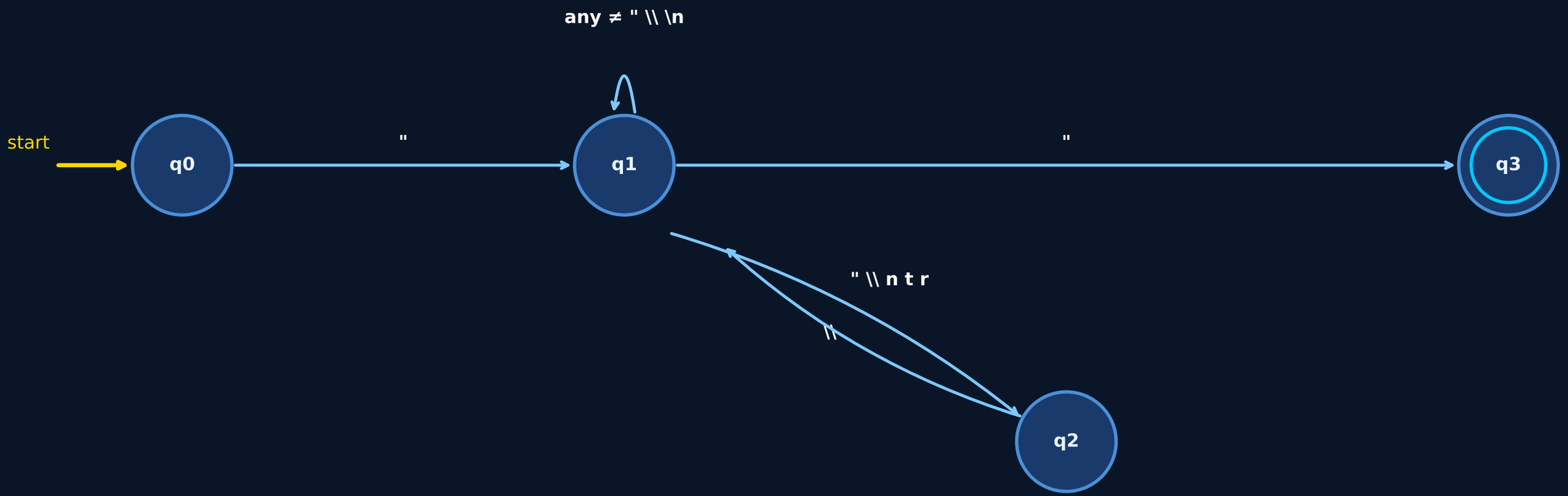
q3 loops on any non-newline character; accepted when newline (or EOF) encountered

NFA — Keywords (representative: 'start') Regex: start|finish|loop|...

Other keywords branch similarly from q0
(finish, loop, condition, declare, output,
input, function, return, break, continue, else)

Each keyword gets a dedicated path; matched before identifiers (priority rule)

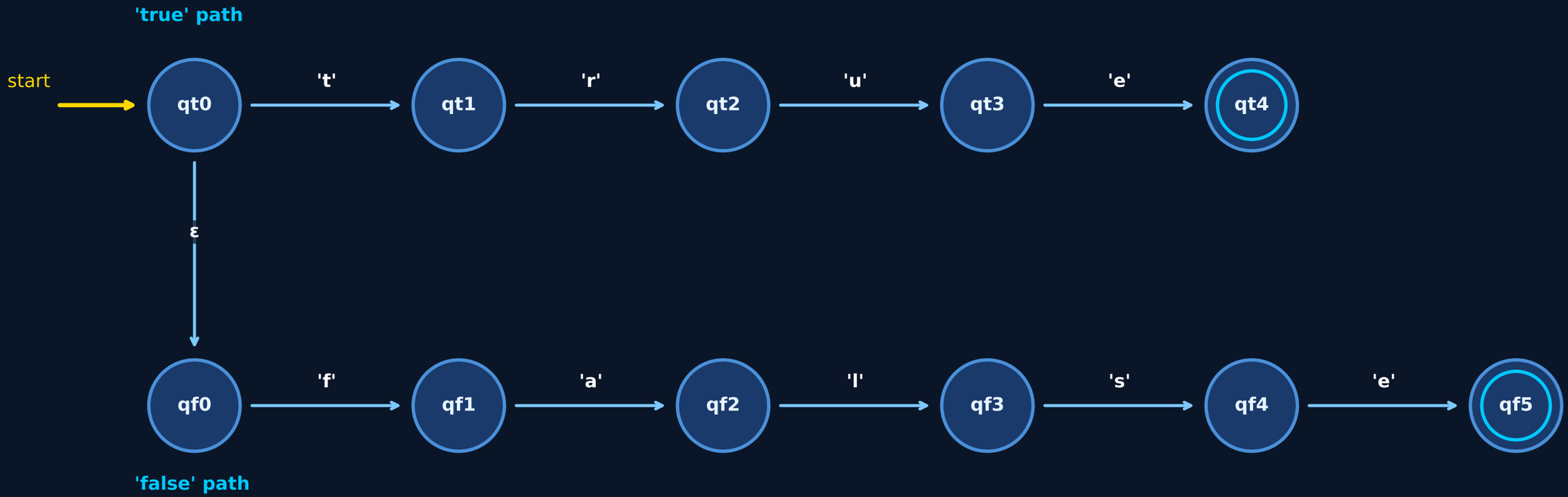
NFA — String Literal Regex: "([^\"\\\n]|\\["\\ ntr])*"



q1: inside string; loops on normal chars; q2 handles escape sequence

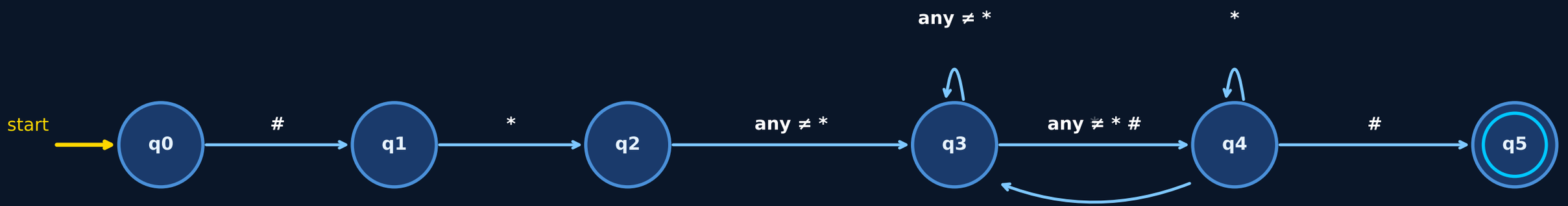
If escape char seen (\\), go to q2; return to q1 on valid escape char

NFA — Boolean Literal Regex: true|false



NFA uses ε-transitions from shared start; each path independently matches 'true' or 'false'

NFA — Multi-line Comment Regex: #*([^*]|*+[^*#])**+#

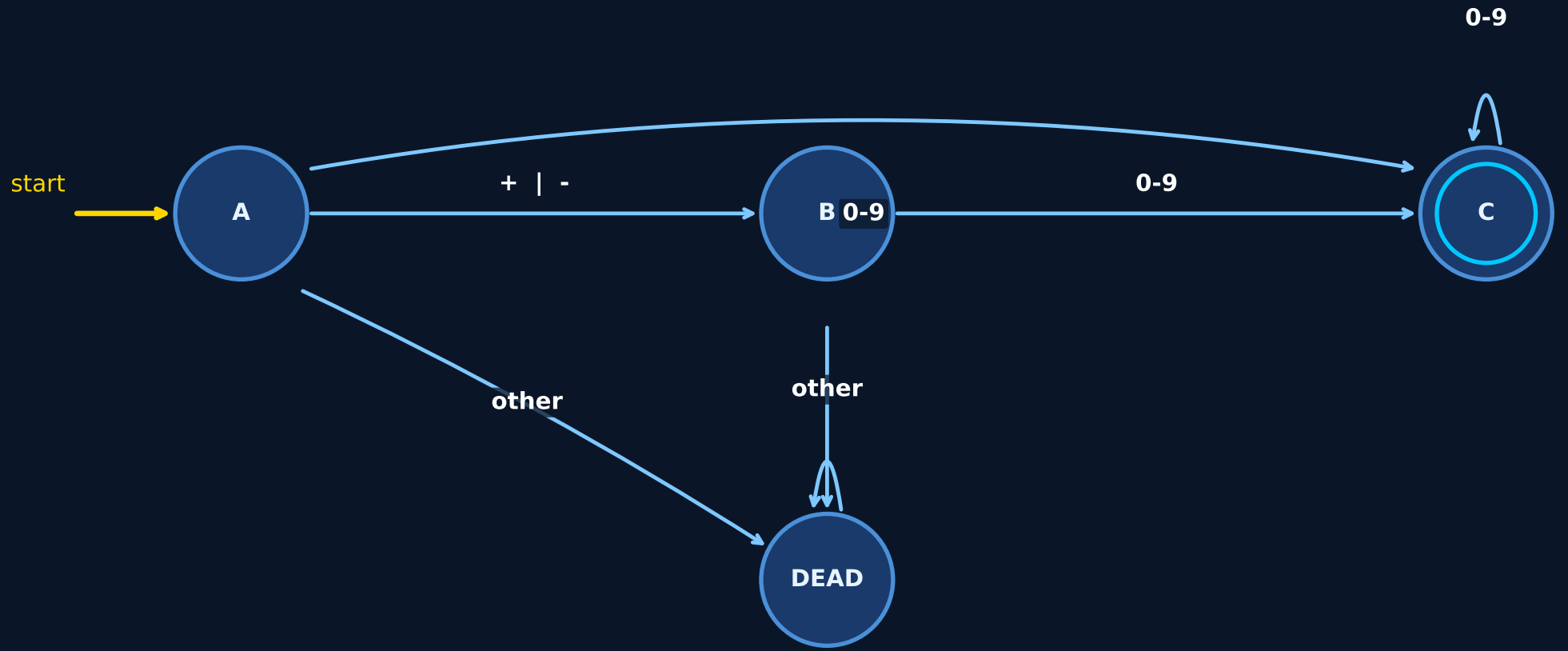


q3 loops on non-* chars; q4 accumulates * chars
if next char is not #, return to q3

PART B — Minimized DFA Diagrams

7 Deterministic Finite Automata (Minimized)

Minimized DFA — Integer Literal Regex: [+-]?[0-9]+

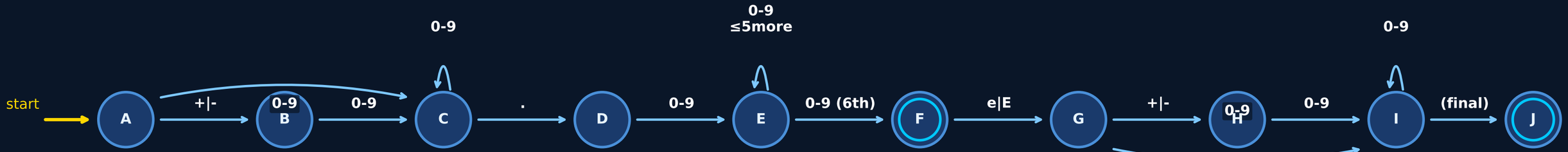


States: A=start, B=sign, C=digits(accept), DEAD=error trap

all

Minimized: sign states merged; dead state handles all invalid inputs

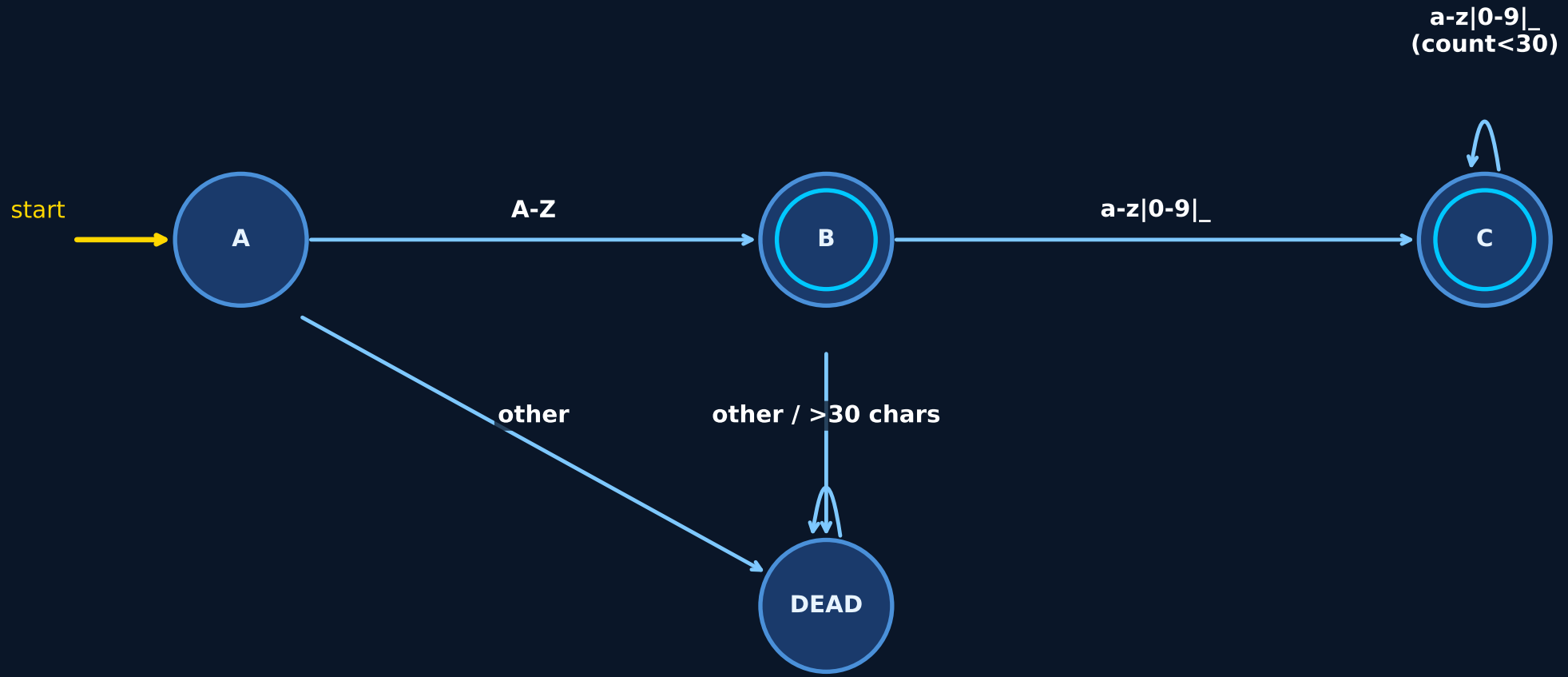
Minimized DFA — Floating-Point Literal Regex: [+-]?[0-9]+\.[0-9]{1,6}([eE][+-]?[0-9]+)?



A=start B=sign C=int-part D=dot E=dec(1-5) F=dec-6(accept)

G=e/E H=exp-sign I=exp-digits J=exp-accept | F also accepts (no exponent)

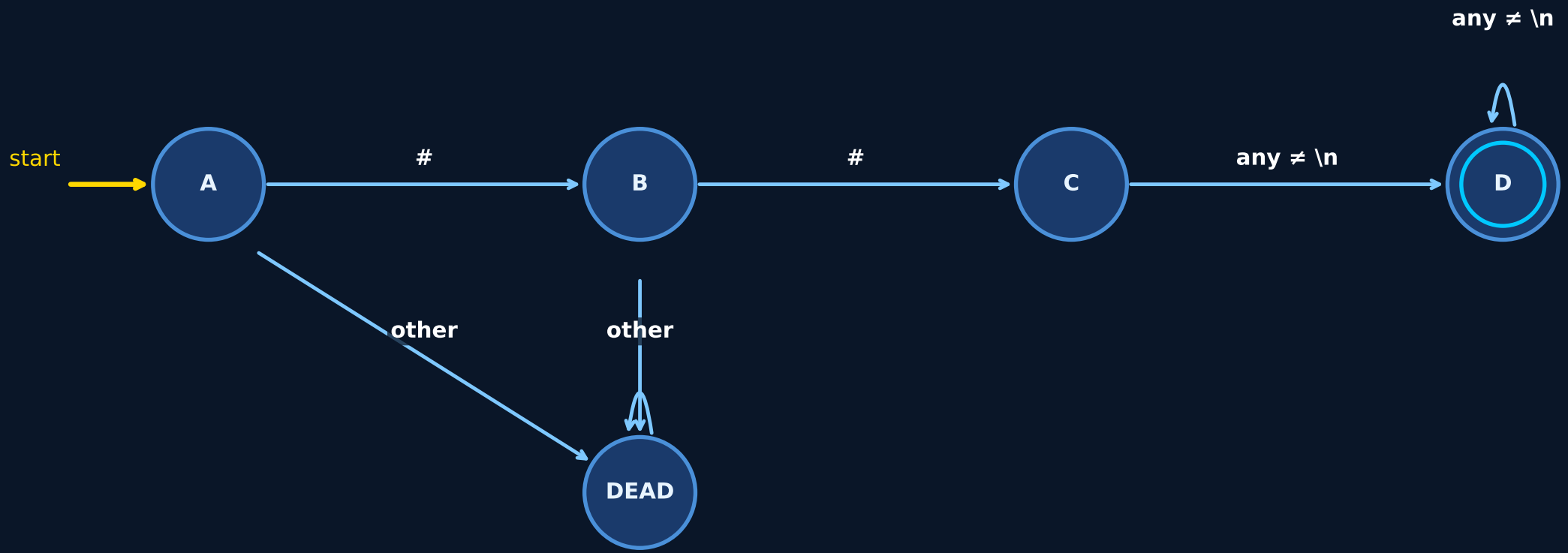
Minimized DFA — Identifier Regex: [A-Z][a-z0-9_]{0,30}



B and C both accept (single uppercase or with suffix); DEAD = error trap

Length counter resets to 0 in implementation; max suffix length = 30

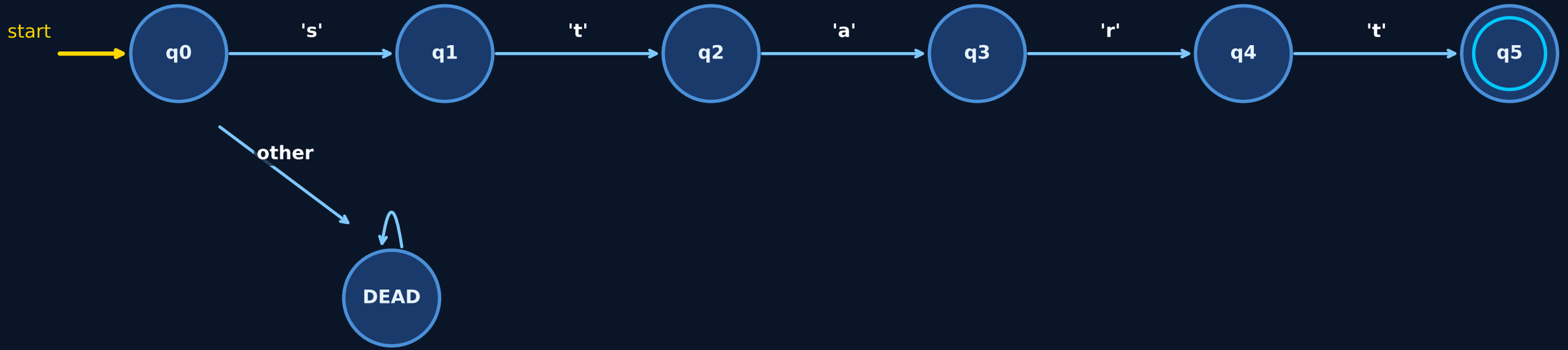
Minimized DFA — Single-Line Comment Regex: ##[^\n]*



A=start B=first# C=second# D=comment-body(accept) DEAD=trap

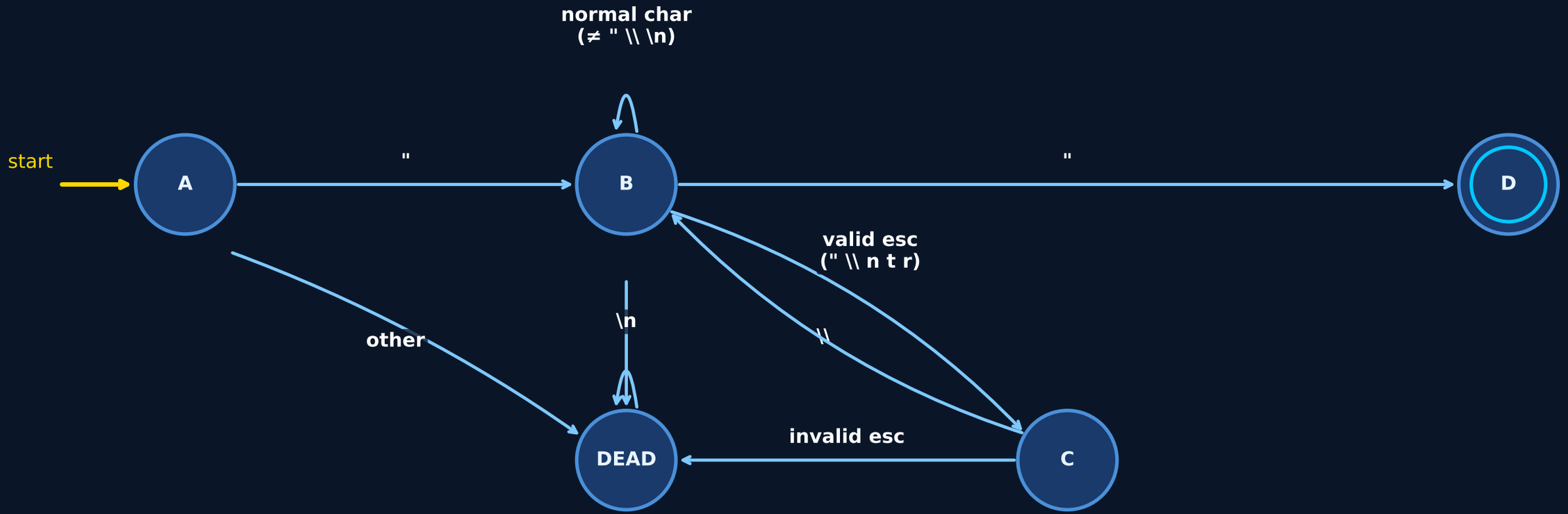
all

Minimized DFA — Keyword (e.g. 'start') All keywords follow identical pattern

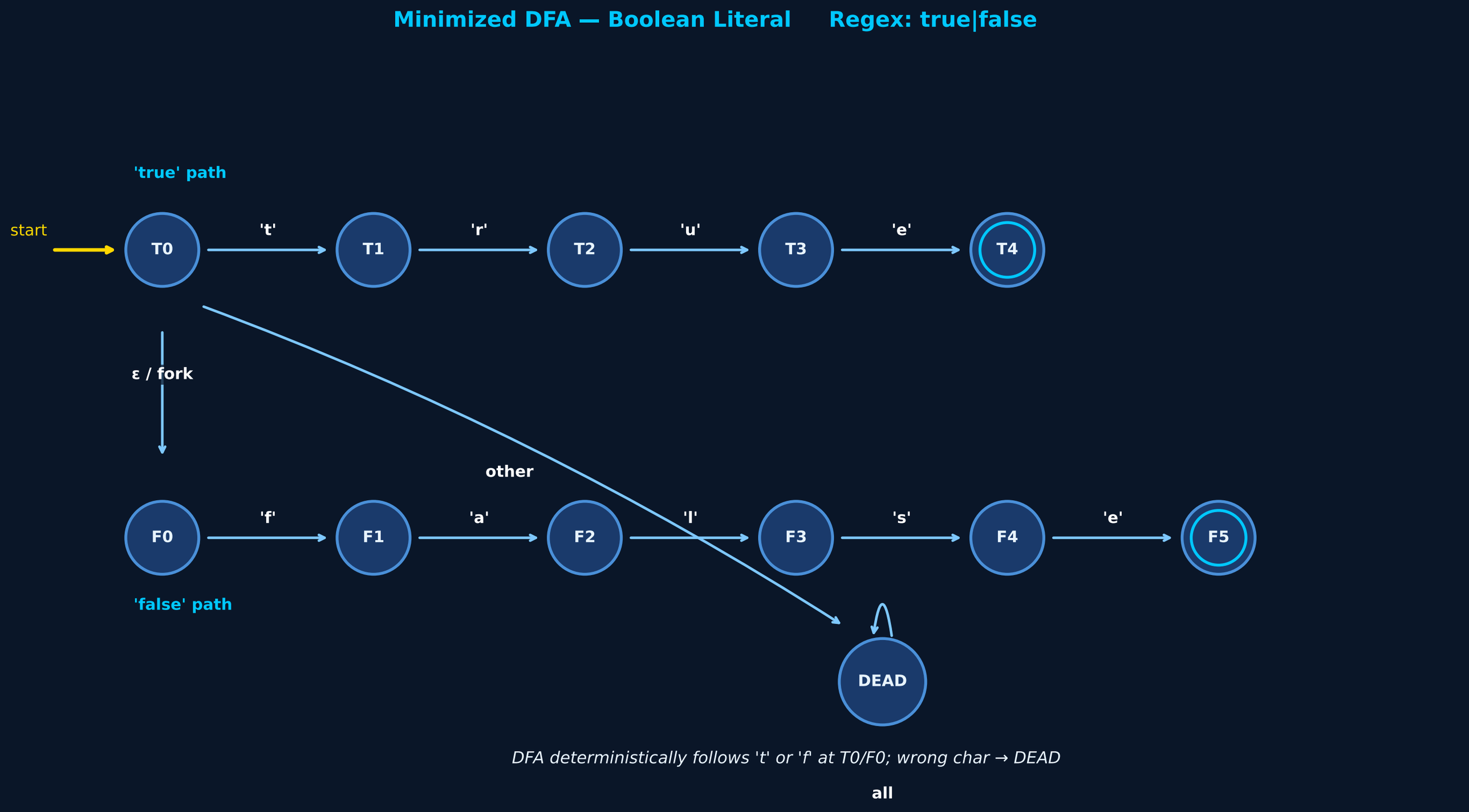


Each wrong character at any state → DEAD trap | Only exact keyword string accepted

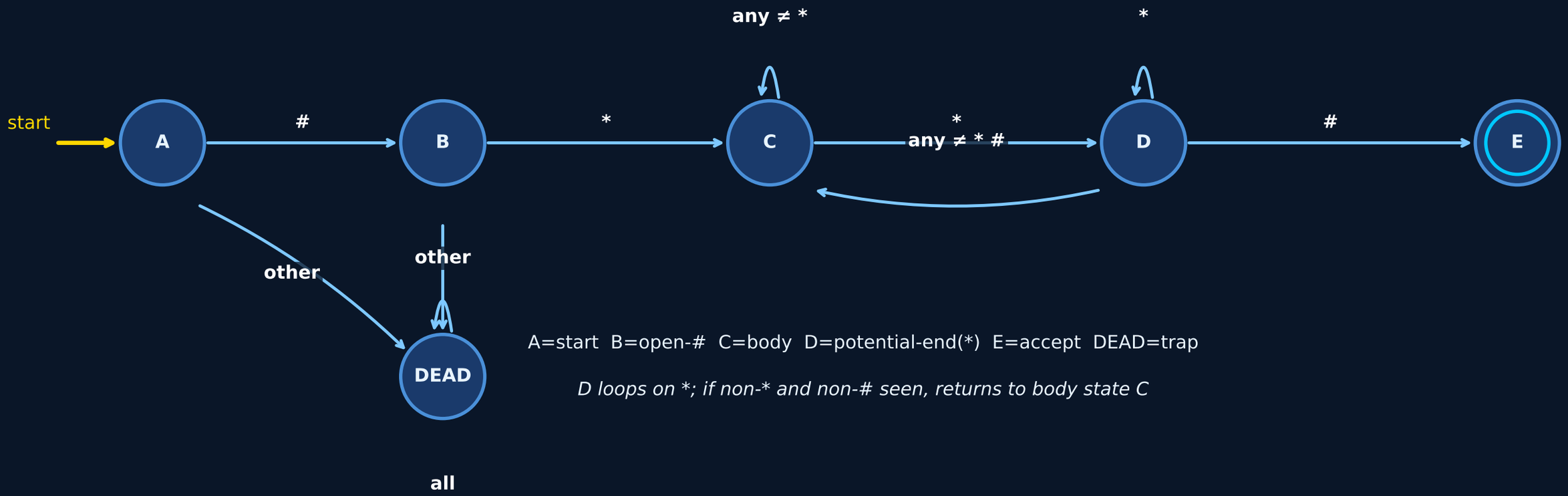
Minimized DFA — String Literal Regex: "([^\"\\\n]|\\["\\ ntr])*"



A=start B=inside-string C=escape-sequence D=accept DEAD=trap


Minimized DFA — Boolean Literal
Regex: true|false
'true' path
start
T0
't'
T1
'r'
T2
'u'
T3
'e'
T4
ε / fork
other
F0
'f'
F1
'a'
F2
'l'
F3
's'
F4
'e'
F5
'false' path
DEAD
DFA deterministically follows 't' or 'f' at T0/F0; wrong char → DEAD
all

Minimized DFA — Multi-line Comment Regex: #*([^*]|*+[^*#])**+#



A=start B=open-# C=body D=potential-end(*) E=accept DEAD=trap

*D loops on *; if non-* and non-# seen, returns to body state C*

PART C — Transition State Tables

7 DFA Transition Tables with All Input Classes

Transition Table — Integer Literal DFA

Regex: [+-]?[0-9]+ States: A=start, B=sign, C=digits, DEAD=trap

State	+ | -	0-9	other
→ A (start)	B	C	DEAD
B (sign)	—	C	DEAD
*C (accept)	—	C	DEAD
DEAD	—	—	DEAD

→ = start state * = accepting state — = dead/trap state

Transition Table — Floating-Point Literal DFA
E transitions to F after 6th decimal digit (counter in implementation)

State	+|-	0-9	.	e|E	valid esc	other
→ A	B	C	—	—	—	DEAD
B	—	C	—	—	—	DEAD
C	—	C	D	—	—	DEAD
D	—	E	—	—	—	DEAD
E	—	E/F	—	G	—	DEAD
*F	—	—	—	G	—	DEAD
G	H	I	—	—	—	DEAD
H	—	I	—	—	—	DEAD
*I	—	I	—	—	—	DEAD
DEAD	—	—	—	—	—	DEAD

→ = start state * = accepting state — = dead/trap state

Transition Table — Identifier DFA
Both B and C are accepting; C loops up to 30 additional characters

State	A-Z	a-z|0-9|_	other / >30
→ A	*B	—	DEAD
*B (accept)	—	*C	DEAD
*C (accept)	—	*C	DEAD
DEAD	—	—	DEAD

→ = start state * = accepting state — = dead/trap state

Transition Table — Single-Line Comment DFA
Newline terminates comment and triggers accept; not consumed by scanner

State	#	any ≠ \n	\n / other
→ A	B	—	DEAD
B	C	—	DEAD
C	*D	*D	exit
*D (accept)	—	*D	exit
DEAD	—	—	DEAD

→ = start state * = accepting state — = dead/trap state

Transition Table — Keyword 'start' DFA (representative)

All 12 keywords follow the same chain pattern; each has its own DFA path

State	's'	't'	'a'	'r'	't'(2nd)	other
→ q0	q1	—	—	—	—	DEAD
q1	—	q2	—	—	—	DEAD
q2	—	—	q3	—	—	DEAD
q3	—	—	—	q4	—	DEAD
q4	—	—	—	—	*q5	DEAD
*q5	—	—	—	—	—	DEAD
DEAD	—	—	—	—	—	DEAD

→ = start state * = accepting state — = dead/trap state

Transition Table — String Literal DFA
normal = any ≠ " \\ \n | valid esc = " \\ n t r

State	"	\\	normal	valid esc	invalid esc/\n
→ A	B	DEAD	DEAD	DEAD	DEAD
B (inside)	*D	C	B	B	DEAD
C (escape)	DEAD	DEAD	DEAD	B	DEAD
*D (accept)	—	—	—	—	—
DEAD	—	—	—	—	DEAD

→ = start state * = accepting state — = dead/trap state

Transition Table — Boolean Literal DFA

T=true path, F=false path; shared start state disambiguates on first char 't' vs 'f'

State	't'	'r'	'u'	'e'	'f'	'a'	'l'	's'	'e'(2)	other
→ T0/F0	T1	—	—	—	F1	—	—	—	—	DEAD
T1	—	T2	—	—	—	—	—	—	—	DEAD
T2	—	—	T3	—	—	—	—	—	—	DEAD
T3	—	—	—	*T4	—	—	—	—	—	DEAD
*T4	—	—	—	—	—	—	—	—	—	DEAD
F1	—	—	—	—	—	F2	—	—	—	DEAD
F2	—	—	—	—	—	—	F3	—	—	DEAD
F3	—	—	—	—	—	—	—	F4	—	DEAD
F4	—	—	—	—	—	—	—	—	*F5	DEAD
*F5	—	—	—	—	—	—	—	—	—	DEAD
DEAD	—	—	—	—	—	—	—	—	—	DEAD

→ = start state * = accepting state — = dead/trap state

Transition Table — Multi-line Comment DFA
C loops on # (valid in body); D exits to E on # (after *); D returns to C on other char

State	#	*	other	invalid
→ A (start)	B	DEAD	DEAD	DEAD
B	DEAD	C	DEAD	DEAD
C (body)	C	D	C	DEAD
D (* seen)	*E	D	C	DEAD
*E (accept)	—	—	—	—
DEAD	—	—	—	DEAD

→ = start state * = accepting state — = dead/trap state