CS143: Lexical Analysis

David L. Dill

Stanford University

Introduction

- PA1 (or PA1J) is assigned today.
  - Due in one week
  - I recommend starting immediately.
- Review notes on regular languages based on CS103
  - Includes DFA minimization – optional
- Notes on the material of lectures 1, 3, 4.

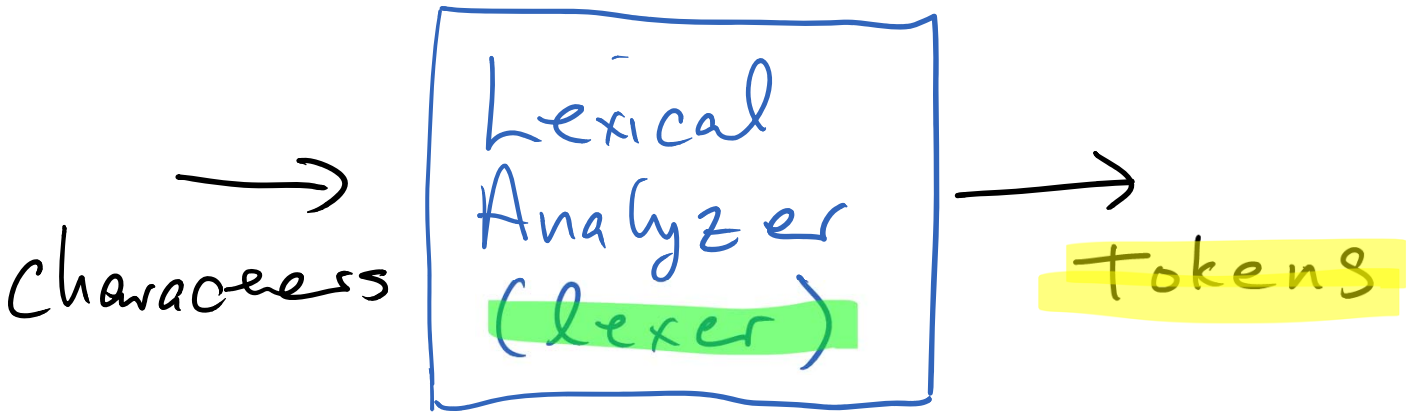
Lexical Analysis

- Interface to Rest of Compiler
- Formal Languages Concepts
- Regular Expressions
- Deterministic Finite Automata
- Theory vs. Practice

Interfaces to Rest of Compiler

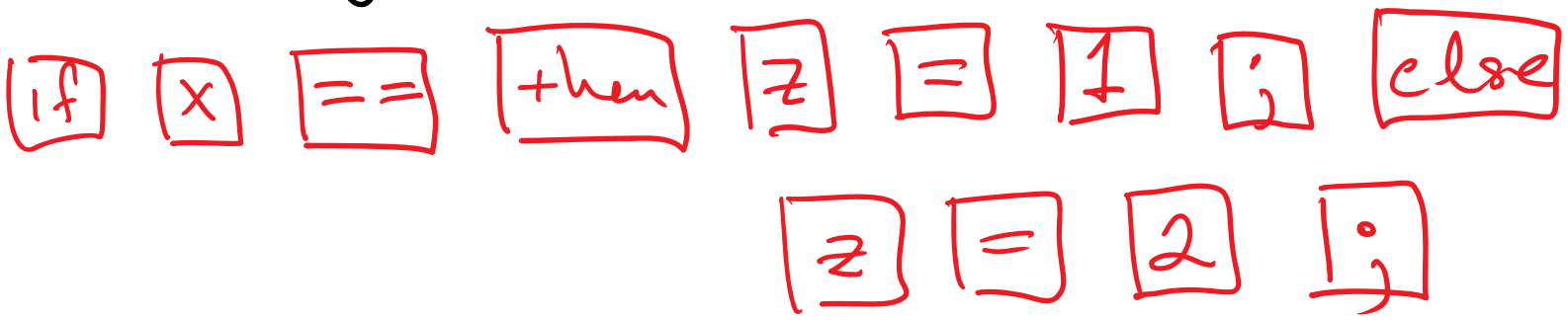
Lexical analysis

l e t t e r s → words

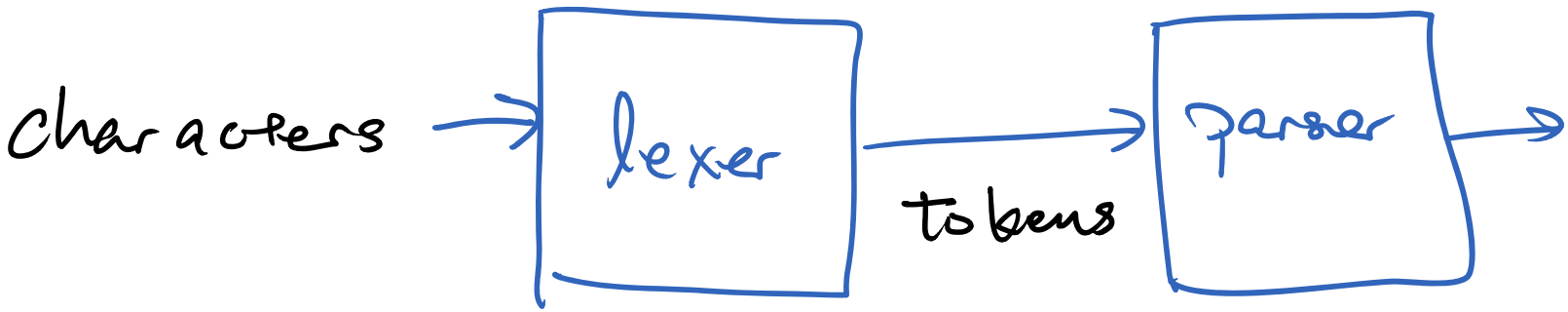


Lexical analysis

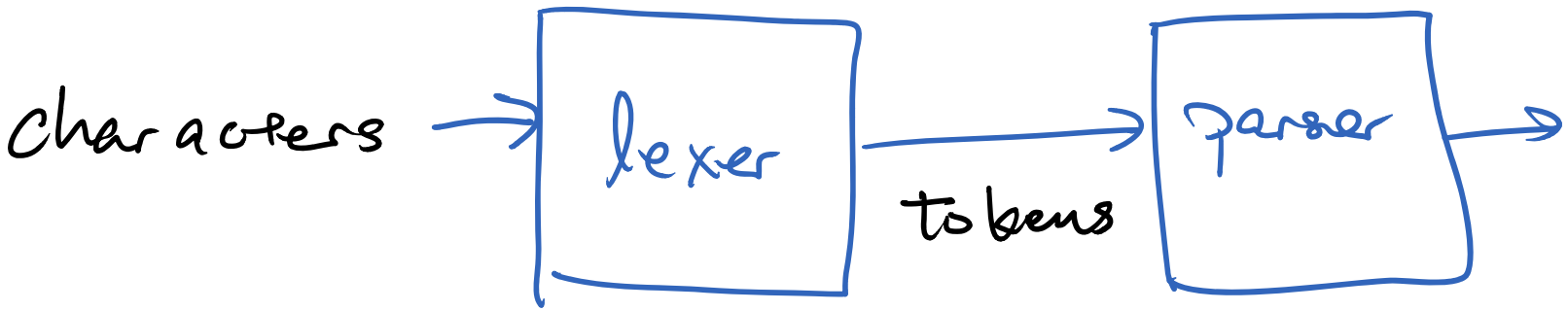
if x == y then z = 1 , else z = 2;



Interface

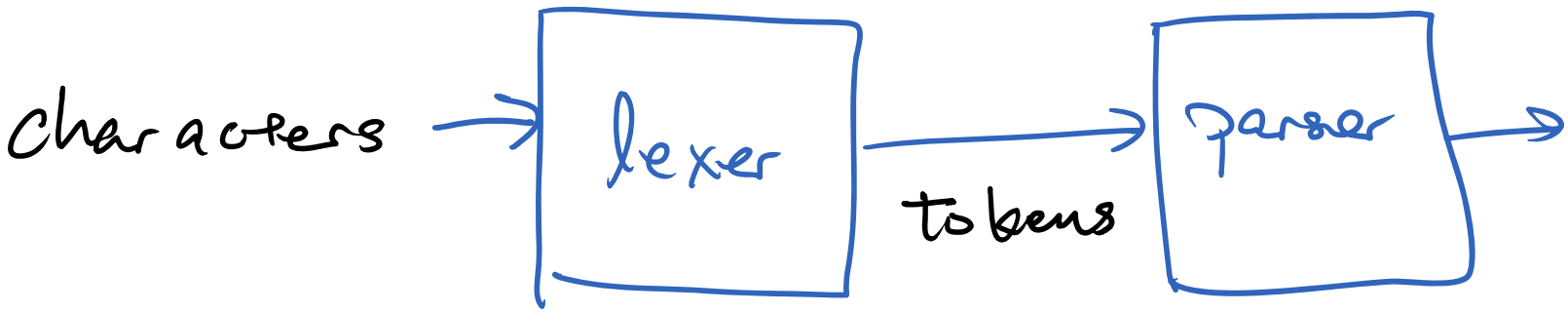


Interface



tokens are generated "on demand" for parser.

Interface

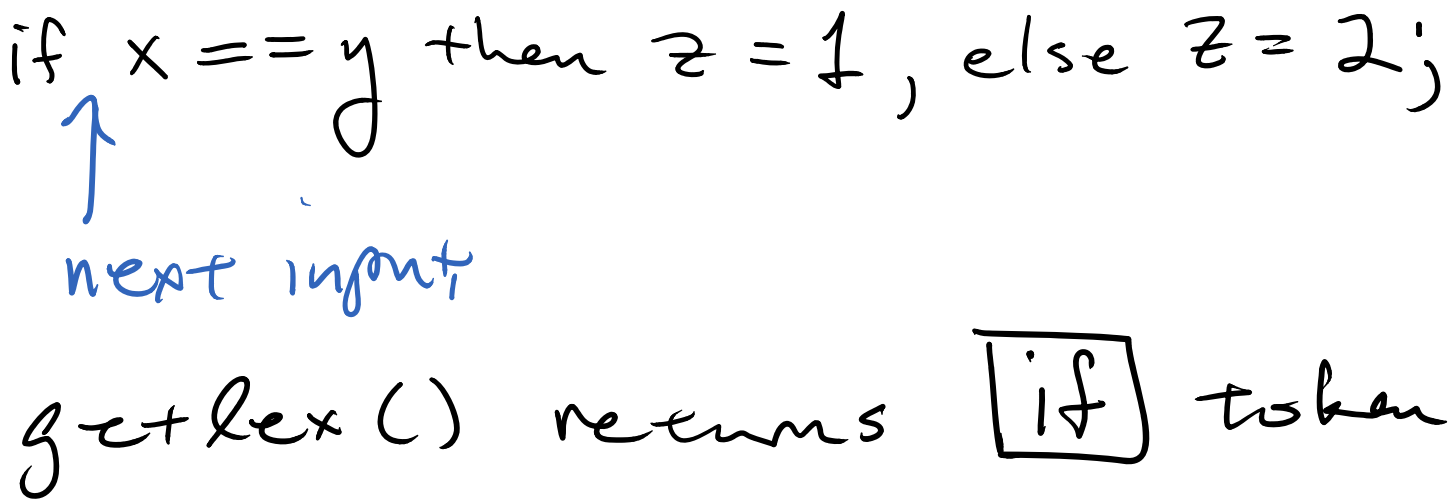


tokens are generated "on demand" for parser.

getlex() → returns next token

if x == y then z = 1, else z = 2;

next input


if x == y then z = 1 , else z = 2 ;
next input
getlex() returns
if
token

if x == y then z = 1 , else z = 2;

next input

getlex() returns [if] token

next getlex() returns [x]

if x == y then z = 1 , else z = 2 ;

↑ next input

getlex() returns [if] token

next getlex() returns [x]

(discards space after "if" and keeps going until it can return a token.)

Terminology

If $x == y$ then ...

Terminology

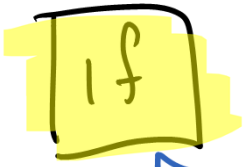
If x == y then ...

↑

lexeme – input string for next token

Terminology

If x == y then ...

getlex() → 

token
data structure returned
by getlex
(goes to parser)

Performance

Lexer used to be performance-critical

↑ e.g. before 1990?

# of characters vs. # of tokens

lexer had to deal with all of these

later stages just had tokens & trees.

Performance

Lexer used to be performance-critical

So we know how to do it FAST =

Q: Why is the North American Antelope so speedy?

Formal Languages Concepts

Lexical Analysis Theory

Precise descriptions – Regular expressions

[a-z A-Z] [a-z A-Z 0-9 _]*

Automatic generation

Regular expressions → NFA → DFA

→ lexical analyzer program

Formal Language Concepts

Alphabet - A finite set of **symbols** (a.k.a **characters**).

Σ - standard symbol for an alphabet.

Formal Language Concepts

Alphabet - A finite set of **symbols** (a.k.a **characters**).

Σ - standard symbol for an alphabet.

For compilers, a character set like ASCII, UTF8

Formal Language Concepts

String – finite sequence of characters

E.g. abc

Formal Language Concepts

$|x|$ – length of the string x

ε – empty string $|\varepsilon| = 0$

Formal Language Concepts

$x \cdot y$ - concatenation of strings
(also written xy)

$abc \cdot def = abcdef$

Formal Language Concepts

Formal language – set of strings over some alphabet Σ.

Formal Language Concepts

If X, Y are languages over Σ,

$X \cdot Y = \{ xy \mid x \in X \wedge y \in Y \}$ is their concatenation.

Formal Language Concepts

If X, Y are languages over Σ,

$X \cdot Y = \{ xy \mid x \in X \wedge y \in Y \}$ is their concatenation.

Ex: $\{a, ab\} \cdot \{b, bb\}$

$= \{ab, abb, abbb\}$

Formal Language Concepts

Let X be a language. Then X^i is X concatenated with itself i times

$X^0 = \{ \varepsilon \}$

$X^1 = X$

$X^2 = XX$

etc.

Formal Language Concepts

$X^* = X^0 \cup X^1 \cup X^2 \cup \ldots$

is the Kleene closure of X.

Regular Expressions

Regular Expressions

Expression	Language
ε	$L(\varepsilon) = \{\varepsilon\}$
$a \quad (a \in \Sigma)$	$L(a) = \{a\}$
$R_1 \mid R_2$	$L(R_1 \mid R_2) = L(R_1) \cup L(R_2)$
$R_1 \cdot R_2$	$L(R_1 \cdot R_2) = L(R_1) \cdot L(R_2)$
R_1^*	$L(R_1^*) = L(R_1)^*$

Regular Expressions

Also: Precedence

*, ·, |

↑

"stickiest"

like exponentiation

Regular Expressions

Also: Precedence

*, ·, |

next stickiest

like multiplication

Regular Expressions

Also: Precedence

*, ·, |

↑

least sticky

like addition

Regular Expressions

Also: Precedence

*, ·, |

Parentheses to enforce grouping

$((a|b)(c|d))^*$

Notation used in tools is somewhat different (and more complex).

Abbreviations

$R?$ — "optional R"

abbrev for $(R|\varepsilon)$

R^+ — positive closure (one or more)

abbrev for RR^* or R^*R.

Regular Definitions

Allow new named abbreviations for regular expressions

Example: Java floating point (partial)

Expt: ((e|E)(+|-)? digit+)

Suffix: (f|F|d|D)

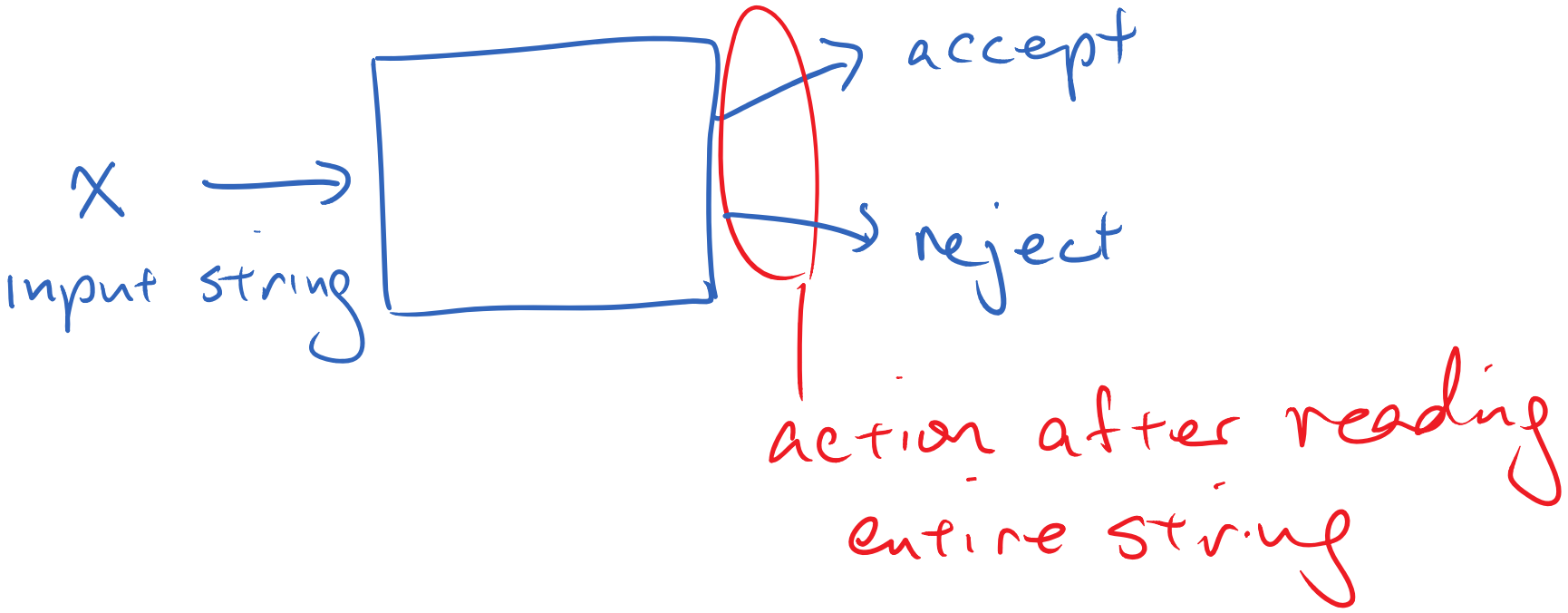
digit+ . digit* Expt? Suffix?
| . digit+ Expt? Suffix?
| digit+ Expt Suffix?
| digit+ Expt? Suffix

} Reg expr for Java FP nums

Deterministic Finite Automata

Deterministic Finite Automata

Good for implementing lexers.



DFA

Q – finite set of states

Σ – alphabet

$\delta : Q \times \Sigma \to Q$ next state function

partial function, unlike CS103

$q_0 \in Q$ start state

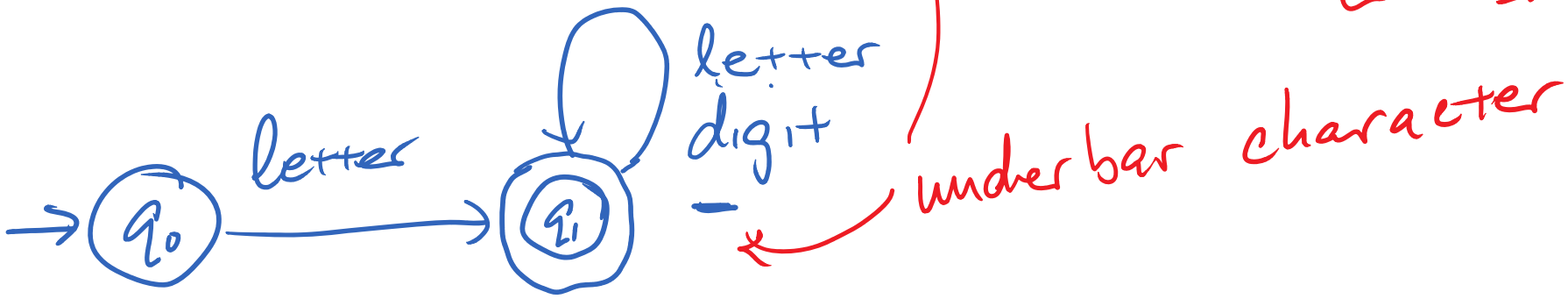
$F \subseteq Q$ Accepting states.

Example: Identifiers

letter (letter | digit | _)*

letter [a-zA-Z]

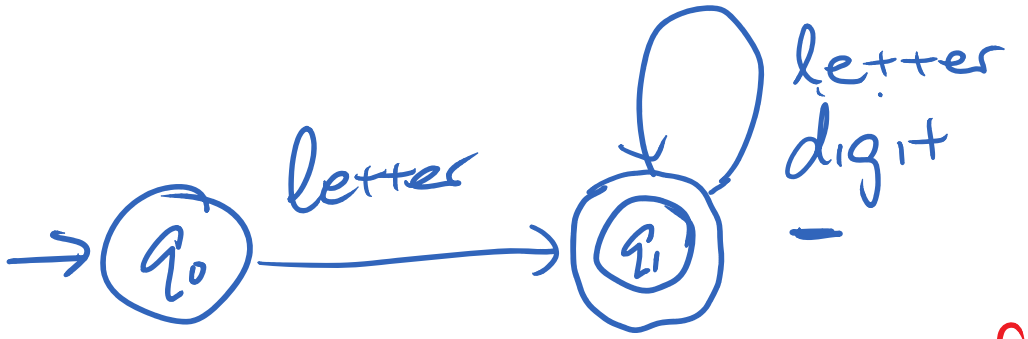
digit [0-9]



Example: Identifiers

letter (letter | digit | _)*

letter [a-zA-Z]
digit [0-9]



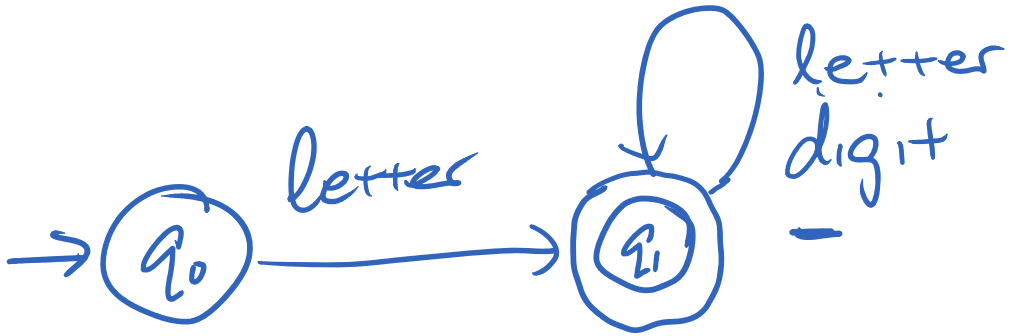
At most one arrow from each state on each symbol ("deterministic")

Ok to have no arrow on symbol (δ partial function)

Example: Identifiers

letter (letter | digit | _)*

letter [a-zA-Z]
digit [0-9]



run

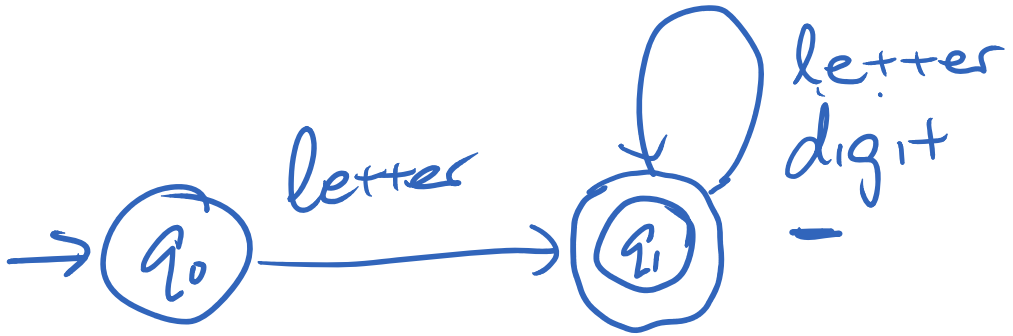
Input: a 1 b

$q_0 \xrightarrow{a} q_1 \xrightarrow{1} q_1 \xrightarrow{b} q_1$

Example: Identifiers

letter (letter | digit | _)*

letter [a-zA-Z]
digit [0-9]



run

Input: a 1 b

$q_0 \xrightarrow{a} q_1 \xrightarrow{1} q_1 \xrightarrow{b} q_1$

start state

last state is accepting, so a1b is ident

Automatic Conversion

Regular expression → NFA with ε transitions

→ DFA

→ minimized DFA

↑ not in CS103

Review notes posted

Example: Modula-2 Real Numbers

digit+ . digit* (E (+|-)? digit+)?

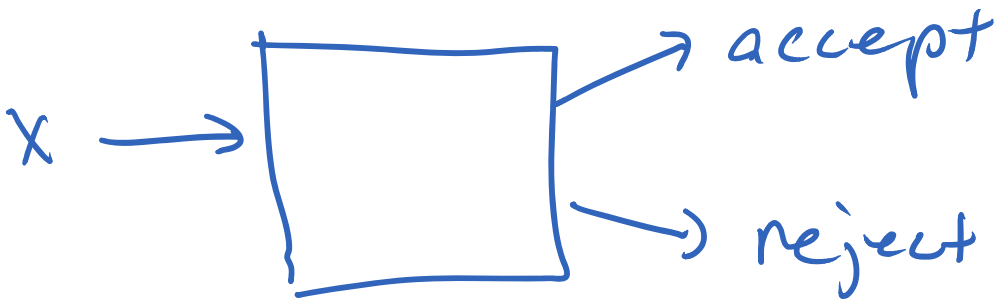
DFA?

Example: Modula-2 Real Numbers

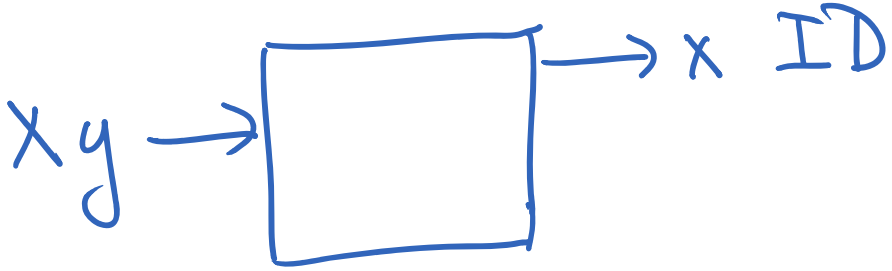
digit+ . digit* (E (+|-)? digit+)?

Theory vs. Practice

Theory



Practice



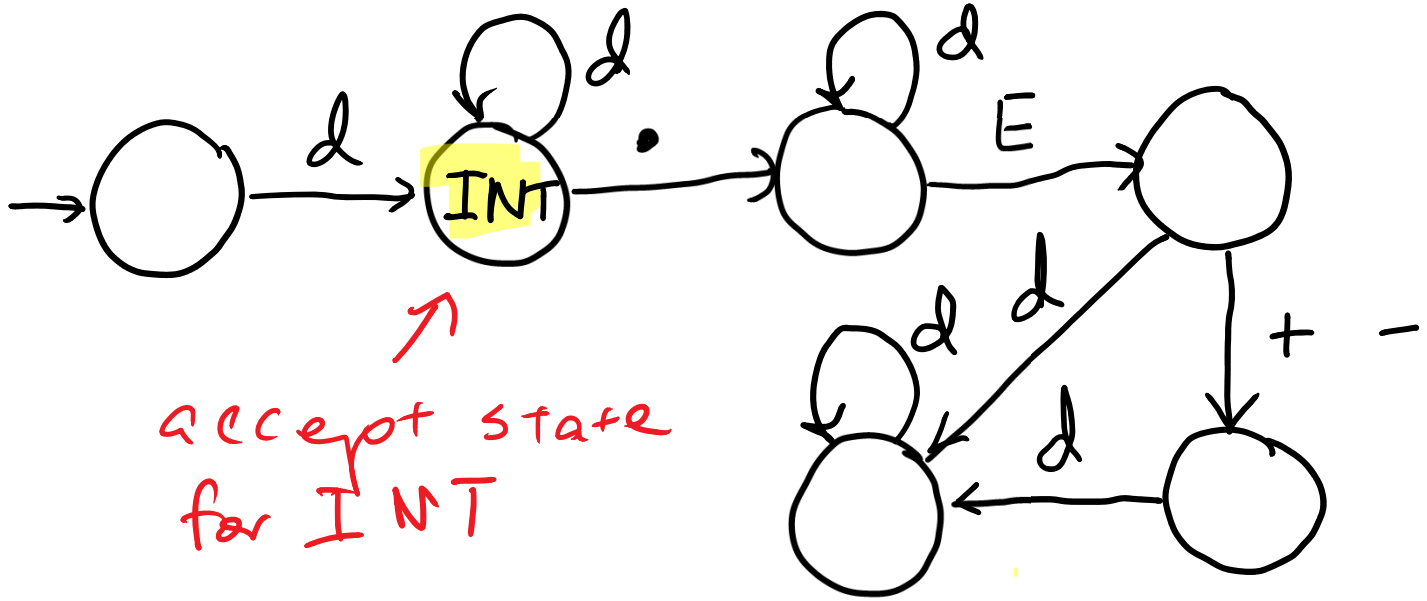
① Lexeme is prefix of input

② Multiple token types

Combined Automaton

Real: $digit^+ \cdot digit^* (E (+|-)? digit^+)$.

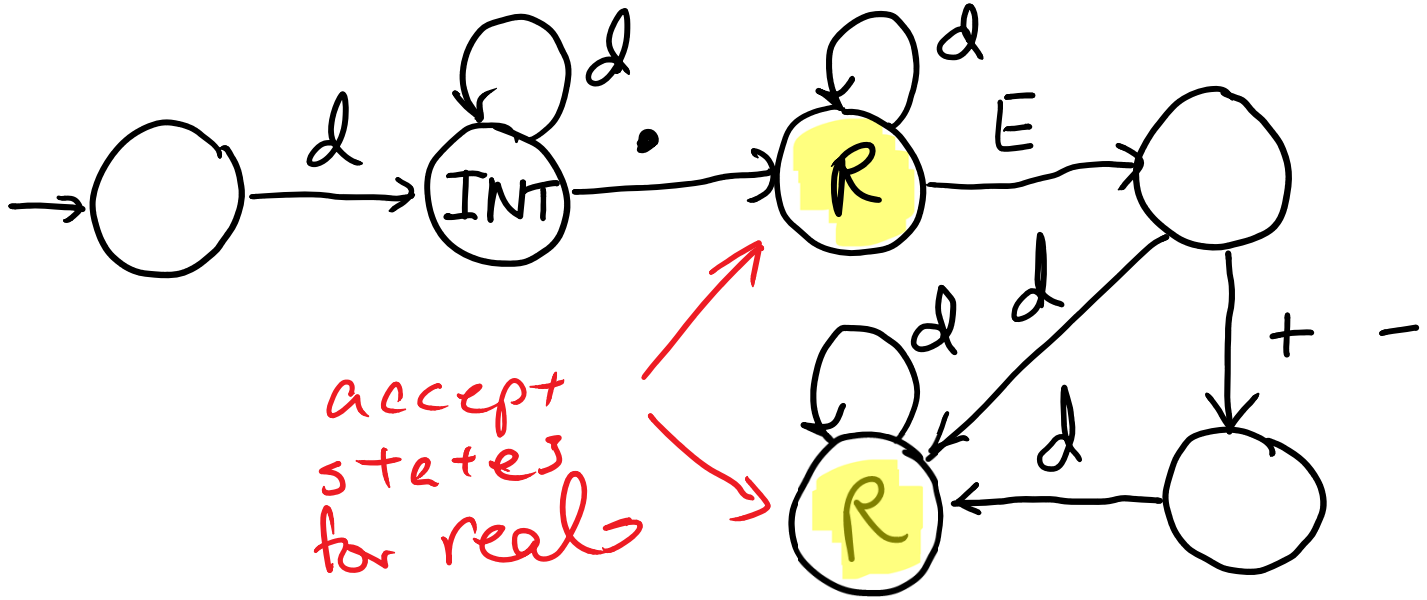
Int: $digit^+$



Combined Automaton

Real: $digit^+ \cdot digit^* (E (+|-)? digit^+)$.

Int: $digit^+$

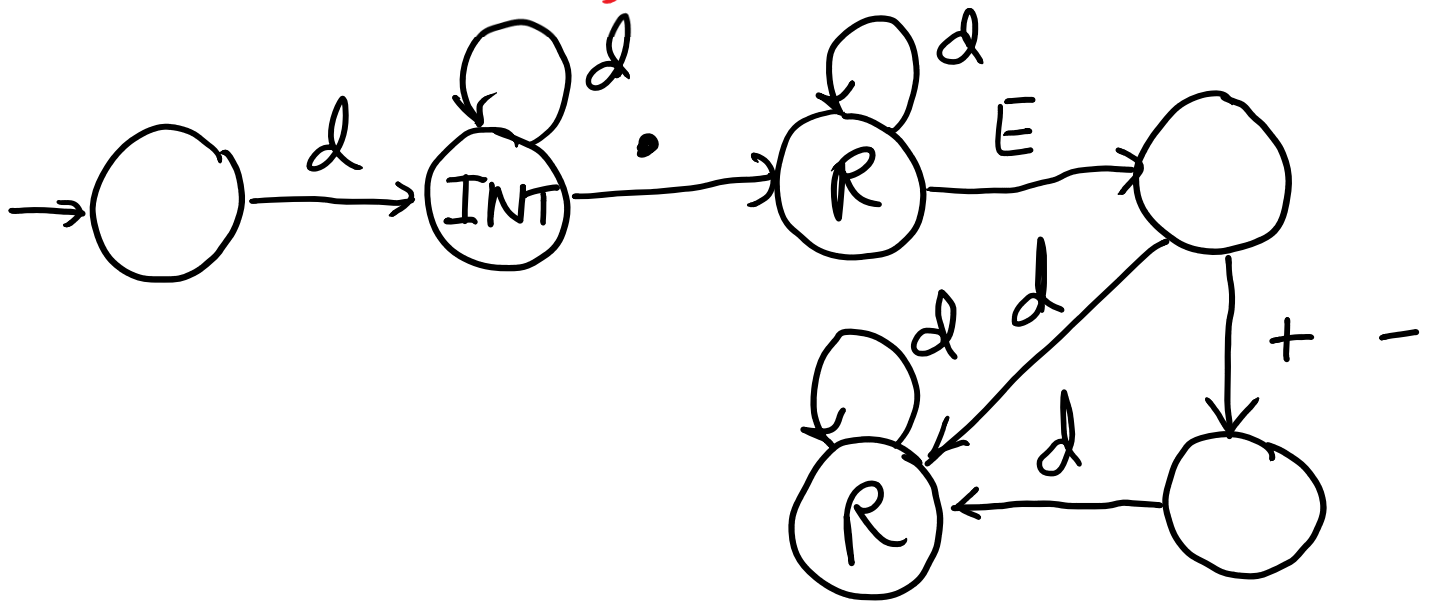


Input: 12.34+6

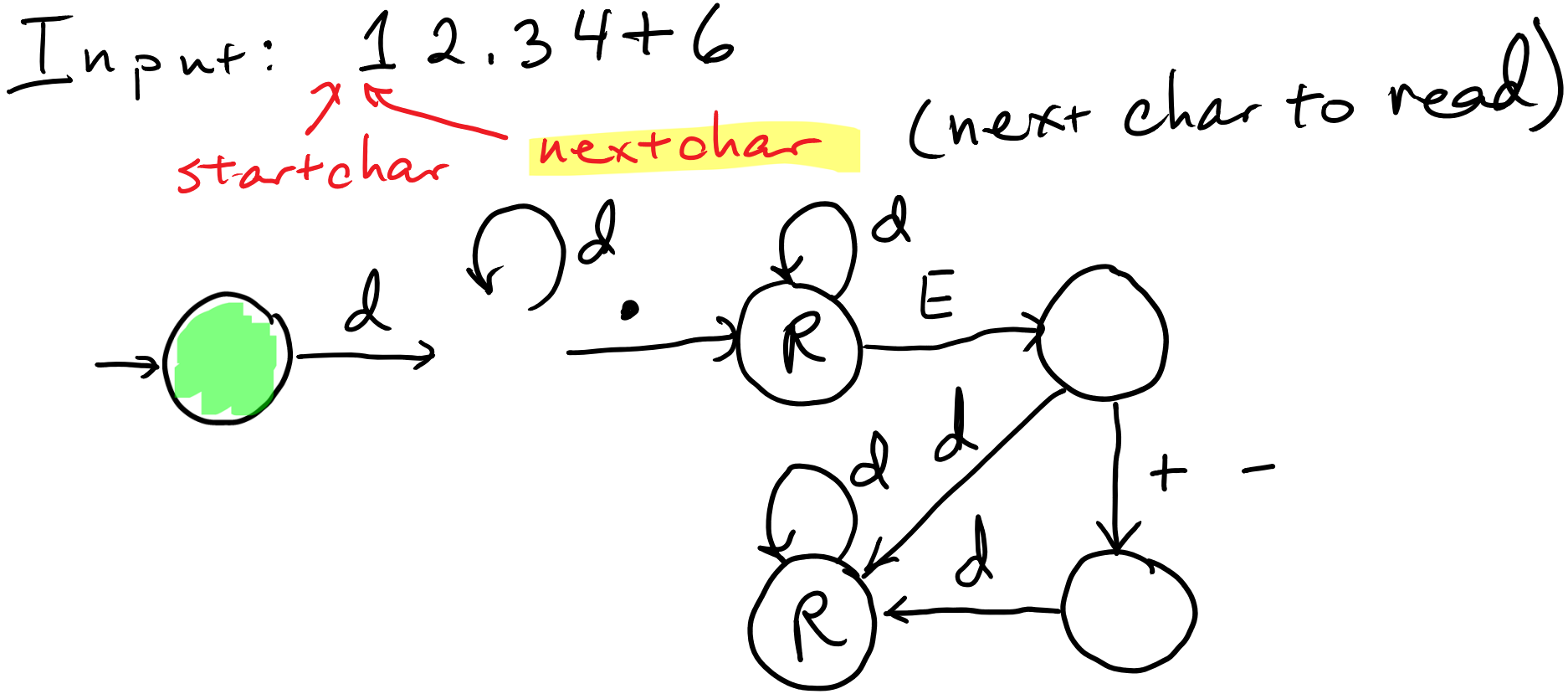
Tokens 1 2.3 4 ?

Longest lexeme rule ("maximal munch")

Tokens 12.34 ?



Input: 1 2 . 3 4 + 6

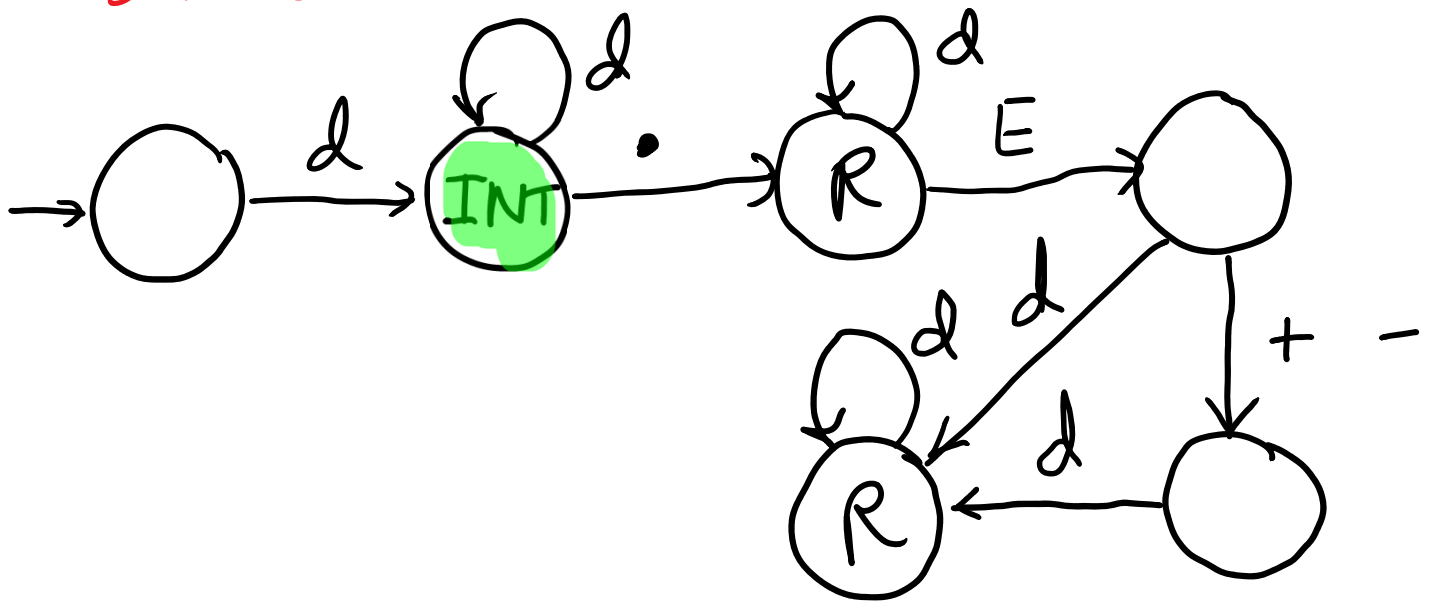


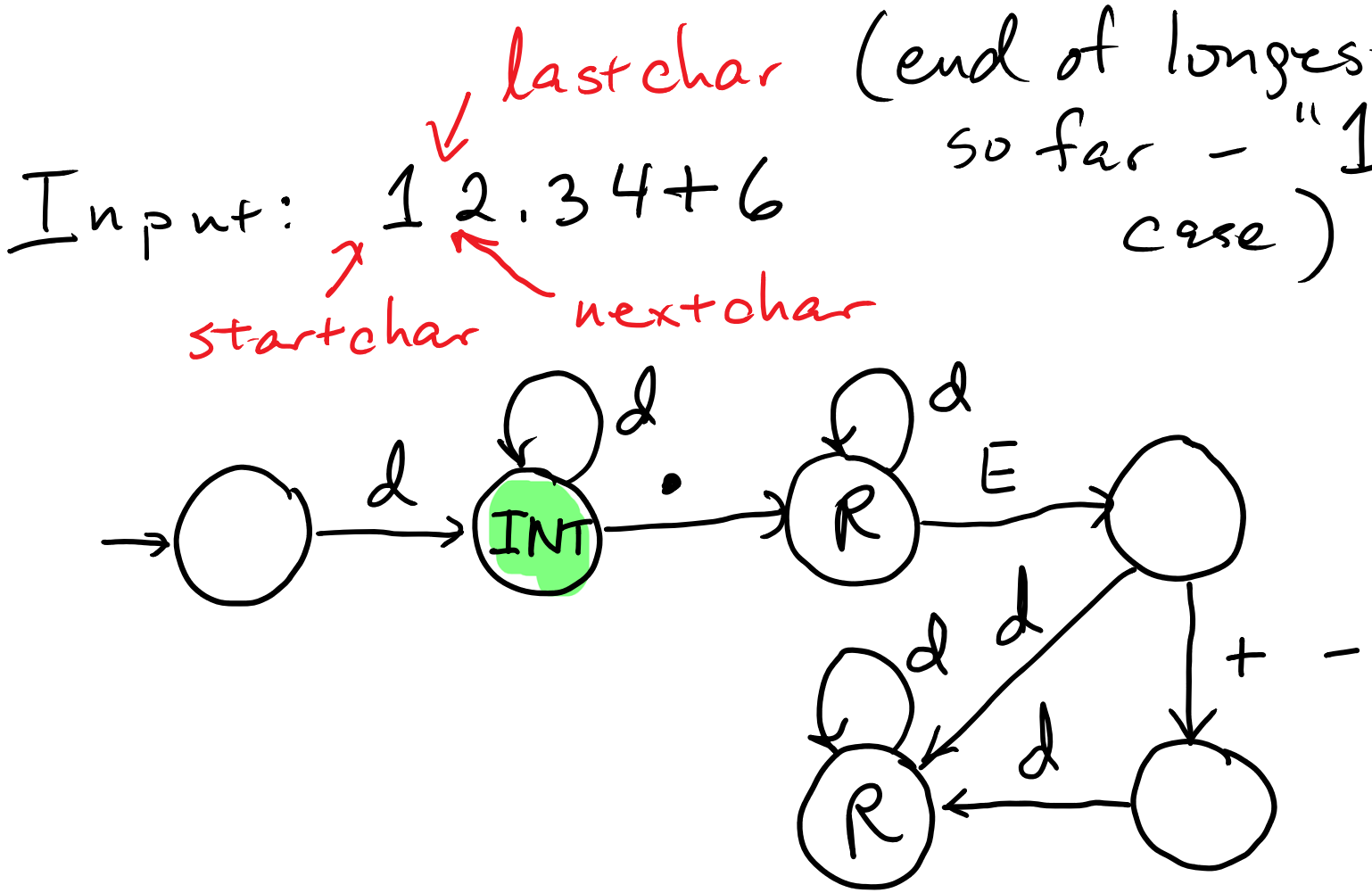
Input: 1 2 . 3 4 + 6

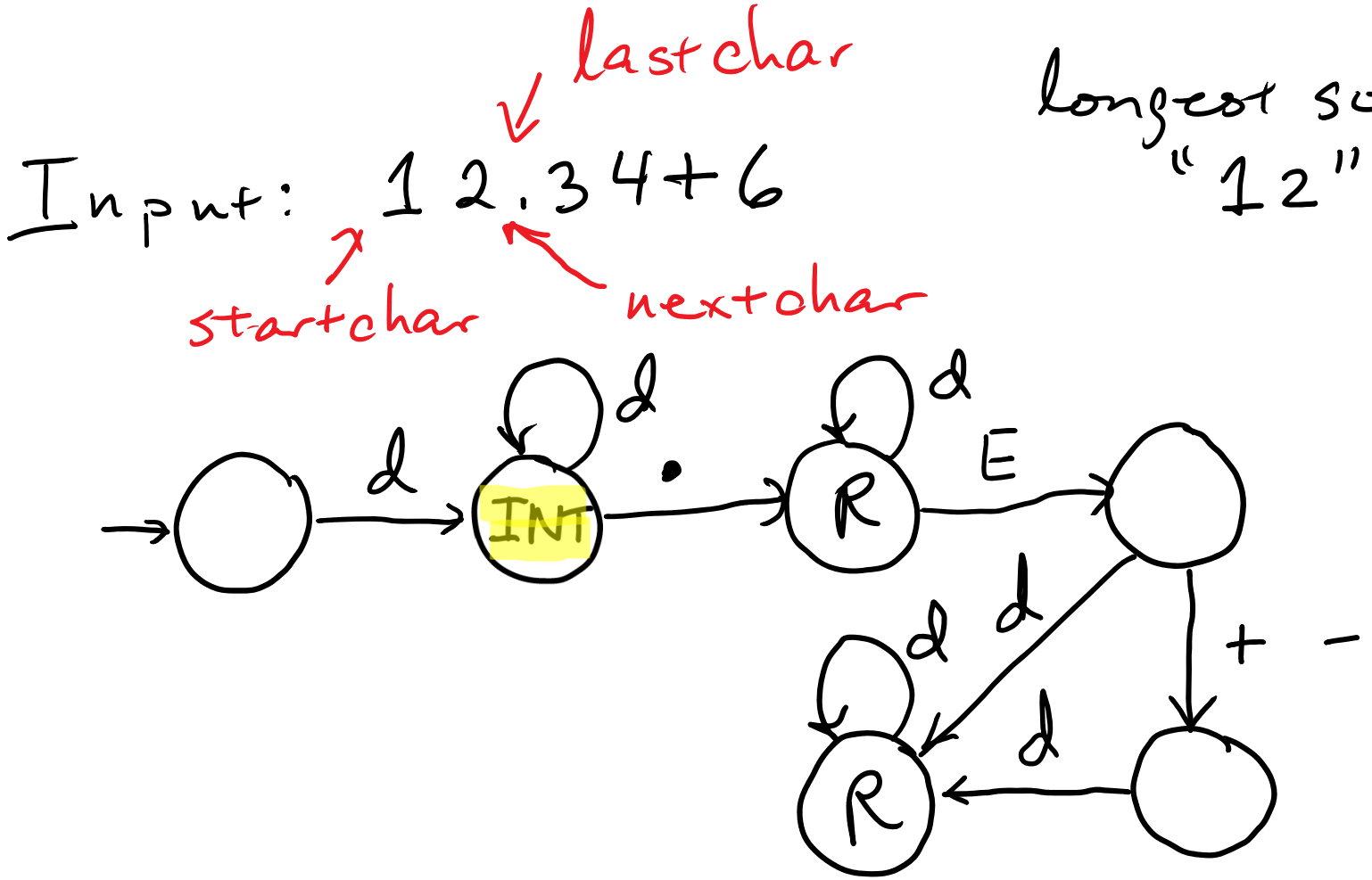
startchar

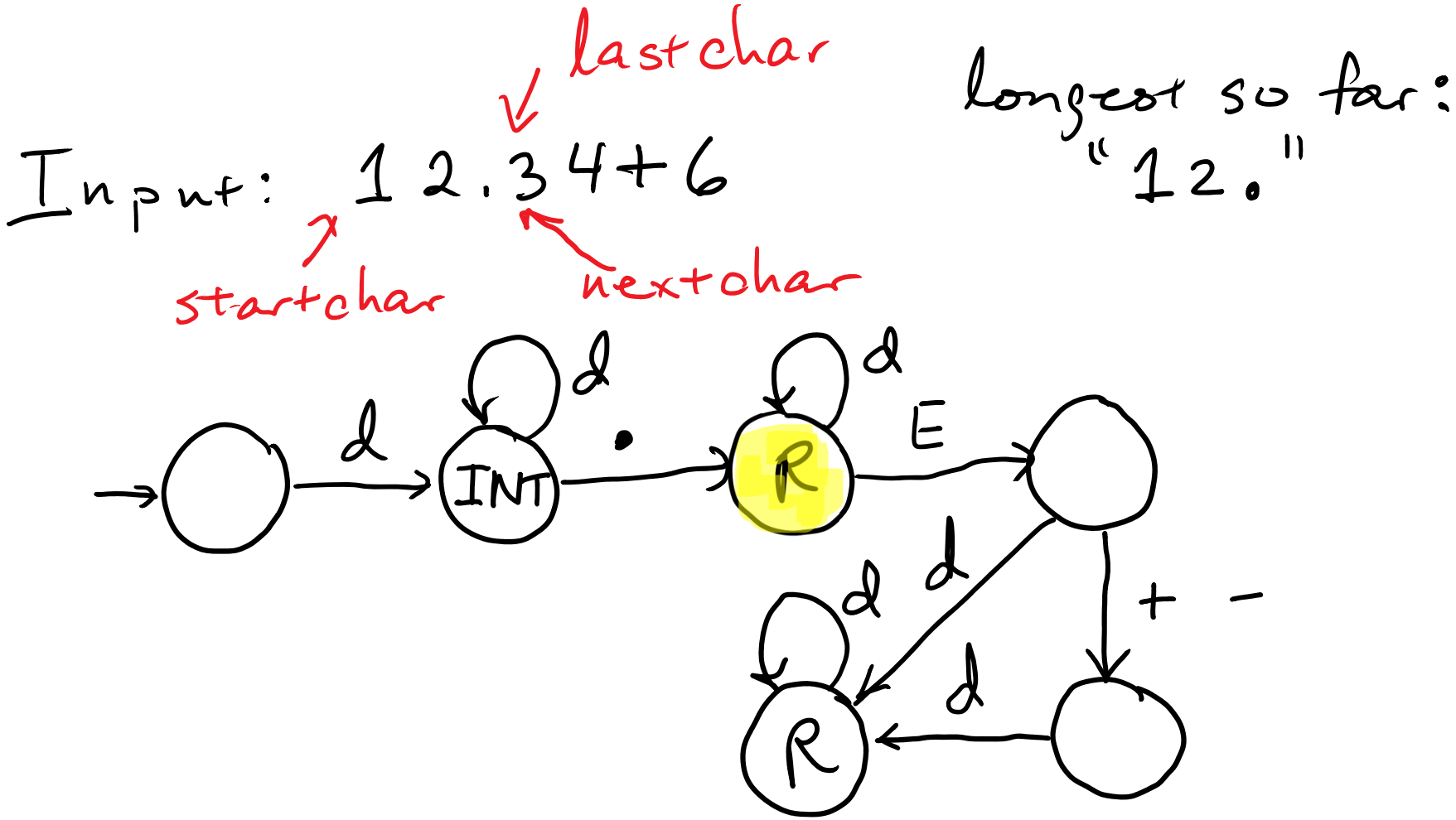
nextchar

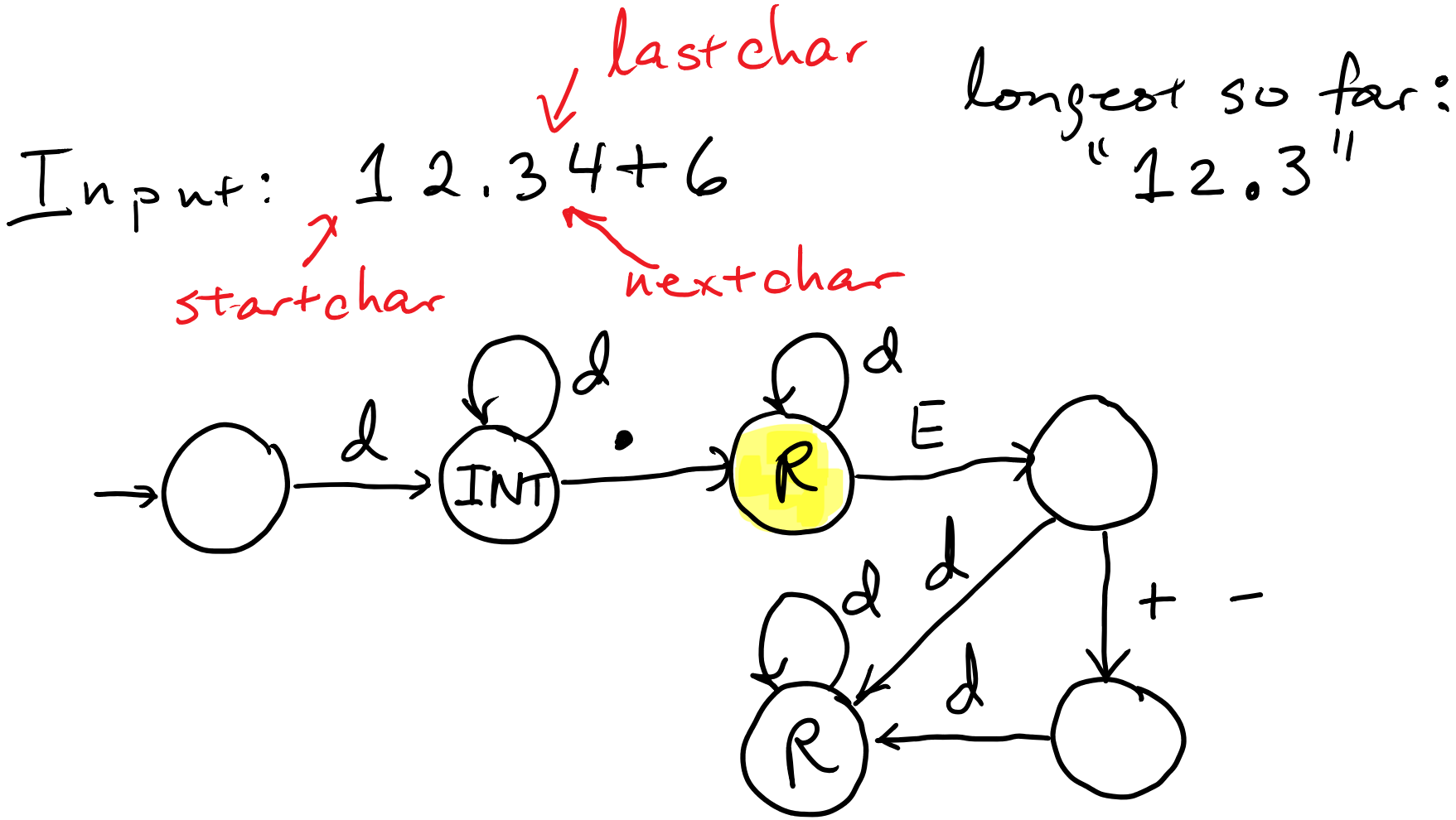
(advanced)

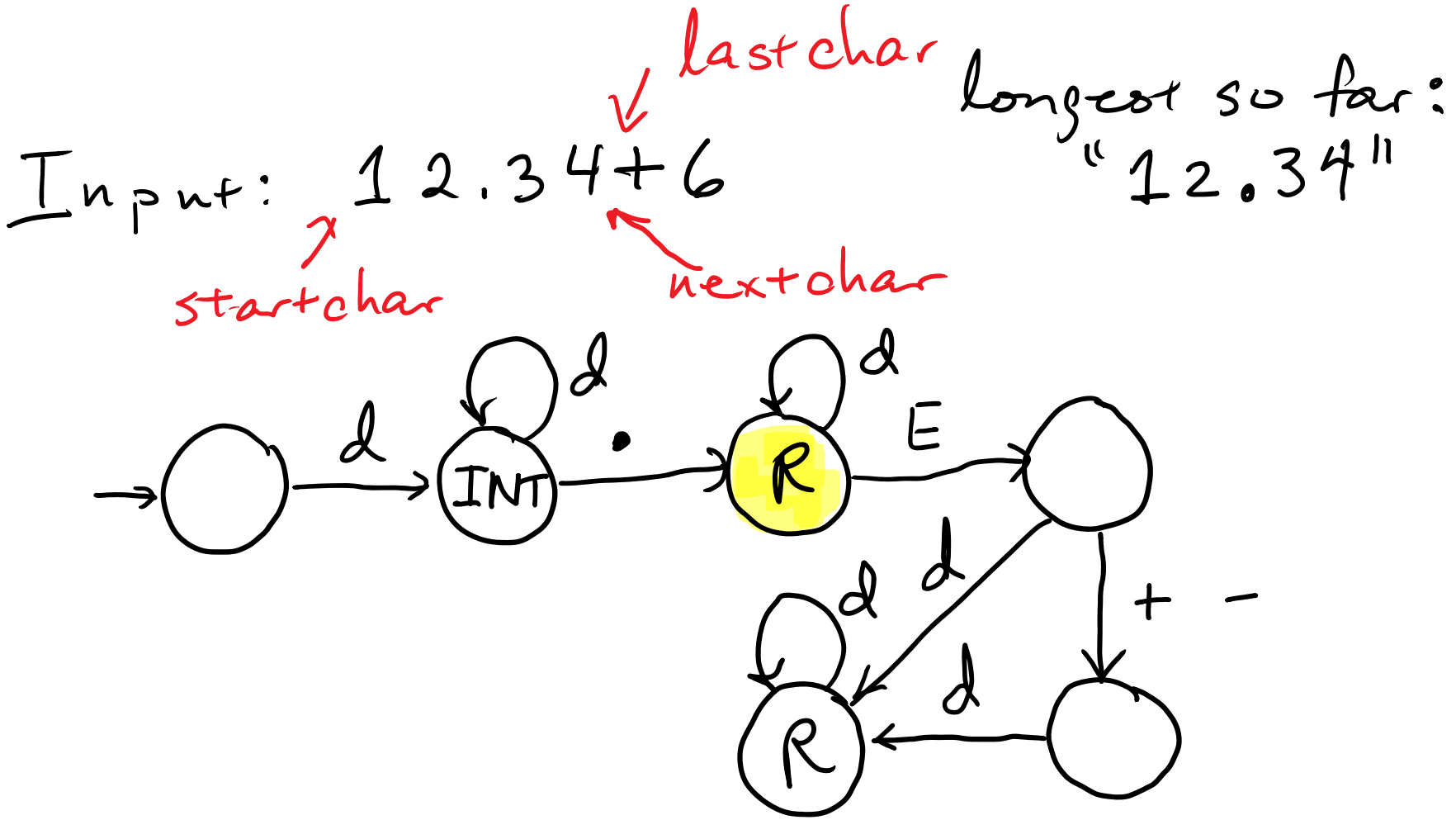


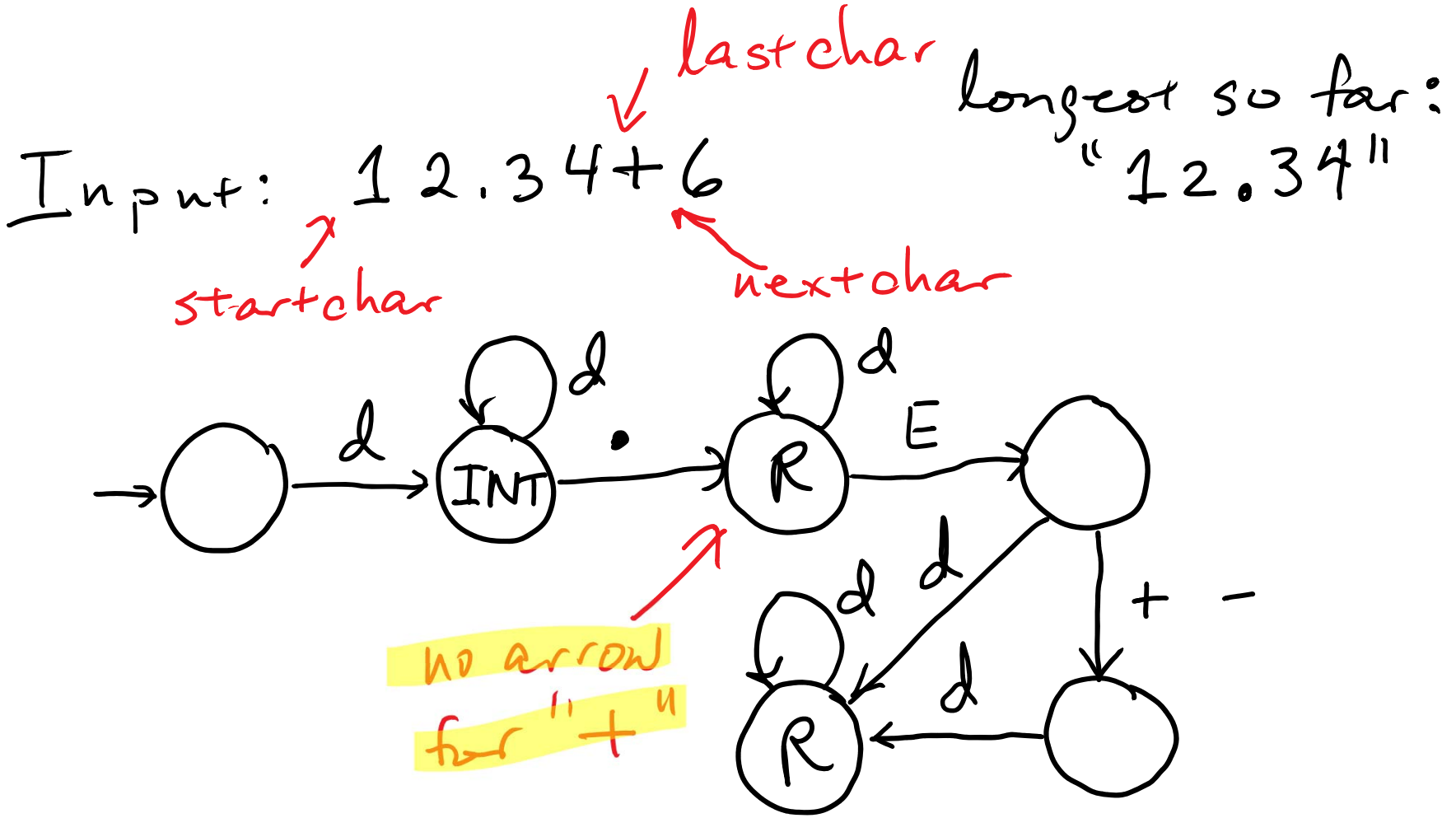

lastchar
(end of longest lexeme so far - "1" in this case)
Input: 12.34+6
startchar
nextchar
d
d
d
INT
.
R
d
E
d
d
+
-
d
R

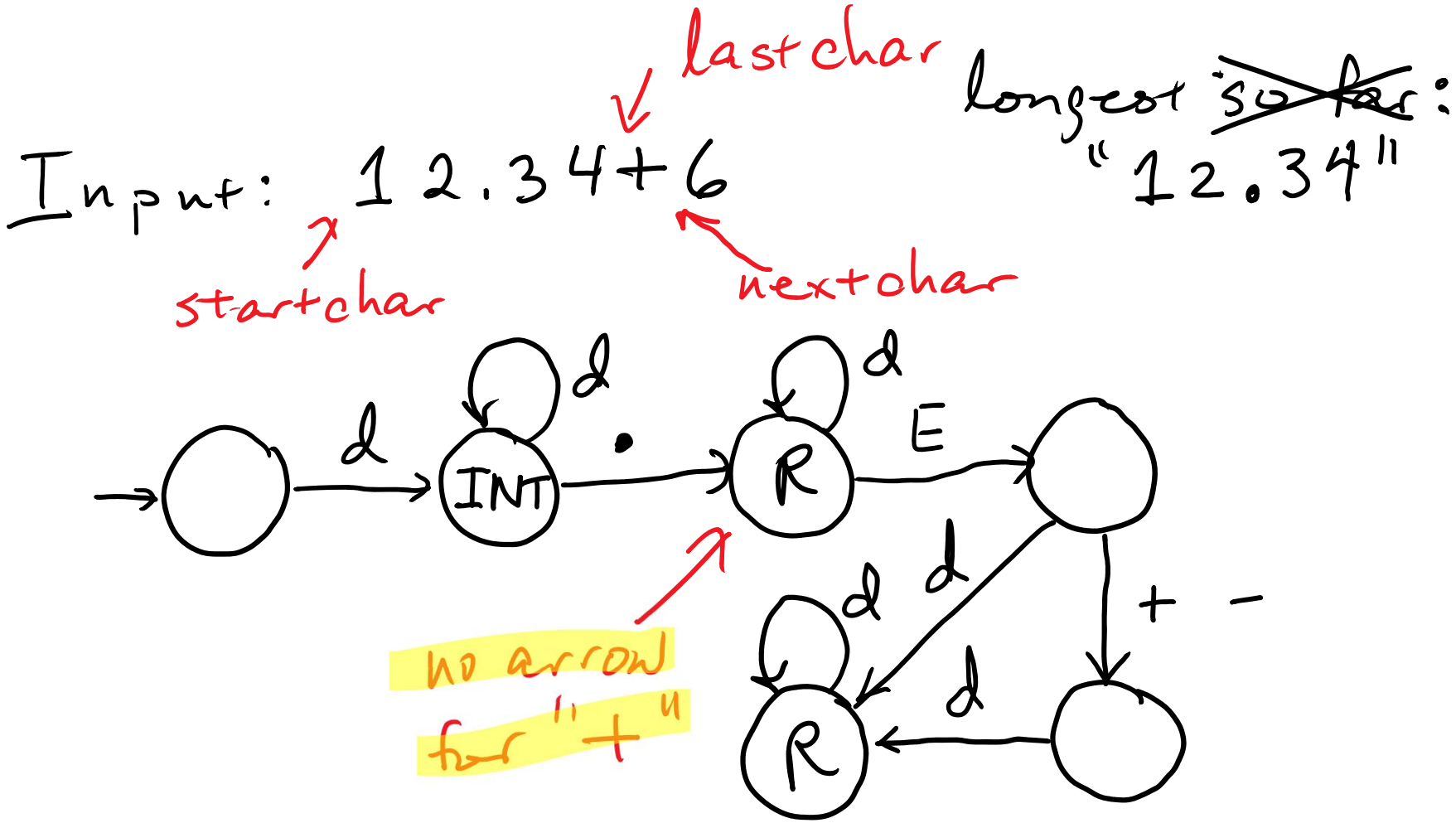

lastchar
Input: 12.34+6
startchar
nextchar
longest so far:
"12"
d
d
d
INT
.
R
d
E
d
d
+
-
d
R


lastchar
Input: 12.34+6
longest so far:
"12."
startchar
nextchar
d
d
d
INT
.
R
d
E
d
d
+
-
d
R


lastchar
Input: 12.34+6
longest so far:
"12.3"
startchar
nextchar
d
d
d
INT
.
R
d
E
d
d
+
-
d
R

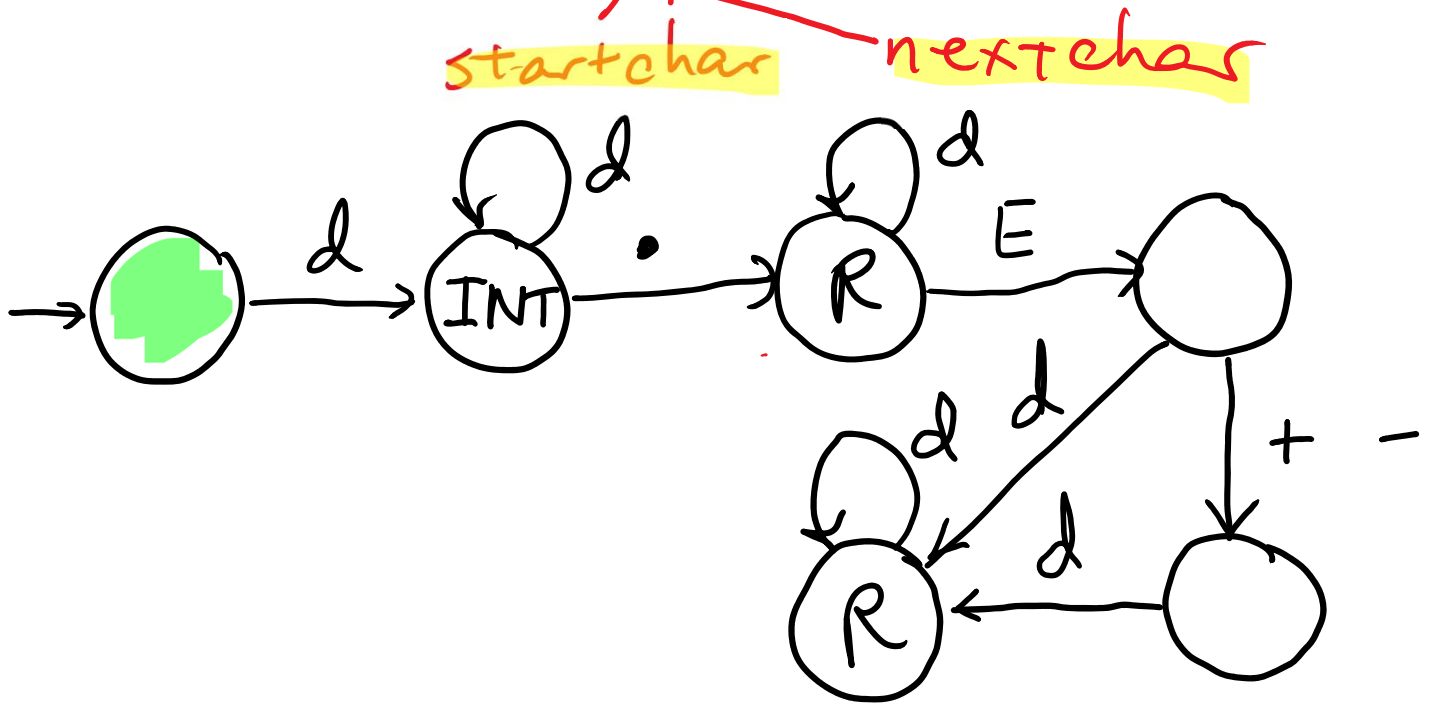

lastchar
longest so far:
"12.34"
Input: 12.34+6
startchar
nextchar
d
d
d
INT
.
R
d
E
d
d
+
-
d
R

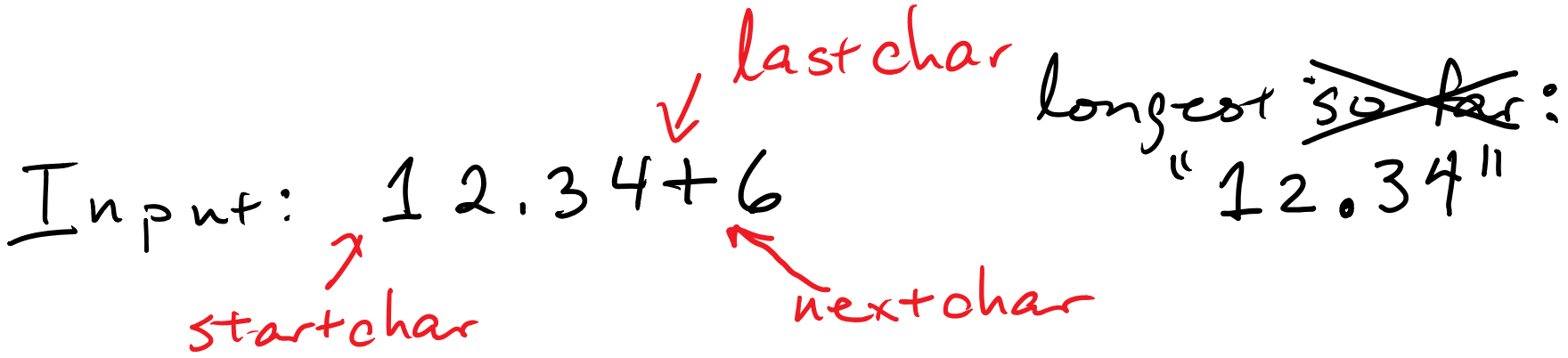

lastchar
longest so far:
"12.34"
Input: 12.34+6
startchar
nextchar
d
d
d
INT
.
R
d
E
d
d
+
-
d
R
no arrow
for "+"


lastchar
longest ~~so far~~:
"12.34"
Input: 12.34+6
startchar
nextchar
d
d
d
INT
.
R
E
d
d
+
-
d
d
R
no arrow
for "+"

Input: 1 2 . 3 4 + 6

Ready for next lexeme

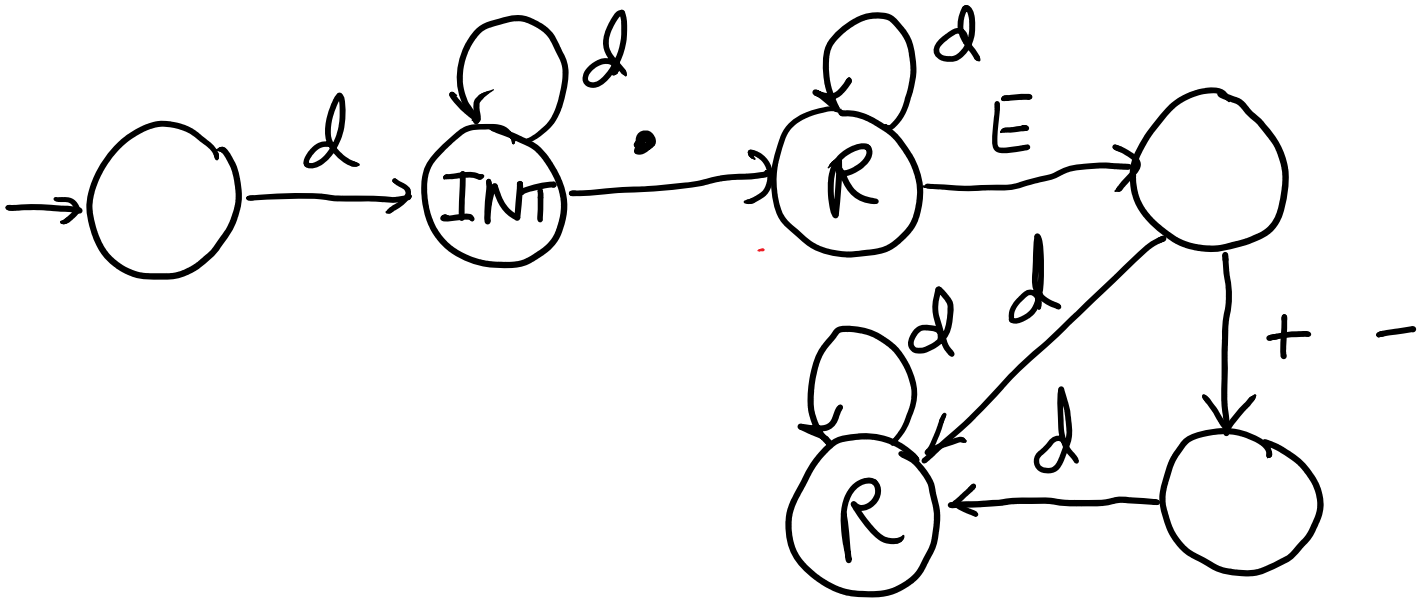




"Look ahead" – had to read "+" to see that end of lexeme was previous char.

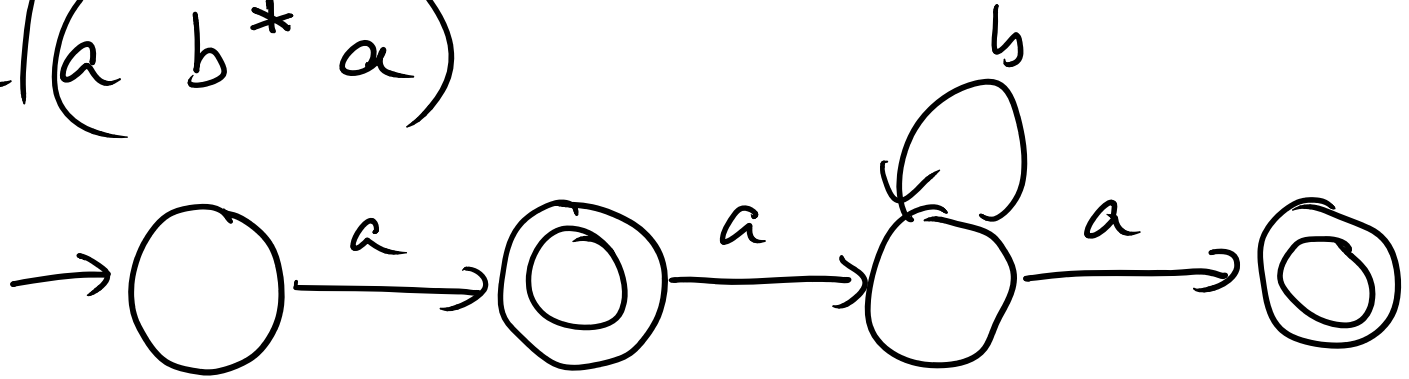
Lookahead = max(nextchar – lastchar)

Lookahead for this example is at least 1. Is it more?



Unbounded lookahead?

$a|(a\ b^*\ a)$



Input: a a b b b b b b b . . . . b

last char

next char

May need to read whole file to find first lexeme!

C Comments

/* ... */

Has unbounded lookahead if handled with a single pattern.

If closing "*/" is missing, tokenizes the comment: / * ...

Saves string unnecessarily

Start Conditions

A better way to handle comments

When you see "/*"

Use new DFA

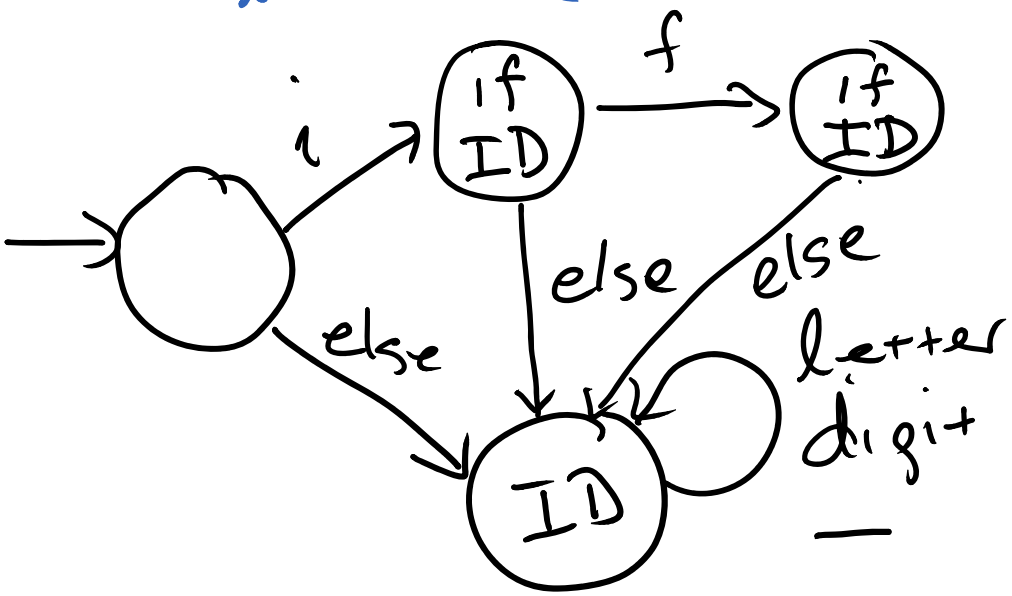
Patterns:

- any single char — discard
- "*/" - discard, return to main DFA

Rule Ordering

`if` `if`

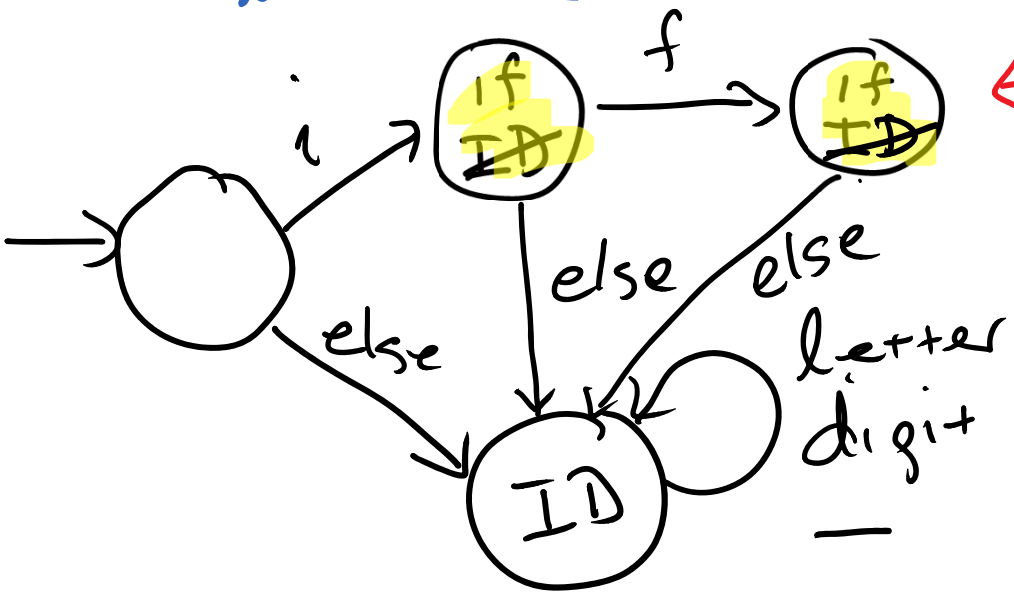
ID letter(letter | digit | _)



Rule Ordering

`if` `if`

ID letter (letter | digit | _)

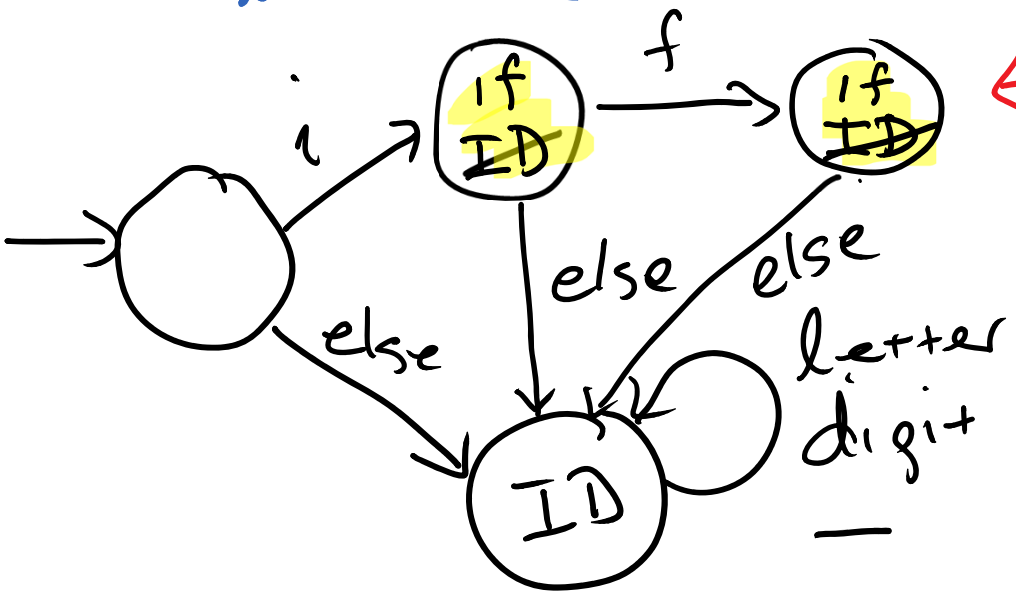


← label for first rule in file wins.

Rule Ordering

`if` `if` ← special case first

ID letter (letter | digit | _) ← default



← label for first rule in file wins.