

Chapter 2

Scanning: Capsule Summary

Course “Compiler Construction”

Martin Steffen

Spring 2021



Chapter 2

Learning Targets of Chapter "Scanning: Capsule Summary".

1. alphabets, languages
2. regular expressions
3. finite state automata / recognizers
4. connection between the two concepts
5. minimization

The material corresponds roughly to [**?** , Section 2.1–2.5] or a large part of [**?** , Chapter 2]. The material is pretty canonical, anyway.



Chapter 2

Outline of Chapter "Scanning: Capsule Summary".

Introduction

Regular languages and regular expressions

Finite-state automata (NFA, DFA)

Implementation aspects

From RE to NFA

Determinization

Minimization

Scanner implementation



Section

Introduction

Chapter 2 “Scanning: Capsule Summary”
Course “Compiler Construction”
Martin Steffen
Spring 2021

Scanner, lexer, tokenizer

- placement in the architecture
- input output



Notions and terminology

What's a

1. lexeme
2. token
3. token value and token class

Examples for that



Lexical aspects of a language

- what's a language?
- What makes an apect of a language **lexical**?



Tasks abstractly and concretely

abstractly

segment and **classify**

what's again lexemes and tokens in that connection?

concretely

1. whitespace
2. comments
3. reserved words, keywords
4. further classes like identifiers, numbers ...



Targets & Outline
Introduction
Regular languages and regular expressions
Finite-state automata (NFA, DFA)
Implementation aspects
From RE to NFA
Determinization
Minimization
Scanner implementation

Classification

name/identifier	`abc123`
integer constant	`42`
real number constant	`3.14E3`
text constant, string literal	`"this is a text constant"`
arithmetic op's	`+ - * /`
boolean/logical op's	`and or not` (alternatively `/\ \/ ~`)
relational symbols	`<= < >= > = == !=`
all other tokens: every one it its own group	`{ } ( ) [ ] , ; := .` etc.



Targets & Outline
Introduction
Regular languages and regular expressions
Finite-state automata (NFA, DFA)
Implementation aspects
From RE to NFA
Determinization
Minimization
Scanner implementation

How to specify a lexer?

short mentioning in preview of parser generators.



Targets & Outline
Introduction
Regular languages and regular expressions
Finite-state automata (NFA, DFA)
Implementation aspects
From RE to NFA
Determinization
Minimization
Scanner implementation

2 Lexical aspects

2.1 Identifiers and literals

- `NAME` must start with a letter, followed by a (possibly empty) sequence of numeric characters, letters, and underscore characters; the underscore is not allowed to occur at the end. Capital and small letters are considered different.
- All *keywords* of the languages are written in with lower-case letters. Keyword *cannot* be used for standard identifiers.
- `INT_LITERAL` contains one or more numeric characters.
- `FLOAT_LITERAL` contains one or more numeric characters, followed by a decimal point sign, which is followed by one or more numeric characters.
- `STRING_LITERAL` consists of a string of characters, enclosed in quotation marks (`"`). The string is not allowed to contain line shift, new-line, carriage return, or similar. The semantic *value* of a `STRING_LITERAL` is only the string itself, the quotation marks are not part of the string value itself.

2.2 Comments

Compila supports *single line* and *multi-line* comments.

1. Single-line comments start with `//` and the comment extends until the end of that line (as in, for instance, Java, C++, and most modern C-dialects).
2. Multi-line comments start with `(*` and end with `*)`.

The latter form cannot be nested. The first one is allowed to be "nested" (in the sense that a commented out line can contain another `//` or the multi-line comment delimiters, which are then ignored).





Section

Regular languages and regular expressions

Chapter 2 “Scanning: Capsule Summary”
Course “Compiler Construction”
Martin Steffen
Spring 2021

General remarks

- applications and uses of regular expression
- use in the context of a lexer



Basic notions



Definition (Alpabet Σ)

Finite set of elements called "letters" or "symbols" or "characters".

Definition (Words and languages over Σ)

Given alphabet Σ, a **word** over Σ is a finite sequence of letters from Σ. A **language** over alphabet Σ is a *set* of finite *words* over Σ.

examples of alphabets

Targets & Outline
Introduction
Regular languages and regular expressions
Finite-state automata (NFA, DFA)
Implementation aspects
From RE to NFA
Determinization
Minimization
Scanner implementation

How to describe languages?

remember

languages are infinite



Regular expressions



Targets & Outline
Introduction
Regular languages and regular expressions
Finite-state automata (NFA, DFA)
Implementation aspects
From RE to NFA
Determinization
Minimization
Scanner implementation

Definition (Regular expressions)

A *regular expression* is one of the following

1. a *basic* regular expression of the form $\mathbf{a}$ (with $a \in \Sigma$), or ϵ, or $\emptyset$
2. an expression of the form $r \mid s$, where r and s are regular expressions.
3. an expression of the form $r\,s$, where r and s are regular expressions.
4. an expression of the form r^*, where r is a regular expression.

Regular expressions as language

$$
\begin{aligned}
r &\rightarrow \mathbf{a} \\
r &\rightarrow \boldsymbol{\epsilon} \\
r &\rightarrow \emptyset \\
r &\rightarrow r \mid r \\
r &\rightarrow r\, r \\
r &\rightarrow r^*
\end{aligned}
\tag{1}
$$



Targets & Outline
Introduction
Regular languages and regular expressions
Finite-state automata (NFA, DFA)
Implementation aspects
From RE to NFA
Determinization
Minimization
Scanner implementation

Or like this

$$
\begin{array}{rcll}
R & \rightarrow & \mathbf{a} \mid \boldsymbol{\epsilon} \mid \boldsymbol{\emptyset} & \text{basic reg. expr.} \\
 & \mid & R \mid R \mid R\,R \mid R^* & \text{compound reg. expr.}
\end{array} \tag{2}
$$



Meaning of regular expressions

Definition not repeated here



Examples



Extended forms

- syntactic sugar.
- ordered alphabet



What's next (preview)

- FSA (NFA, DFA)
- determinism vs. non-determism
- what is determinism important?

Translating REs

RE $\Rightarrow$ NFA $\Rightarrow$ DFA $\Rightarrow$ minimal DFA

- automata as (basis for) lexers
- fine points
  - **priorities**
  - how to **tokenize**?



Targets & Outline
Introduction
Regular languages and regular expressions
Finite-state automata (NFA, DFA)
Implementation aspects
From RE to NFA
Determinization
Minimization
Scanner implementation



Section

Finite-state automata (NFA, DFA)

Chapter 2 “Scanning: Capsule Summary”
Course “Compiler Construction”
Martin Steffen
Spring 2021

Machine models

- regular expressions vs. machine model
- (finite-state) automata & variations
- language generator vs. "recognizer" (cf. also CF grammars)

Determinism

why important?



Targets & Outline
Introduction
Regular languages and regular expressions
Finite-state automata (NFA, DFA)
Implementation aspects
From RE to NFA
Determinization
Minimization
Scanner implementation

FSA



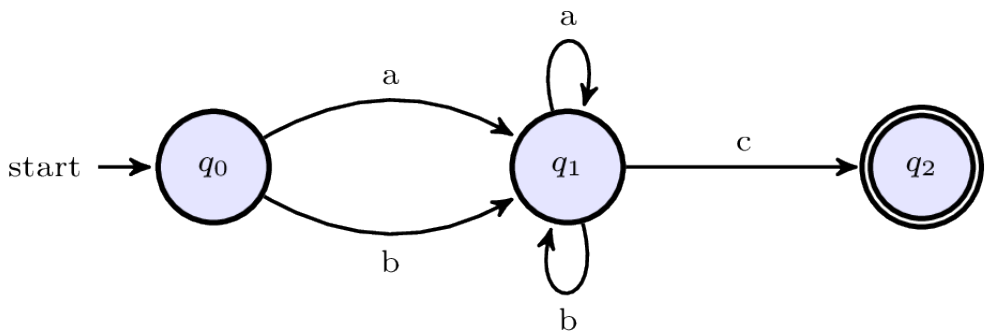
Definition (FSA)

A FSA $\mathcal{A}$ over an alphabet Σ is a tuple $(\Sigma, Q, I, F, \delta)$

- Q: finite set of states
- $I \subseteq Q$, $F \subseteq Q$: initial and final states.
- $\delta \subseteq Q \times \Sigma \times Q$ transition relation

Targets & Outline
Introduction
Regular languages and regular expressions
Finite-state automata (NFA, DFA)
Implementation aspects
From RE to NFA
Determinization
Minimization
Scanner implementation

Graphical representation



Instead of $(q_1, a, q_2) \in \delta$

$$q_1 \xrightarrow{a} q_2$$

$$q_1 \xrightarrow{a} q_2 \xrightarrow{b} q_3 \ .$$

Accepted words, language



Definition (Accepted words and language of an automaton)

A word $c_1 c_2 \ldots c_n$ with $c_i \in \Sigma$ is accepted by automaton $\mathcal{A}$ over Σ, if there exists states $q_0, q_2, \ldots, q_n$ from Q such that

$$q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} q_2 \xrightarrow{c_3} \quad \ldots \quad q_{n-1} \xrightarrow{c_n} q_n \ ,$$

and were $q_0 \in I$ and $q_n \in F$. The language of an FSA $\mathcal{A}$, written $\mathcal{L}(\mathcal{A})$, is the set of all words that $\mathcal{A}$ accepts.

Targets & Outline
Introduction
Regular languages and regular expressions
Finite-state automata (NFA, DFA)
Implementation aspects
From RE to NFA
Determinization
Minimization
Scanner implementation

FSA as scanning machine?

2 problematic aspects

- undefinedness
- non-determinism



DFA



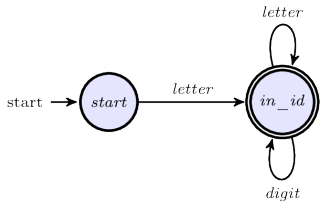
Definition (DFA)

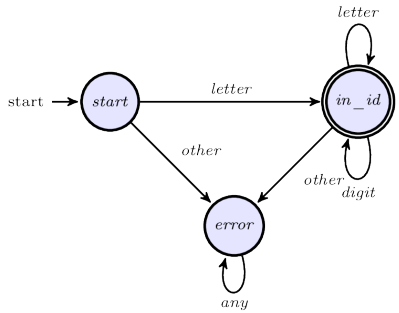
A *deterministic, finite automaton* $\mathcal{A}$ (DFA for short) over an alphabet Σ is a tuple $(\Sigma, Q, I, F, \delta)$

- Q: finite set of states
- $I = \{i\} \subseteq Q$, $F \subseteq Q$: initial and final states.
- $\delta : Q \times \Sigma \to Q$ transition function

Targets & Outline
Introduction
Regular languages and regular expressions
Finite-state automata (NFA, DFA)
Implementation aspects
From RE to NFA
Determinization
Minimization
Scanner implementation

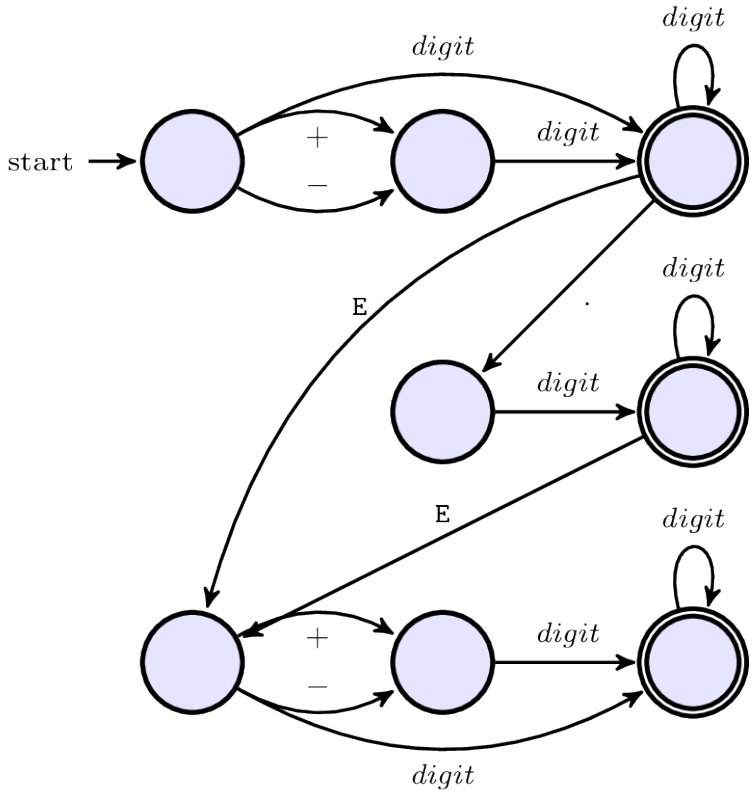
Deterministic (+ complete)





Series of number examples

see the use of EBNF (pile of definitions)





Section

Implementation aspects

Chapter 2 “Scanning: Capsule Summary”
Course “Compiler Construction”
Martin Steffen
Spring 2021

Scanner = DFA++

<= and<

```
if := if+6;
```

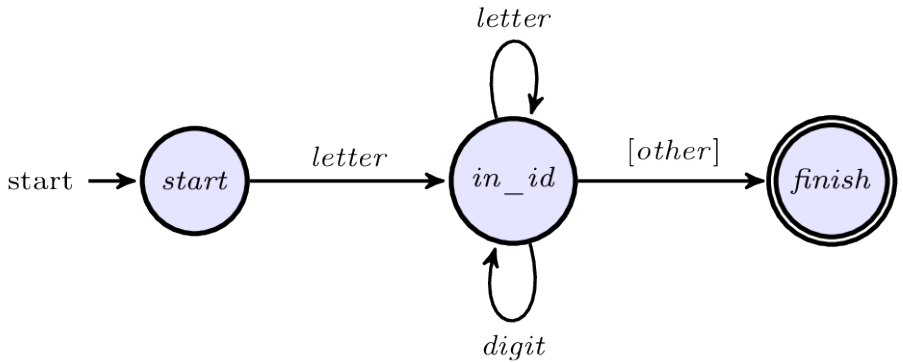
Prioritizing

- longest match
- keywords



DFA++

- tables





Section

From RE to NFA

Chapter 2 “Scanning: Capsule Summary”
Course “Compiler Construction”
Martin Steffen
Spring 2021

NFA with ϵ



Definition (NFA (with ϵ transitions))

A non-deterministic finite-state automaton (NFA for short) $\mathcal{A}$ over an alphabet Σ is a tuple $(\Sigma, Q, I, F, \delta)$, where

- Q: finite set of states
- $I \subseteq Q$, $F \subseteq Q$: initial and final states.
- $\delta : Q \times \Sigma \to 2^Q$ transition function

In case, one uses the alphabet $\Sigma + \{\epsilon\}$, one speaks about an NFA with ϵ-transitions.

Language: $\mathcal{L}(\mathcal{A})$

ϵ's don't count

Targets & Outline
Introduction
Regular languages and regular expressions
Finite-state automata (NFA, DFA)
Implementation aspects
From RE to NFA
Determinization
Minimization
Scanner implementation

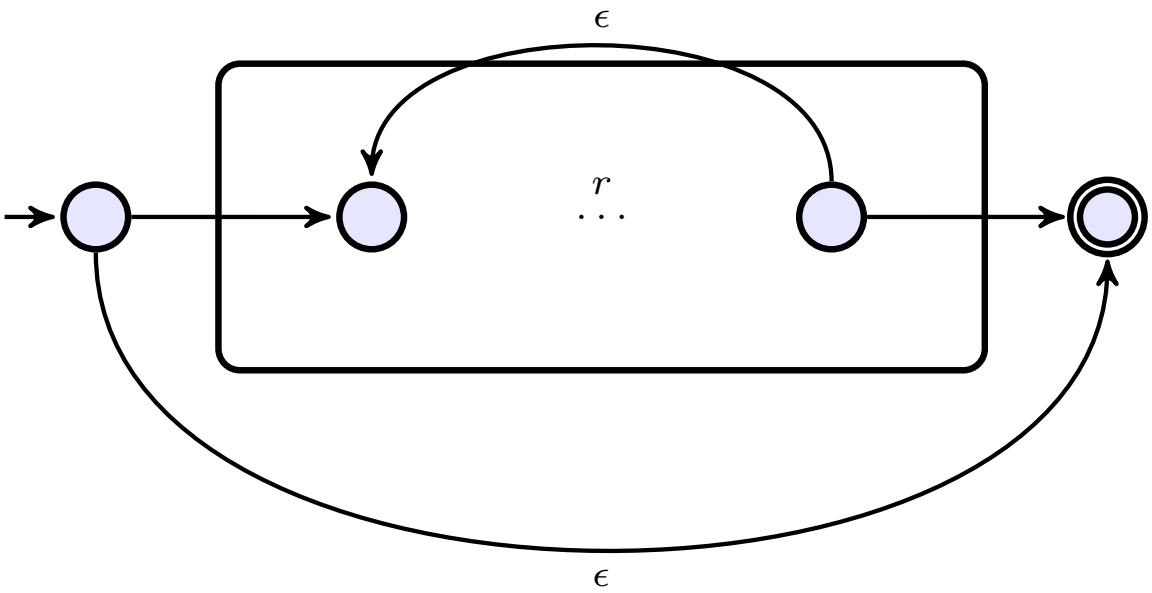
Thompson's construction

- construct an NFA, not DFA
- easier, compositional
- uses lot of ϵ-transitions
- inductive
- one case per syntax construct



Targets & Outline
Introduction
Regular languages and regular expressions
Finite-state automata (NFA, DFA)
Implementation aspects
From RE to NFA
Determinization
Minimization
Scanner implementation

Case for star







Section

Determinization

Chapter 2 “Scanning: Capsule Summary”
Course “Compiler Construction”
Martin Steffen
Spring 2021

Determinization

- avoid backtracking

Subset construction: DFA $\overline{\mathcal{A}}$

- new states: sets of states of $\mathcal{A}$
- a-transition in $\overline{\mathcal{A}}$: move to the "meta"-state of $\overline{\mathcal{A}}$that represents all a-successors in A

- watch out for ϵ-transitions
- ϵ-closure



ϵ-closure (left out of the discussion)



ϵ-closure (left out of the discussion)



Definition (ϵ-closure, a-successors)

Given a state q, the *ϵ-closure* of q, written $close_\epsilon(q)$, is the set of states reachable via zero, one, or more ϵ-transitions. We write q_a for the set of states, reachable from q with one a-transition. Both definitions are used analogously for sets of states.

Targets & Outline
Introduction
Regular languages and regular expressions
Finite-state automata (NFA, DFA)
Implementation aspects
From RE to NFA
Determinization
Minimization
Scanner implementation

Determinization (left out of the discussion)



Determinization (left out of the discussion)



Input: NFA $\mathcal{A}$ over a given Σ
Output: DFA $\overline{\mathcal{A}}$

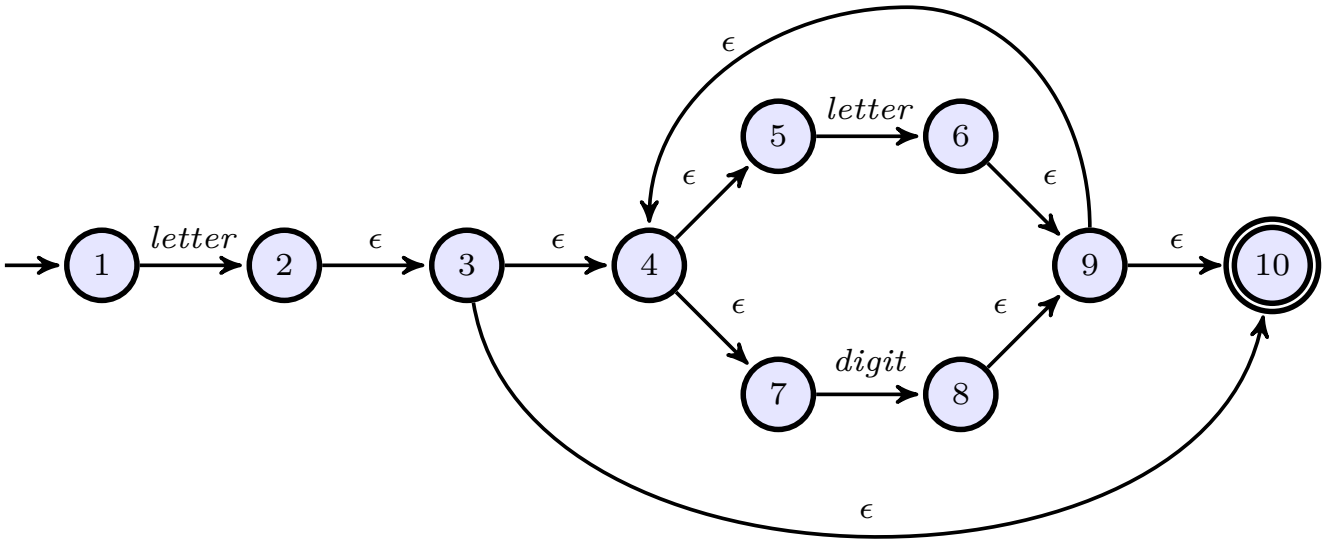
1. the *initial* state: $close_\epsilon(I)$, where I are the initial states of $\mathcal{A}$
2. for a state Q in $\overline{\mathcal{A}}$: the *a-successor* of Q is given by $close_\epsilon(Q_a)$, i.e.,

$$Q \xrightarrow{a} close_\epsilon(Q_a) \tag{3}$$

3. repeat step 2 for all states in $\overline{\mathcal{A}}$ and all $a \in \Sigma$, until no more states are being added
4. the *accepting* states in $\overline{\mathcal{A}}$: those containing *at least* one accepting state of $\mathcal{A}$

Targets & Outline
Introduction
Regular languages and regular expressions
Finite-state automata (NFA, DFA)
Implementation aspects
From RE to NFA
Determinization
Minimization
Scanner implementation

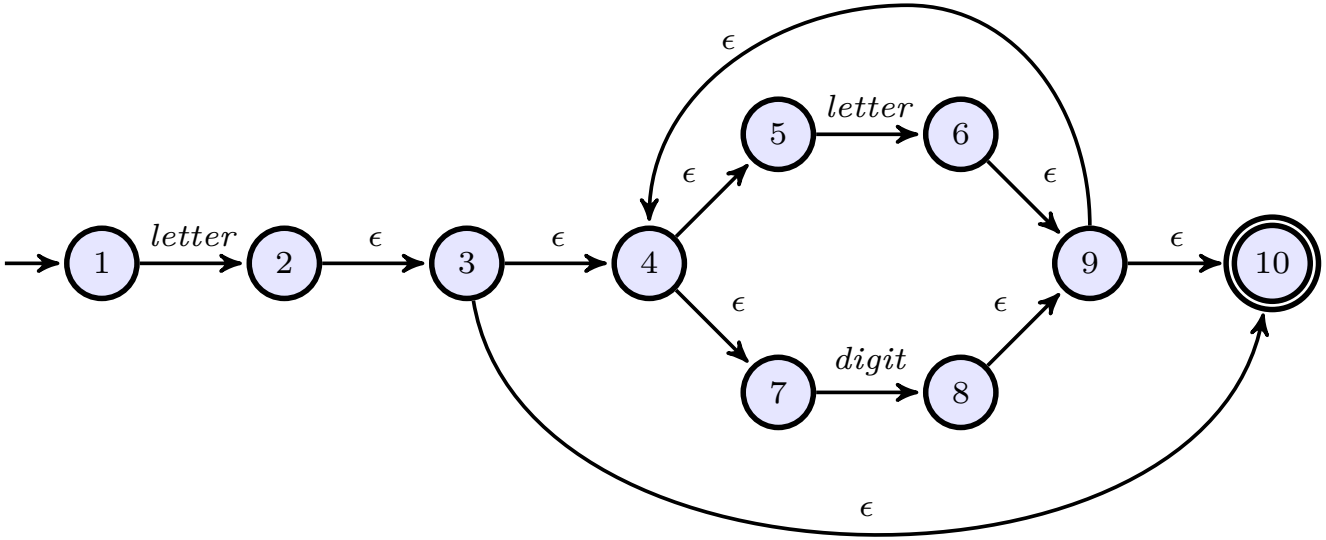
Example (more in the script)

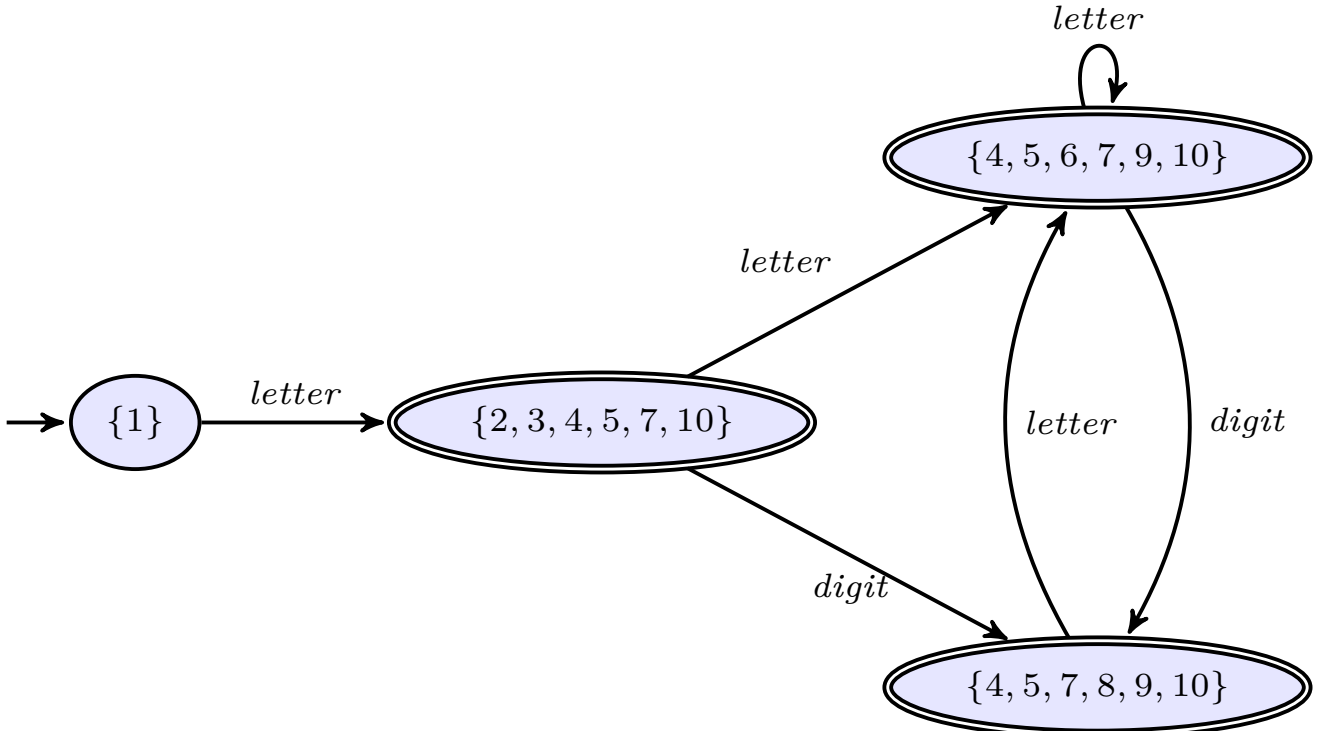




Targets & Outline
Introduction
Regular languages and regular expressions
Finite-state automata (NFA, DFA)
Implementation aspects
From RE to NFA
Determinization
Minimization
Scanner implementation

Example (more in the script)







Section

Minimization

Chapter 2 “Scanning: Capsule Summary”
Course “Compiler Construction”
Martin Steffen
Spring 2021

Minimization

Properties of minimization:

Minimality: the resulting DFA has *minimal* number of states.

Canonicity: all DFA for the same language result in the *same* minimal DFA.



Targets & Outline
Introduction
Regular languages and regular expressions
Finite-state automata (NFA, DFA)
Implementation aspects
From RE to NFA
Determinization
Minimization
Scanner implementation

Partition refinement

The **partition refinement** construction works the other way around! instead of collapsing equivalent states:

Initialization: start by collapsing as much as possible (details apply).

Iteration: iteratively, detect *non-equivalent* states, and then *split* a "collapsed" state

Termination: stop when no violations of "equivalence" are detected

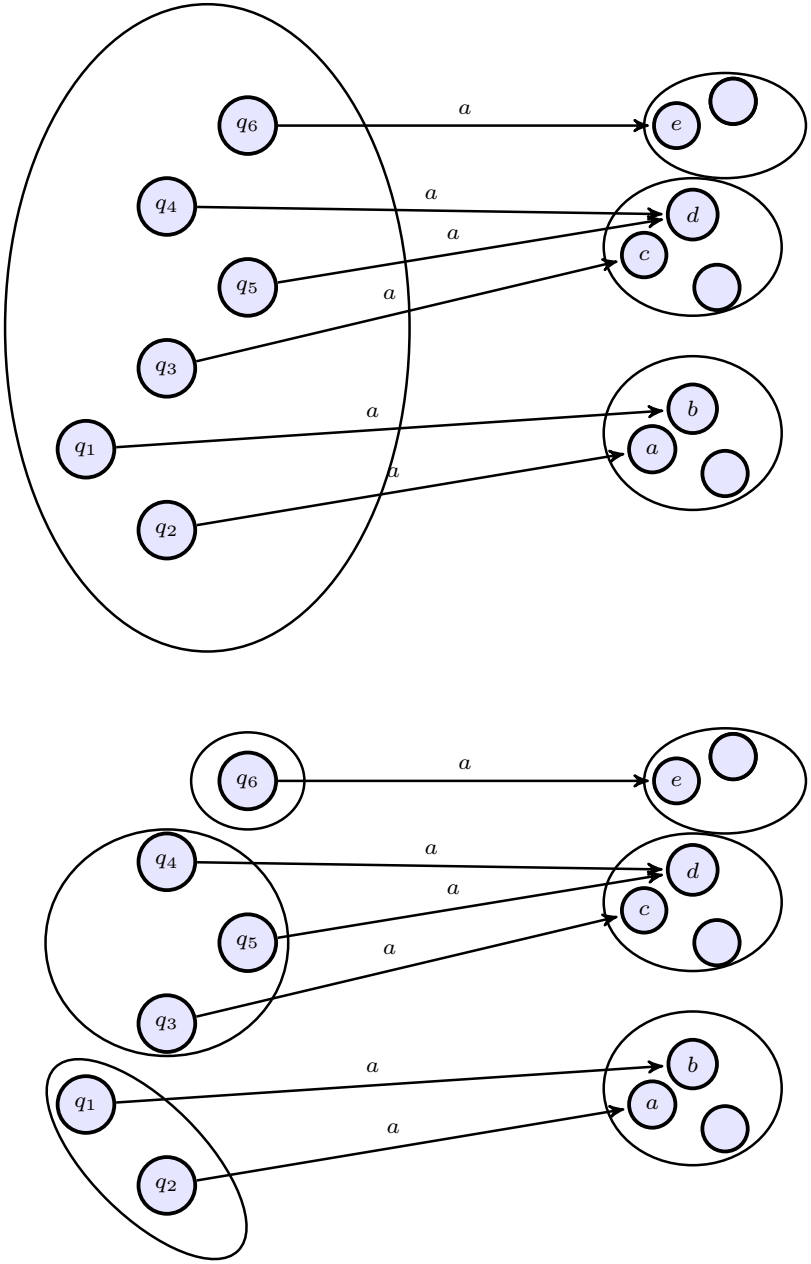


Algo (left out of the discussion)

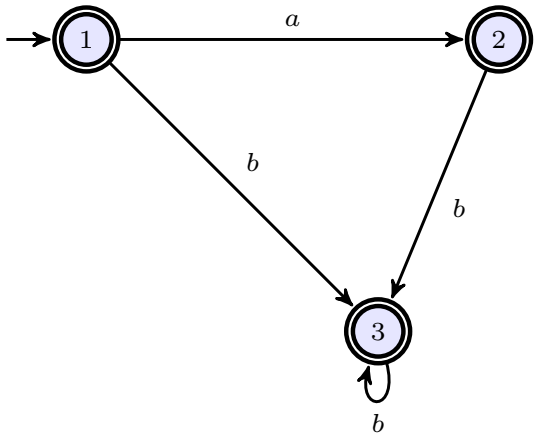
- **Initial** partitioning: 2 partitions: set containing all *accepting* states F, set containing all *non-accepting* states $Q \backslash F$
- **Loop** do the following: pick a current equivalence class Q_i and a symbol a
  - if for all $q \in Q_i$, $\delta(q, a)$ is member of the *same* class $Q_j \Rightarrow$ consider Q_i as **done** (for now)
  - else:
    - **split** Q_i into $Q_i^1, \ldots Q_i^k$ s.t. the above situation is repaired for each Q_i^l (but don't split more than necessary).
    - be aware: a split may have a "cascading effect": other classes being fine before the split of Q_i need to be reconsidered $\Rightarrow$ *worklist* algo
- **stop** if the situation stabilizes, i.e., no more split happens (= worklist empty, at latest if back to the original DFA)

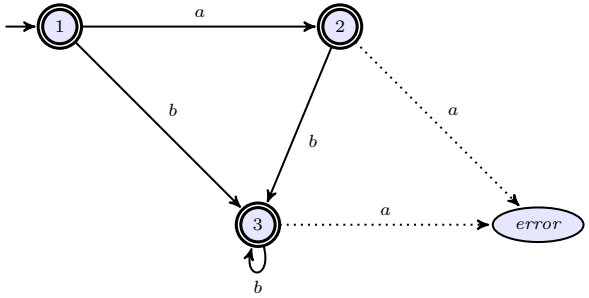


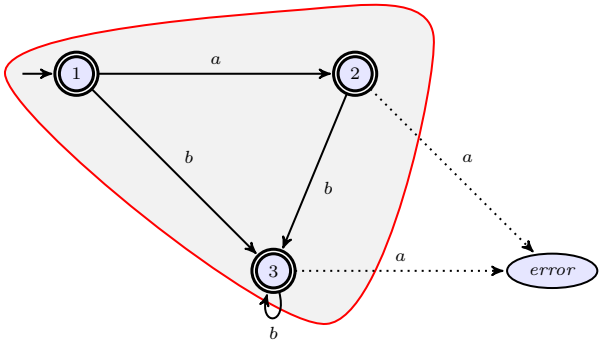
Split step (on a) of the state on the left

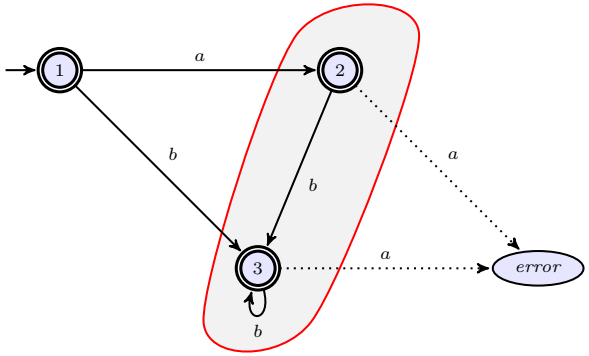


Example (another one in the script)











Section

Scanner implementation

Chapter 2 “Scanning: Capsule Summary”
Course “Compiler Construction”
Martin Steffen
Spring 2021

Lex and friends

- short section only
- see also the oblig later

```
1
2  DIGIT    [0-9]
3  ID       [a-z][a-z0-9]*
4
5  %%
6
7  {DIGIT}+    {
8              printf( "An integer: %s (%d)\n", yytext,
9                      atoi( yytext ) );
10             }
11
12 {DIGIT}+"."{DIGIT}*        {
13             printf( "A float: %s (%g)\n", yytext,
14                     atof( yytext ) );
15             }
16
17 if|then|begin|end|procedure|function        {
18             printf( "A keyword: %s\n", yytext );
19             }
```



Targets & Outline
Introduction
Regular languages and regular expressions
Finite-state automata (NFA, DFA)
Implementation aspects
From RE to NFA
Determinization
Minimization
Scanner implementation

What's next

Two big chapters

- Context-free grammars
- Parsing

some aspects from here will reappear (in different form):

Issues

- **automaton** model, then with **stack**, to **recognize** (the syntax of) a language
- importance of **determinism**
  - (un-)**ambiguity** in a grammar
  - absence of **conflicts** in a parser machine



Targets & Outline
Introduction
Regular languages and regular expressions
Finite-state automata (NFA, DFA)
Implementation aspects
From RE to NFA
Determinization
Minimization
Scanner implementation