DD1362
Programming Paradigms

Formal Languages and Syntactic Analysis
Lecture 1

Philipp Haller

March 29th, 2021

About Myself

- 2006 Dipl.-Inform.
  Karlsruhe Institute of Technology (KIT), Germany

- 2010 Ph.D. in Computer Science
  Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland

- Jan 2011—Jan 2012 Postdoctoral fellow
  Stanford University, USA and **EPFL, Switzerland**

- Feb 2012—Nov 2014 Consultant and software engineer
  Typesafe, Inc.

  Lightbend

- Dec 2014—Nov 2018 Assistant Professor of Computer Science
  Dec 2018—present Associate Professor of Computer Science
  Jun 2018 Docent in Computer Science
  KTH Royal Institute of Technology, Stockholm, Sweden

Formal Languages

Languages Formally

- A ***word*** is a finite, possibly empty, sequence of elements from some set Σ

  Σ – *alphabet*, Σ^* - set of all words over Σ

- By a ***language*** we mean a subset of Σ^*
- uv denotes the concatenation of words u and v
- Concatenation of languages and Kleene star:

$$L_1 L_2 = \{ u_1 u_2 \mid u_1 \text{ in } L_1 , u_2 \text{ in } L_2 \}$$

$$L^0 = \{\varepsilon\} \qquad \varepsilon = \text{empty word} = \text{empty sequence}$$

$$L^{k+1} = L L^k \qquad L^* = \cup_k L^k \quad \text{(Kleene star)}$$

Examples of Languages

$\Sigma = \{a, b\}$

$\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, \ldots \}$

Examples of two languages (subsets of Σ^*):

$L_1 = \{a, bb, ab\}$ (finite language, three words)

$L_2 = \{ab, abab, ababab, \ldots \}$
$= \{ (ab)^n \mid n > 0 \}$ (infinite language)

Examples of Operations

L = { **a**, **ab** }

L L = { **aa**, **aab**, **aba**, **abab** }

L* = { **ε**, **a**, **ab**, **aa**, **aab**, **aba**, **abab**, **aaa**, ... }

(is **bb** inside **L*** ?)

= { **w** | immediately before each **b** there is **a** }

Formal Languages and Compilers

- ***Lexical analyzer of a compiler*** recognizes the different ***tokens*** of a programming language
  - Keywords: `class`, `while`, `if`, …
  - Names of variables, parameters, methods, classes, etc.
  - Operators and delimiters: +, -, *, /, %, ;, …
  - Alphabet Σ of the lexical analyzer: characters
- ***Syntactic analyzer (parser) of a compiler*** recognizes syntactic constructs (statements, expressions, variable declarations, etc.)
  - Alphabet Σ of the syntactic analyzer: tokens

Regular Expressions

Regular Expressions

- One way to denote (often infinite) languages
- A regular expression is an expression built from:
  - empty language $\varnothing$
  - $\{\varepsilon\}$, denoted by **ε**
  - $\{a\}$ for a in Σ, denoted simply by ***a***
  - union, denoted **|** (or, sometimes, **+**)
  - concatenation, as multiplication (dot), or omitted
  - Kleene star ***** (repetition)

Example 1

- Names of labs in DD1362:
  - F1, F2, F3, S1, S2, S3, Inet, X1
- We could describe this set of strings with the following regular expression:
  - F1 | F2 | F3 | S1 | S2 | S3 | Inet | X1

Language = subset of Σ*

- Explanation:
  - Regex F stands for language {F} where F in Σ
  - Regex F1 stands for language {F1} where F, 1 in Σ
  - Regex F1 | F2 stands for language {F1, F2} where F, 1, 2 in Σ
  - Etc.

Example 1 Continued

- Names of labs in DD1362:
    - F1, F2, F3, S1, S2, S3, Inet, X1
- The names follow a certain ***pattern:***
    - either it is string Inet, or
    - it starts with F, S, or X followed by 1, or
    - it starts with F or S followed by 2 or 3.
- This pattern can be described using the following regular expression:
    - Inet | (F|S|X)1 | (F|S)(2|3)

Example 2

- All binary strings:
  - “”, “0”, “1”, “00”, “01”, “10”, “000”, “001”, …
- Fundamental difference to previous example?
  - There is an ***unbounded*** number of binary strings!
  - We cannot list them all.
- Solution: make use of repetition operator *: (0|1)*
- Regex a* matches an arbitrary number of occurrences of pattern a (“0 or more times”)

> A regular expression is a ***pattern*** for describing a set of strings

Syntactic Extensions for Regular Expressions that Preserve Definable Languages

- [a-z] = a|b|...|z (use ASCII ordering)
  (also other shorthands for finite languages)
- e? (optional expression)
- e+ (repeat at least once)
- $e^{k..*} = e^k \, e^*$ $e^{p..q} = e^p \, (\varepsilon | e)^{q-p}$
- complement: !e (do not match)
- intersection: e1 & e2 (match both) = ! (!e1|!e2)

Examples of Regular Expressions

- Decimal digits
  - digit ::= 0 | 1 | .. | 8 | 9
- Integer constants
  - intConst ::= digit digit*
- Alphabetic characters
  - letter ::= [a-z] | [A-Z]
- Identifiers
  - ident ::= letter (letter | digit)*

e.g., variable names

Regular Expressions in Practice

- Regular expressions are used for a variety of ***text processing tasks***
  - Syntax highlighting in code editors and IDEs, search-and-replace, …
- Many tools and languages implement regular expression matchers
  - A number of different syntax variations
  - Check documentation for regex syntax of specific tool

Regular Expressions in Unix Tools

- `grep '<regex>' <file>`
- Outputs all lines in `<file>` where some text matching `<regex>` occurs

```
$ grep '..ing' grep_wikipedia.txt
grep is a command-line utility for searching plain-text
has the same effect: doing a global search with the
and printing all matching lines.
```

- `sed 's/<regex>/<replacement>/g' < <file>`
- Replaces all occurrences of text matching `<regex>` by `<replacement>`

```
$ sed 's/Bell/Whistle/g' < grep_wikipedia.txt >
grep_wikipedia_funny.txt
```

Regular Expressions in Java

- Package **`java.util.regex`** contains classes “for matching character sequences against patterns specified by regular expressions.”
  - “An instance of the **`Pattern`** class represents a regular expression that is specified in string form”
  - See JDK API documentation
- Example:

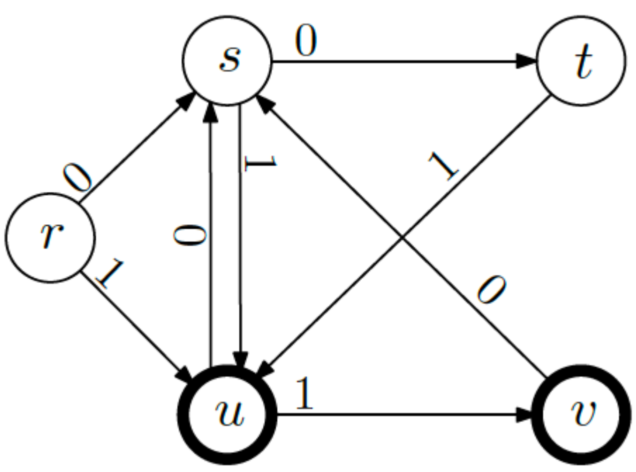
```
import java.util.regex.*;

Pattern p = Pattern.compile("cat");
Matcher m = p.matcher("one cat, two cats in the yard");
String s = m.replaceAll("dog");
// --> s = "one dog, two dogs in the yard"
```

Finite Automata

What is a Finite Automaton?

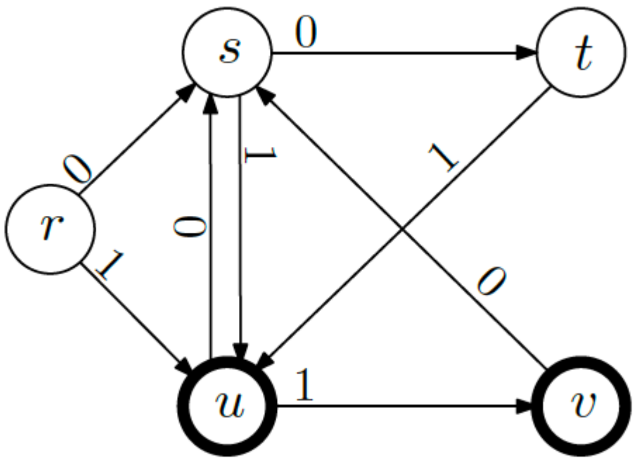
A finite automaton consists of:

- An alphabet Σ
- A finite set of states
- An initial state
- A set of state transitions with labels in Σ
- A set of final states (also "accepting states")



- Start state r
- Final states u and v

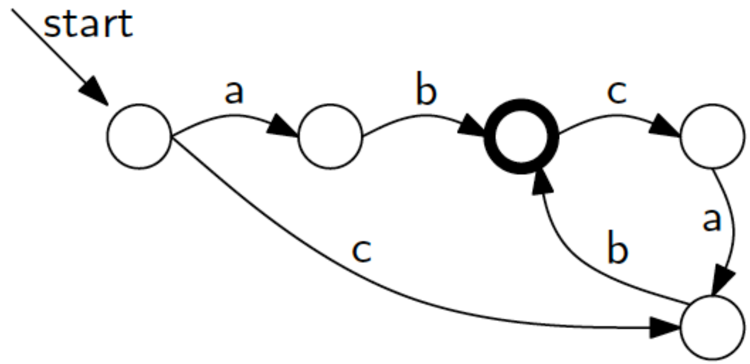
Example 1



- Start state r
- Final states u and v

- Input 1: 01100101 Accepted
- Input 2: 01110101 Not accepted
- Input 3: 01100100 Not accepted

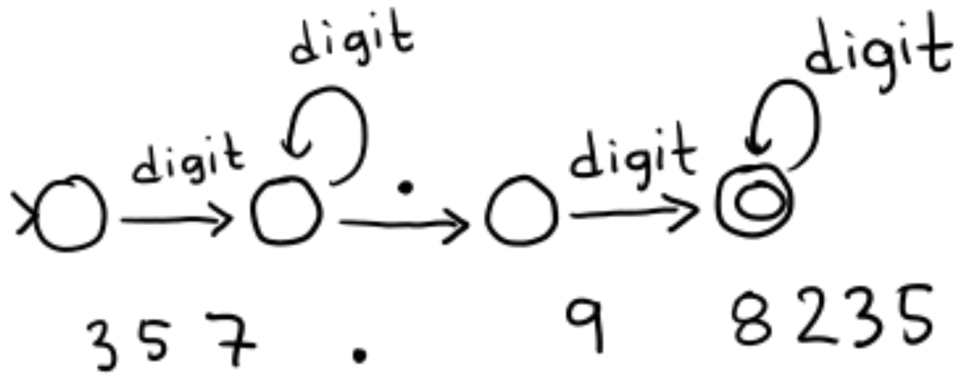
Example 2



- Q: How to find example strings that the automaton accepts?
- A: Follow the arrows to find a path ending with an accepting/final state!
- Accepted strings: ab, cb, cbcab, abcab, ..

Using DFAs to Recognize Languages

DFA for recognizing valid floating-point numbers?



Corresponding regular expression?

digit digit* . digit digit*

Exercise: what if the decimal part is optional?