Introduction to Logic

Algorithm Design & Software Engineering
October 26, 2016
Stefan Feuerriegel

Today's Lecture

Objectives

1. Learn about set operators
2. Understand the specification of first-order logic
3. Design and implement regular impressions in R

Outline

Outline

Sets

- A set is a collection of different values or variables

$$\{1,2,4\} \quad \text{or} \quad \{a,b,\ldots,z\}$$

  where elements are placed in curly parenthesis
- Rule: only distinct elements matter, e. g. $\{1,1,2\} = \{2,1\}$
- Membership is indicated via "$\in$"

$$a \in \{a,b,\ldots,z\} \quad \text{and} \quad 3 \notin \{a,b,\ldots,z\}$$

- Set-builder notation with a "where" clause

$$\left\{ n^2 \mid n \text{ is an integer} \right\}$$

- Common notation to refer to special sets, e. g.
  - $\mathbb{R}$ gives all real numbers
  - $\mathbb{N}$ denotes all positive integers (including zero)

Cardinality and Subsets

Cardinality

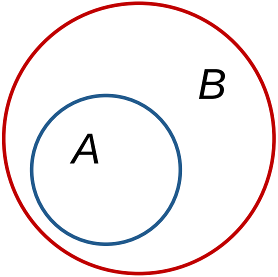
- Cardinality counts the number of elements

$$|\{a, b, \ldots, z\}| = 26$$

- Sets can have an infinite cardinality, e. g. $|\mathbb{N}| = \infty$
- Empty set is given by $\emptyset$ with $|\emptyset| = 0$

Subsets

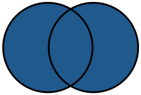
- A is a subset of B if every element of A is also in B, written $A \subseteq B$
- $A \subseteq B$ can also imply $A = B$
- $A \subset B$ if one element $x \in B$ fulfills $x \notin A$
- For all sets X, $\emptyset \subseteq X$



Set Operators

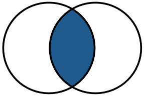
- Given sets $A = \{1, 2, 3\}$ and $B = \{3, 4\}$ as examples
- Visualizations are called Venn diagrams

Union $X \cup Y$



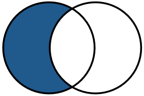
- Set of all elements which are either in X **or** Y
- $A \cup B = \{1, 2, 3, 4\}$

Intersection $X \cap Y$



- Set of all elements which are in both X **and** Y
- $A \cap B = \{3\}$
- If $X \cap Y = \emptyset$, then X and Y are called disjoint

Complement $X \setminus Y$



- Set of all elements of X which are **not in** Y
- $A \setminus B = \{1, 2\}$

Set Operators

Exercises

Given $A = \{1,2,3,5,8\}$, $B = \{0,2,5,7,1,9\}$ and $C = \{-1,4,8,10\}$

1. Find $A \setminus B$, $B \setminus A$, $(A \cup B) \cap C$ and $(A \cap B) \cap C$
   - $A \setminus B = \{3,8\}$
   - $B \setminus A = \{0,7,9\}$
   - $(A \cup B) \cap C = \{0,1,2,3,5,7,8,9\} \cap \{-1,4,8,10\} = \{8\}$
   - $(A \cap B) \cap C = \{1,2,5\} \cap \{-1,4,8,10\} = \emptyset$
2. Show that $A \setminus (A \setminus B) = A \cap B$ holds for all sets A and B
   - Proof

$$
\begin{aligned}
A \setminus (A \setminus B) &= A \setminus \{a \in A \mid a \notin B\} \\
&= \{a \in A \mid a \notin \{a \in A \mid a \notin B\}\} \\
&= \{a \in A \mid a \in B\} \\
&= A \cap B
\end{aligned}
$$

Cartesian Product

An n-tuple is an ordered list $(x_1, \ldots, x_n)$ of n elements; e. g. $(1, 2) \neq (2, 1)$

Cartesian product

- Mathematical operator "$\times$" creates a new set by building tuples, i. e.

$$A \times B := \{ (a, b) \mid a \in A \text{ and } b \in B \}$$

- Special rule $M \times \emptyset = \emptyset$ for all sets M

Examples

- Given sets $A = \{ 1, 2 \}$ and $B = \{ i, j, k \}$, then

$$A \times B = \{ (1, i), (1, j), (1, k), (2, i), (2, j), (2, k) \}$$

- The Cartesian product of sets $A_1, \ldots, A_n$ is

$$A_1 \times A_2 \times \cdots \times A_n = \{ (a_1, a_2, \ldots, a_n) \mid a_1 \in A_1, a_2 \in A_2, \ldots, a_n \in A_n \}$$

Cartesian Product

Exercise: show for all finite sets A and B that $|A \times B| = |A| \cdot |B|$ holds

Proof: let $|A| = m$, we then prove by induction over $|B| = n$

Base case with $n = 0$:

let $B = \emptyset$, then $|A \times B| = |\emptyset| = 0 = m \cdot 0 = |A| \cdot |B|$

Inductive step:

- **Assumption:** $|A \times B| = |A| \times |B|$ holds for all sets B with $|B| = n$
- **Show:** the assumption implies the statement for all B with $|B| = n+1$
  - Let $|B| = n+1$ and thus $|B| \geq 1$
  - B has at least one element and we can choose an arbitrary $b \in B$
  - Now we define $\tilde{B} := B \setminus \{b\}$ and thus $B = \tilde{B} \cup \{b\}$

$$\begin{aligned}
|A \times B| &= |A \times (\tilde{B} \cup \{b\})| = |(A \times \tilde{B}) \cup (A \times \{b\})| \\
&= |A \times \tilde{B}| + |A \times \{b\}| \\
&= |A \times \tilde{B}| + m \overset{\text{with assumption}}{=} nm + m \\
&= m(n+1) = |A| \cdot |B|
\end{aligned}$$

Power Set

- Sets can be elements of sets
- Example: $\{a,b\} \in \{\{a,c\},\{a,b\}\}$, but $\{a,b\} \notin \{a,b,\{a,b,c\}\}$

Power set

- Power set $\mathscr{P}(A)$ of set A is the set of all subset of A, i. e.

$$\mathscr{P}(A) = \{B \mid B \subseteq A\}$$

- Cardinality $|\mathscr{P}(A)| = 2^n$ with $n = |A|$

Example

- Given set $A = \{1,2,3\}$, then

$$\mathscr{P}(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$$

De Morgan's Law

Let X be a set with subsets $A, B \subseteq X$

1. $X \setminus (A \cup B) = (X \setminus A) \cap (X \setminus B)$
2. $X \setminus (A \cap B) = (X \setminus A) \cup (X \setminus B)$

Proof (of the first law)

$$
\begin{aligned}
x \in X \setminus (A \cup B) \Leftrightarrow &\quad x \in X \text{ and } x \notin A \cup B \\
\Leftrightarrow &\quad x \in X \text{ and } x \notin A \text{ and } x \notin B \\
\Leftrightarrow &\quad (x \in X \text{ and } x \notin A) \text{ and } (x \in X \text{ and } x \notin B) \\
\Leftrightarrow &\quad (x \in X \setminus A) \text{ and } (x \in X \setminus B) \\
\Leftrightarrow &\quad x \in (X \setminus A) \cap (X \setminus B)
\end{aligned}
$$

Sets in R

- R has not dedicated data type for sets, instead it uses vectors

```
x <- c(1, 2, 3)
y <- c(3, 4)
```

- `is.element(x, y)` or operator `%in%` tests $x \in y$

```
is.element(3, x)

## [1] TRUE

4 %in% x

## [1] FALSE
```

- Equality is tested via `setequal(x, y)`

```
setequal(x, y)

## [1] FALSE

setequal(c(1), c(1, 1, 1))

## [1] TRUE
```

Set Operators in R

- Set operators come as functions named `union(x, y)`, `intersect(x, y)` and `setdiff(x, y)`

```
union(x, y)

## [1] 1 2 3 4

intersect(x, y)

## [1] 3

setdiff(x, y)

## [1] 1 2

setdiff(y, x)

## [1] 4
```

Outline

Boolean Algebra

- Boolean algebra is a calculus with only two elements: $\{0, 1\}$
- Values can be interpreted as “true”, “false” or “on”, “off”
- Common in computers where “0” refers to 0 volts, “1” to a reference voltage (e. g. 5 V)
- Combinations (i. e. Cartesian products) allow to store more information; e. g. 1011000101

1 bit		=	2 combinations
2 bits		=	4 combinations
...			
8 bits	(or 1 byte)	=	256 combinations
10 bits		=	1024 combinations
...			
32 bits	(or 4 bytes)	=	4 294 967 296 combinations

Boolean Operators

Boolean calculus includes 3 basic operations:

not $\neg$ **and** $\wedge$ **or** $\vee$

Truth table

x	y	$\neg x$	$x \wedge y$	$x \vee y$
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

Examples

Let a be the proposition "the car is red" and b "the car is big"

- **Not:** $\neg a$ means "The car is not red"
- **And:** $a \wedge b$ means "the car is both red and big"
- **Or:** $a \vee b$ means "the car is red or big or both"

Derived Operators

Additional operators can be derived, such as

implication $\Rightarrow$ **XOR** $\oplus$ **equivalence** $\Leftrightarrow$

x	y	$x \Rightarrow y$	$x \oplus y$	$x \Leftrightarrow y$
0	0	1	0	1
0	1	1	1	0
1	0	0	1	0
1	1	1	0	1

Examples

Let *a* denote “grandpa cooks” and *b* “grandma is in a good mood”

- $a \rightarrow b$ means “if Grandpa cooks, then grandma is in a good mood”
- $a \oplus b$ means “either Grandpa cooks or grandma is in a good mood”
- $a \leftrightarrow b$ means “grandpa cooks if and only if grandma is in a good mood”

Implication Operator

- Attention is needed as implication does not imply causality
- Rewriting is possible, i. e. $x \Rightarrow y = (\neg x) \vee y$
- The statement $x \Rightarrow y$ always true if x is false

$$
\begin{aligned}
x \Rightarrow y &= (\neg x) \vee y \\
&= (\neg 0) \vee y \\
&= 1 \vee y = 1
\end{aligned}
$$

Example

- Let $x =$ "all cats bark" and $y =$ "all cats are green"
- Then the implication $x \Rightarrow y$ is always true:
  "if all cats bark, then all cats are green"
- Cats commonly don't bark, so the content of proposition y can successfully be derived from it

Laws in Boolean Algebra

	$\wedge$	$\vee$
Commutativity	$x \wedge y = y \wedge x$	$x \vee y = y \vee x$
Associativity	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$	$x \vee (y \wedge z) = (x \vee y) \vee z$
Identity	$x \wedge 1 = x$	$x \vee 0 = x$
Annihilator	$x \wedge 0 = 0$	$x \vee 1 = 1$
Idempotence	$x \wedge x = x$	$x \vee x = x$
Complementation	$x \wedge (\neg x) = 0$	$x \vee (\neg x) = 1$

Distributivity	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
Absorbtion	$x \wedge (x \vee y) = x$
	$x \vee (x \wedge y) = x$
Duality	$\neg 0 = 1$
Double negation	$\neg(\neg x) = x$
De Morgan's law	$\neg(x \vee y) = (\neg x) \wedge (\neg y)$
	$(\neg x) \vee (\neg y) = \neg(x \wedge y)$

Normal Forms

Every finite logical formula can be reduced to two normal forms:

1. Conjunctive normal form (CNF): $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} \underbrace{\neg}_{\text{optional}} x_{ij}$
2. Disjunctive normal form (DNF): $\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} \underbrace{\neg}_{\text{optional}} x_{ij}$

Example

Let f be a logical expression as follows:

x	0	0	0	0	1	1	1	1
y	0	0	1	1	0	1	0	1
z	0	1	0	1	0	0	1	1
f(x,y,z)	0	0	0	1	1	1	1	1

CNF: $(x \vee y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z)$

DNF: $(\neg x \wedge y \wedge z) \vee (x \wedge \neg y \wedge \neg z) \vee (x \wedge y \wedge \neg z) \vee (x \wedge \neg y \wedge z) \vee (x \wedge y \wedge z)$

Outline

Relation

- A n-dimensional relation $R \subseteq A_1 \times \cdots \times A_n$ is a set of ordered n-tuples
- Relations can be interpreted as functions by defining

$$R(x_1, \ldots, x_n) := \begin{cases} 1, & \text{if } (x_1, \ldots, x_n) \in R, \\ 0 & \text{otherwise} \end{cases}$$

Example:

- Let A be the set of all animals, C be the set of all colors
- Let R be the relation "animal a has color c" $\subset A \times B$
- Then (frog, green) $\in A \times B$ and it is also an element of R
- Thus, $R(\text{frog}, \text{green}) = 1$ (representing true)

Quantifiers

Quantifiers express propositions about quantities (in the context of relations)

1. **Existential quantifier** $\exists$
   - Interpreted as "there exists at least one"
2. **Universal quantifier** $\forall$
   - Interpreted as "for all holds"

Examples

1. $\exists n \in \mathbb{N} : n > 5$
   - Read: there exists at least one natural number n such that n is greater than 5
   - Expression is true, since e. g. $6 \in \mathbb{N}$ and $6 > 5$
2. $\forall p \in \mathbb{P} : p > 2$
   - Read: for all prime numbers p the value of p is greater than or equal to 2
   - Expression is not true, since 2 is also a prime number

Laws in First-Order Logic

Given sets A and B and relations $R, S \subseteq A$, as well as $X \subseteq A \times B$

$$\neg\forall a \in A : R(a) \Leftrightarrow \exists a \in A : \neg R(a)$$

$$\exists a \in A \exists b \in B : T(a,b) \Leftrightarrow \exists b \in B \exists a \in A : T(a,b)$$

$$(\forall a \in A : R(a)) \wedge (\forall a' \in A : S(a')) \Leftrightarrow \forall a \in A : R(a) \wedge S(a)$$

Caution: existential and universal quantifiers are not commutative

$$\forall a \in A \quad \exists b \in B \quad T(a,b) \quad \not\Leftrightarrow \quad \exists b \in B \quad \forall a \in A \quad T(a,b)$$

- $A =$ set of all keys, $B =$ set of all locks
- $T(a,b) =$ key a fits into lock b
- Statements are not identical:
  1. For all locks exists a key that fits into
  2. There exists a key that fits into all locks

Quantifiers

Examples

List all elements of $\{\, n \in \mathbb{N} \mid \exists a \in \mathbb{N} a < 20 \wedge a = n^2 \,\}$

- Remember $0 \notin \mathbb{N}$
- $1^2 = 1 < 20$, $2^2 = 4 < 20$, $3^2 = 9 < 20$, $4^2 = 16 < 20$, $5^2 = 25 > 20$ and all other squares are even larger
- Solution is $\{\, 1, 2, 3, 4 \,\}$

List all elements in $\{\, n \in \mathbb{N} \mid \forall a \in \mathbb{N} \text{ with } a < n : a \text{ is prime } \vee a = 1 \,\}$

- $n = 1$ has no smaller natural numbers a
- $n = 2, 3, 4$: smaller number are elements of the set $\{\, 1, 2, 3 \,\}$ of which all elements are prime
- For $n = 5$, there is $a = 4 = 2 \cdot 2$ which is not prime
- For all $n > 4$, there is also $a = 4$ which is not prime
- Solution is $\{\, 1, 2, 3, 4 \,\}$

Outline

Pattern Matching

- A regular expression defines a search pattern for pattern matching
- Useful when searching for strings with placeholders or wildcards
  $\rightarrow$ one uses meta-characters with specific means for that purpose
- Patterns specify characters, repetitions and locations within the string
- Common use cases are finding a certain string, replacing it or extracting information
- Syntax varies slightly across programming languages

Example

- Locate all elements which contain a pattern (here: foo)

```
grep("foo", c("arm", "food"))

## [1] 2
```

Pattern Matching in R

- `grep(pattern, x)` searches a pattern in `x`
- It returns all indices of the vector which match the pattern

```
txt <- c("a", "ab", "acb", "accb", "acccb", "bacccc")
grep("b", txt)

## [1] 2 3 4 5 6
```

- Argument `value=TRUE` returns the matching values

```
grep("b", txt, value=TRUE)

## [1] "ab"     "acb"    "accb"   "acccb"  "bacccc"
```

- Alternatively, `grepl` returns Boolean values if an element matches

```
grepl("b", txt)

## [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

- Search is case-sensitive by default (off:`ignore.case=TRUE`)

```
grepl("B", txt, ignore.case=TRUE)

## [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

Pattern Matching in R

- `grepexpr(pattern, string)` returns the position of a match in a string

```r
# b appears as the second character
gregexpr("b", "abc")
```

```
## [[1]]
## [1] 2
## attr(,"match.length")
## [1] 1
## attr(,"useBytes")
## [1] TRUE
```

```r
gregexpr("b", "abc")[[1]][1]
```

```
## [1] 2
```

- Returns -1 if not found

```r
gregexpr("d", "abc")[[1]][1]
```

```
## [1] -1
```

Regular Expressions in R

Location meta-characters

- ^ matches the starting position within a string

```
txt
## [1] "a"      "ab"     "acb"    "accb"   "acccb"  "bacccc"
grep("^b", txt, value=TRUE)
## [1] "bacccc"
```

- $ matches the ending position of a string

```
grep("b$", txt, value=TRUE)
## [1] "ab"    "acb"   "accb"  "acccb"
```

Regular Expressions in R

Special characters

- `\n` denotes a new line
- Quotation marks must be escaped via the backslash

```r
x <- "\"string\""
```

Boolean OR

- A vertical bar `|` distinguishes alternatives

```r
grep("gray|grey", c("gray", "grey", "different"), value=TRUE)

## [1] "gray" "grey"
```

Grouping

- Parentheses group logical units

```r
grep("gr(a|e)y", c("gray", "grey", "different"), value=TRUE)

## [1] "gray" "grey"
```

Regular Expressions in R

Quantifiers

- A dot `.` matches any character

```r
grep(".", c("a", "b", "c", "\n"), value=TRUE)

## [1] "a"  "b"  "c"  "\n"
```

- A question mark `?` denotes zero or one occurrences of the preceding literal
  → i. e. makes the previous character optional

```r
grep("colou?r", c("color", "colour"), value=TRUE)

## [1] "color"  "colour"

grep("ab?a", c("a", "aa", "aba", "abba"), value=TRUE)

## [1] "aa"  "aba"

# can be used together with grouping
grep("a(xxx)?a", c("a", "aa", "axxxa", "axxxxa"), value=TRUE)

## [1] "aa"    "axxxa"
```

Regular Expressions in R

Quantifiers

- A plus + indicates one or more occurrences

```r
grep("a+", c("", "b", "a", "aa", "aaab"), value=TRUE)

## [1] "a"    "aa"   "aaab"
```

- An asterisk * indicates zero or more occurrences

```r
grep("a*", c("", "b", "a", "aa", "aaab"), value=TRUE)

## [1] ""     "b"    "a"    "aa"   "aaab"

grep("xa*y", c("", "a", "xy", "xay", "xaay"), value=TRUE)

## [1] "xy"   "xay"  "xaay"
```

- Alternatively, specify a fixed number of occurrences or a range

```r
grep("x{2}", c("x", "xx", "xxx", "xxxx"), value=TRUE)

## [1] "xx"   "xxx"  "xxxx"

grep("x{1,3}", c("x", "xx", "xxx", "xxxx"), value=TRUE)

## [1] "x"    "xx"   "xxx"  "xxxx"
```

Regular Expressions in R

Symbol classes

- Bundle a set of different characters inside `[` and `]` for ease-of-use

```
grep("analy[sz]e", c("analyse", "analyze"), value=TRUE)

## [1] "analyse" "analyze"
```

- Digits via `[[:digit:]]` or `\\d` or `[0-9]`

```
grep("[[:digit:]] euro", c("3 euro", "33 euro",
                              "three euro"),
      value=TRUE)

## [1] "3 euro"  "33 euro"
```

- Lower-case letters via `[[:lower:]]` or `[a-z]`

```
grep("[[:lower:]]", c("", "a", "z", "A"), value=TRUE)

## [1] "a" "z"
```

- Both letters and digits via `\\w` or `[A-z0-9\_]`

Regular Expressions in R

Symbol classes

- Any space character (tabulator, new line, space, etc.) via `[[:space:]]`

```
grep("[[:space:]]", c("", ".", "!", "x", " ", "\n"),
     value=TRUE)

## [1] " "  "\n"
```

- Any punctuation via `[[:punct:]]`

```
grep("[[:punct:]]", c("", ".", "!", "x", " ", "\n"),
     value=TRUE)

## [1] "." "!"
```

Regular Expressions in R

Examples

```
cars <- rownames(mtcars)
grep("*er", cars)

##  [1]  7  8  9 10 11 12 13 14 17 22 29 30 31

grep("*er", cars, value=TRUE)

##  [1] "Duster 360"        "Merc 240D"          "Merc 230"
##  [4] "Merc 280"          "Merc 280C"          "Merc 450SE"
##  [7] "Merc 450SL"        "Merc 450SLC"        "Chrysler Imperia
## [10] "Dodge Challenger"  "Ford Pantera L"     "Ferrari Dino"
## [13] "Maserati Bora"

grep("er+a", cars, value=TRUE)

## [1] "Ford Pantera L" "Ferrari Dino"   "Maserati Bora"
```

Regular Expressions in R

Examples

```
grep("d+er", cars, value=TRUE)

## character(0)

grep("*er{2}", cars, value=TRUE)

## [1] "Ferrari Dino"

grep("^F", cars, value=TRUE)

## [1] "Fiat 128"       "Fiat X1-9"      "Ford Pantera L" "Ferrar

grep("(t|g)er", cars, value=TRUE)

## [1] "Duster 360"       "Dodge Challenger" "Ford Pantera L"
```

Regular Expressions in R

Examples

```
grep("\\dS", cars, value=TRUE)

## [1] "Merc 450SE"  "Merc 450SL"  "Merc 450SLC"

grep("D([a-z]*)", cars, value=TRUE)

## [1] "Datsun 710"        "Hornet 4 Drive"   "Duster 360"
## [4] "Merc 240D"         "Dodge Challenger" "Ferrari Dino"

grep("^d([a-z]*)", cars, value=TRUE, ignore.case=TRUE)

## [1] "Datsun 710"        "Duster 360"       "Dodge Challenger"
```

Replacements and Extraction

Replacements

- `gsub(pattern, replacement, x)` replaces patterns in `x`

```r
gsub("x", "a", "abcxyz")
## [1] "abcayz"
gsub("colou?r", "red", "Please write in colour")
## [1] "Please write in red"
```

Extraction

- Load package `gsubfn`

```r
library(gsubfn)
```

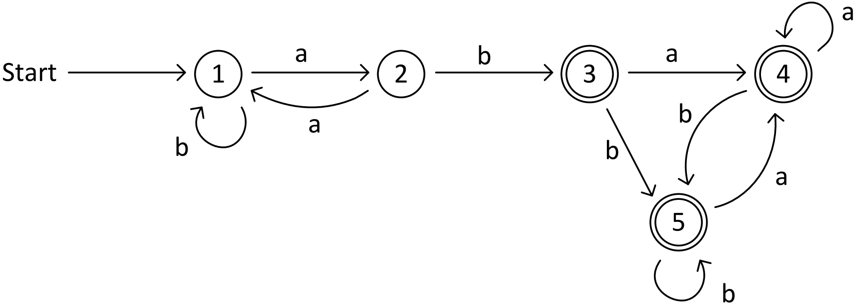
- `strapply(x, pattern)` extracts part in parentheses from any match

```r
strapply("3 euro", "([[:digit:]]) euro")[[1]]
## [1] "3"
```

Representation as Automaton

- Regular expression can be visualized as a finite automaton
- Arrows indicate allowed expressions
- Terminal states have two circles

Example: `ab(a|b)+`



Outline

Wrap-Up

1. Sets
   - Operations: union, intersection, complement, cartesian product, power set, cardinality
   - De Morgan's law
2. Boolean algebra
   - Elements 0 and 1 with basic operators: $\land$, $\lor$ and $\neg$
   - Derived operators: $\Rightarrow$, $\oplus$ and $\Leftrightarrow$
   - Conjunctive and disjunctive normal forms
3. First-order logic
   - Relations
   - Quantifiers: $\exists$ and $\forall$
4. Regular expressions
   - Search patterns for string matching, extraction of sub-strings and replacements
   - Include meta-characters, symbol classes, ORs and quantifiers
   - Regular expressions be rewritten as automatons
   - R: `grep(...)`, `grepl(...)`, `grepexpr(...)` and `gsub(...)`