2. Formal Languages

2.1 Introduction

2.1.1 Truth Tables & Operator Precedence

It's sometimes easier to add intermediate steps, so we can e.g. add $\neg$ A and $\neg$ B as columns:

A	B	C	$\neg$ A	$\neg$ B	$\neg$ A $\vee$ $\neg$ B $\vee$ C
T	T	T	F	F	T
T	T	F	F	F	F
T	F	T	F	T	T
T	F	F	F	T	T
F	T	T	T	F	T
F	T	F	T	F	T
F	F	T	T	T	T
F	F	F	T	T	T

And of course we can also add intermediate expressions as columns:

A	B	C	D	B ∨ ¬ C	¬ (B ∨ ¬ C)	A ∧ ¬ (B ∨ ¬ C) ∨ D	¬ (A ∧ ¬ (B ∨ ¬ C) ∨ D)
T	T	T	T	T	F	T	F
T	T	T	F	T	F	F	T
T	T	F	T	T	F	T	F
T	T	F	F	T	F	F	T
T	F	T	T	F	T	T	F
T	F	T	F	F	T	T	F
T	F	F	T	T	F	T	F
T	F	F	F	T	F	F	T
F	T	T	T	T	F	T	F
F	T	T	F	T	F	F	T
F	T	F	T	T	F	T	F
F	T	F	F	T	F	F	T
F	F	T	T	F	T	T	F
F	F	T	F	F	T	F	T
F	F	F	T	T	F	T	F
F	F	F	F	T	F	F	T

2.1.2 Morgan's Law

$$
\begin{aligned}
& \neg(A \wedge \neg(B \vee \neg C) \vee D) \\
\rightarrow\ & \neg(A \wedge \neg B \wedge C \vee D) \\
\rightarrow\ & (\neg A \vee B \vee \neg C) \wedge \neg D
\end{aligned}
$$

$$
\begin{aligned}
& \neg(\neg(\neg A \wedge B) \wedge \neg(C \vee \neg D)) \\
\rightarrow\ & \neg((A \vee \neg B) \wedge (\neg C \wedge D)) \\
\rightarrow\ & \neg(A \vee \neg B) \vee \neg(\neg C \wedge D) \\
\rightarrow\ & \neg A \wedge B \vee C \vee \neg D
\end{aligned}
$$

$$
\begin{aligned}
& A \vee (C \wedge \neg(B \vee C)) \\
\rightarrow\ & A \vee (C \wedge \neg B \wedge \neg C) \\
\rightarrow\ & A \vee C \wedge \neg B \wedge \neg C
\end{aligned}
$$

Note that it doesn't always make an expression more readable if you leave away parentheses - even if you can. At least for me, the intermediate step in the last exercise is more quickly "parsed" than the last one. Particularly, it's easier to note that the expression in the parenthesis always evaluates to `false`, so the entire expression could be further simplified to `A`.

2.1.3 Simplifying Expressions

$$\begin{aligned}
& A \wedge (\neg A \wedge B) \\
\rightarrow\ & A \wedge \neg A \wedge B \\
\rightarrow\ & (A \wedge \neg A) \wedge B \\
\rightarrow\ & false \wedge B \\
\rightarrow\ & false
\end{aligned}$$

$$\begin{aligned}
& A \wedge (\neg A \vee B) \\
\rightarrow\ & (A \wedge \neg A) \vee (A \wedge B) \\
\rightarrow\ & false \vee (A \wedge B) \\
\rightarrow\ & A \wedge B
\end{aligned}$$

$$\begin{aligned}
& (A \wedge \neg B) \vee (\neg A \wedge B) \\
\rightarrow\ & (A \vee (\neg A \wedge B)) \wedge (\neg B \vee (\neg A \wedge B)) \\
\rightarrow\ & ((A \vee \neg A) \wedge (A \vee B)) \wedge ((\neg B \vee \neg A) \wedge (\neg B \vee B)) \\
\rightarrow\ & (true \wedge (A \vee B)) \wedge ((\neg B \vee \neg A) \wedge (true)) \\
\rightarrow\ & (A \vee B) \wedge (\neg B \vee \neg A) \\
\rightarrow\ & A \veebar B
\end{aligned}$$

$$\begin{aligned}
& (A \vee \neg B) \wedge (\neg A \vee B) \\
\rightarrow\ & (A \wedge (\neg A \vee B) \vee (\neg B \wedge (\neg A \vee B)) \\
\rightarrow\ & ((A \wedge \neg A) \vee (A \wedge B)) \vee ((\neg B \wedge \neg A) \vee (\neg B \wedge B)) \\
\rightarrow\ & (false \vee (A \wedge B)) \vee ((\neg B \wedge \neg A) \vee false) \\
\rightarrow\ & (A \wedge B) \vee (\neg B \wedge \neg A) \\
\rightarrow\ & A \Leftrightarrow B
\end{aligned}$$

You may have come to the conclusion that this is $\neg(A \veebar B)$, or `XNOR` - that's also correct.

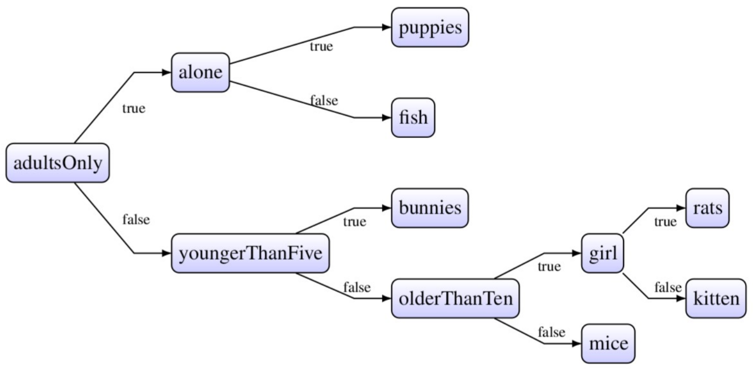
$$\begin{aligned}
& \neg(A \wedge \neg B) \vee A \wedge \neg B \\
\rightarrow\ & (\neg A \vee B) \vee (A \wedge \neg B) \\
\rightarrow\ & \neg A \vee B \vee (A \wedge \neg B) \\
\rightarrow\ & (\neg A \vee B \vee A) \wedge (\neg A \vee B \vee \neg B) \\
\rightarrow\ & (\neg A \vee A \vee B) \wedge (\neg A \vee B \vee \neg B) \\
\rightarrow\ & (true \vee B) \wedge (\neg A \vee true) \\
\rightarrow\ & true \wedge true \\
\rightarrow\ & true
\end{aligned}$$

2.2 Control Structures

2.2.1 Conditions

Pseudocode

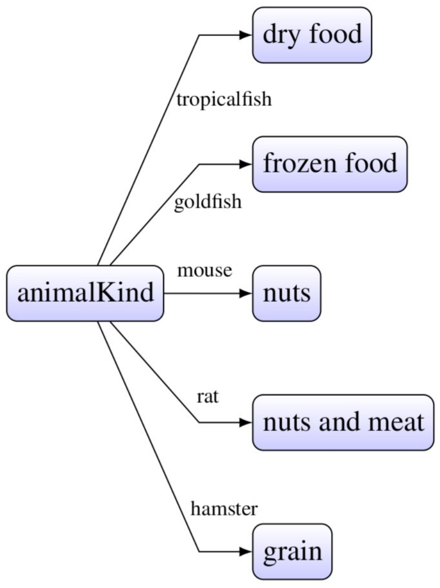
```
if adultsOnly then
  if alone then
    // puppies
  else
    // fish
  end if
else
  if youngerThanFive then
    // bunnies
  else
    if olderthanTen then
      if girl then
        // rats
      else
        // kitten
      end if
    else
      // mice
    end if
  end if
endif
```



2.2.2 Case Matching

Pseudocode

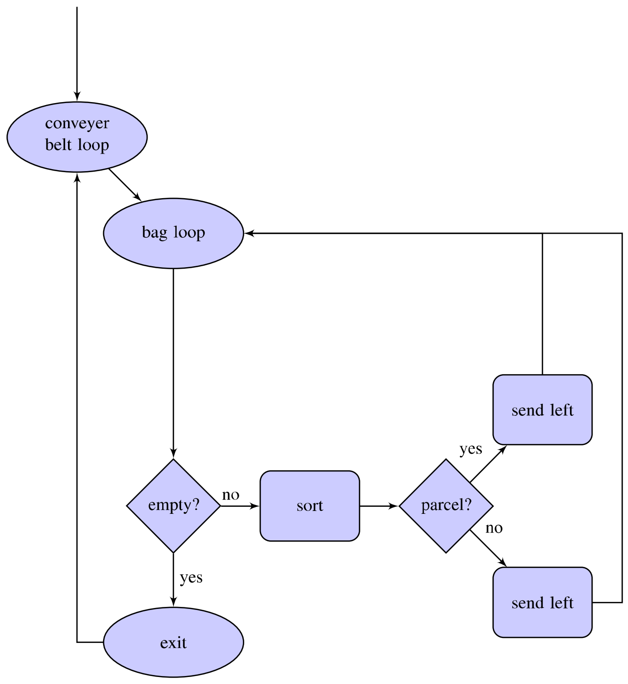
```
if animalKind
  matches tropicalfish then // dry food
  matches goldfish then     // frozen food
  matches mouse then        // nuts
  matches rat then          // nuts and meat
  matches hamster then      // grain
endif
```



2.2.3 Loops

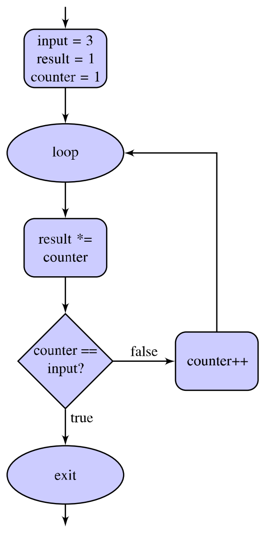
Pseudocode

```
loop // over bags
  loop // over content
    exit if // bag empty
    if parcel then
      // send left
    else
      // send right
    end if
  end loop
end loop
```



2.3 Data Structures

2.3.1 Constants and Variables



Pseudocode

```
constant input = 3

variable result = 1
variable counter = 2

loop
  exit if counter > input
  result = result * counter
  counter = counter + 1
end loop
```

Step	Line	counter	result	In Loop	counter > input
1	3	n/a	n/a	F	F
2	4	n/a	1	F	F
3	6	2	1	F	F
4	7	2	1	T	F
5	8	2	1	T	F
6	9	2	2	T	F
7	7	3	2	T	F
8	8	3	2	T	F
9	9	3	6	T	F
10	7	4	6	T	T
11	10	4	6	F	T

2.3.2 Arrays

Pseudocode

```
constant input = [2, 4, 9, 1, 7]

variable result = 0
variable counter = 0

loop
  exit if counter == input.length
  result = result + input[counter]
  counter++
end loop
```

2.3.3 Maps

Pseudocode

```
constant shapeCorners = { "Triangle": 3, "Rectangle": 4, "Pentagon": 5, "Hexagon": 6,
    "Heptagon": 7, "Octagon": 8 }
constant input = ["Triangle", "Hexagon", "Octagon"]

variable result = 0
variable counter = 0

loop
  exit if counter == input.length
  result = result + shapeCorners[input[counter]]
  counter++
end loop
```

2.4 Program Structures

2.4.1 Methods

Pseudocode

```
method addShapeCorners = (input) => {
  constant shapeCorners = { "Triangle": 3, "Rectangle": 4, "Pentagon": 5, "Hexagon": 6,
    "Heptagon": 7, "Octagon": 8 }

  variable result = 0
  variable counter = 0

  loop
    exit if counter == input.length
    result = result + shapeCorners[input[counter]]
    counter++
  end loop

  return result
}
```

It's a function - it returns a value. Call it using e.g.
`addShapeCorners(["Triangle", "Hexagon", "Octagon"])`

`calculateFactorial(addShapeCorners(["Triangle", "Rectangle"]));`

2.5 Reading & Printing

Pseudocode

```
method calculateFactorialInteractive = () => {
  print("Please enter a number: ")
  constant input = read();
  print("The factorial of " + input + " is " + calculateFactorial(input));
}
```

3. Regular Expressions

3.1 Matching

3.1.1 Literal matching

You could have searched for anything ;-) Did you encounter anything that wasn't covered later?

3.1.2 Alternatives

`/if|then|else/`

You may have noticed that you found matches you didn't expect, such as e.g. "different", which contains the pattern `if`.

3.1.3 Word Boundaries

- Trivially, we just add word boundaries to each `/\bif\b|\bthen\b|\belse\b/`.
- A shorter version could e.g. be `/\b(if|then|else)\b/`

3.1.4 Whitespace

Any "if" at the beginning or end of the text in which we're looking for it wouldn't be matched by `/\sif\s/`. On the other hand, `/\bif\b/` still matches.

3.1.5 Character Classes

- `/[nwkl]it/`
- `/[Aa]loo[fF]/`

3.1.6 Quantifiers

- `?`: `{0,1}`
- `+`: `{1,}`
- `*`: `{0,}`

3.1.7 Wildcard

`/\b[0-9]+\h.+\b/`

3.1.8 Negation

- `/\b[^0-9]*\b/`
- `/\b[^a-z]*\b/`

3.1.9 Escaping Characterss

- `true \/ false`
- `/price \[EUR\]/`

3.1.10 Groups

- `/(?:[a-zA-Z]:|\\)(?:\\[^\\\/:*?"<>|\v]+)*\\?/`
- `/Knock,\hknock.\nWho's\hthere\?\n([a-zA-Z\s]*).\n\1\hwho\?\n\1\h.*\!/`
- `/Knock,\hknock.\nWho's\hthere\?\n(?'name'[a-zA-Z\s]*).\n\g{name}\hwho\?\n\g{name}\h.*\!/`

3.1.11 Anchors and Multi-Line Matching

- `/^\h+|\h+$/gm`
- `/^ +$/gm`

3.1.12 Lookahead / Lookbehind

- before the word: `(?=\w)(?<!\w)`
- after the word: `(?<=\w)(?!\w)`

To match either, we can use `(?<!\w)(?=\w)|(?<=\w)(?!\w)`

3.1.13 Regex Delimiters

`~^/(?:[^/]+/)*[^/]*$~`

3.2 Replacing

3.2.1 Referencing Groups

- `/^\s*if\s+(.+)\s+then\s+(.+)\s+else\s+(.+)\s+end if\s+$/if $1 then\n\t$2\nelse\n\t$3\nend if/`
- `/^(?:https?:\/\/)?(.*)$/<a href="https://$1">$1</a>/mg`

3.2.2 Case Conversion

`/\b(if|then|else|end\hif)\b/\U$1/gi`