MOSS09 foundations
Mathematical concepts, Computational Complexity, Logic

Thomas Fogwill tom.fogwill@gmail.com
Ronell Alberts ralberts@csir.co.za

Masters Ontology Spring School, 2009
Presented: 7-8 September 2009
## Contents

### I Mathematical background

1. Introduction 4  
2. Sets 4  
   2.1 Basics 4  
   2.2 Properties 5  
   2.3 Operations 6  
3. Relations and functions 8  
   3.1 Relations 8  
   3.2 Properties 8  
   3.3 Functions 9  
4. Graphs 9  
   4.1 Definitions 9  
   4.2 Problems 11  

### II Computability and Complexity 12  
1. Overview 12  
2. Computability 12  
   2.1 Introduction 12  
   2.2 Formal languages 13  
   2.3 Turing machines 15  
   2.4 Computability/decidability 16  
3. Computational complexity 17  
   3.1 Concepts 17  
      3.1.1 Cost of computation 17  
      3.1.2 Best vs worst case 18  
      3.1.3 Rate of growth 19  
   3.2 Algorithm vs problem complexity 21  
   3.3 Complexity classes 23  
   3.4 $\mathcal{NP}$, $\mathcal{NP}$-Hard, $\mathcal{NP}$-Complete 25  
4. References and further reading 30  

### III Propositional Logic

1. Introduction 31  
   1.1 What is Logic? 31  
   1.2 Propositional Logic - what does it mean? 31
Part I
Mathematical background

Required mathematical background

1 Introduction

Introduction
This course begins with a (very) brief review of a few simple mathematical concepts that are used later. This includes

- sets
- relations and functions
- graphs

2 Sets

2.1 Basics

Basics

- A set is a mathematical object that allows us to talk about groups of objects i.e. a set contains a collection of other objects (e.g. numbers)
  - For example, we can talk about \( \mathbb{N} \), the set of natural numbers
  - We can represent sets visually using Venn diagrams...
  - Example: suppose we have three individuals objects, \( A \), \( B \) and \( C \). We represent this set visually as follows:

Notation
There are two common notations used in text for representing sets

- Explicit: \( S = \{ A, B, C \} \) - every element of the set \( S \) is listed explicitly
  - Implicit: \( S = \{ x \mid x \geq 0 \} \) - elements are not listed explicitly, but the members of the set are defined in terms of some condition. For this example, the definition means that \( S \) contains all \( x \) such that \( x \geq 0 \)
Special sets

• The empty set is denoted \(\emptyset\) - it contains no elements (i.e. \(\emptyset = \{\}\))

• The universe (often denoted \(U\)) is the set of all possible objects of the type that we want to consider. The precise definition depends on the problem. For example, if we’re working with the natural numbers, then \(U\) could be the set of all integers (or even of all numbers).

2.2 Properties of sets

Membership

• If some object \(A\) is in a set \(S\), we say that \(A\) is a member or an element of \(S\)

• Notation: \(A \in S\)

• Example: for the set \(S = \{A, B, C\}\) (represented as a Venn diagram below), we have that
  - \(A \in S\)
  - \(B \in S\)
  - \(C \in S\)

Equality

• If we have two sets \(S\) and \(T\) such that they have exactly the same members, then we say the sets are equal

• Notation: \(S = T\)

• Example: the sets \(S = \{A, B, C\}\) and \(T = \{C, B, A\}\) are equal (note: order is irrelevant in sets)
Containment

- Assume we have two sets, $S$ and $T$, such that every member of $S$ is also a member of $T$
- then we say that $S$ is contained in $T$, or $S$ is a subset of $T$, and write $S \subseteq T$
- If, in addition, $T$ has elements that are not members of $S$, then we say $S$ is a proper subset of $T$, and write $S \subset T$
- Note: if $S \subseteq T$ and $T \subseteq S$, then $T = S$

2.3 Set operations

Union
In the following slides, we quickly review the basic set operations, including
- Union ($\cup$)
- Intersection ($\cap$)
- Difference ($\setminus$)
- Complement ($^c$)

Union
- The result of the union of two sets $S$ and $T$ is another set that contains all the elements of $S$ and $T$.
- $A \in S \cup T$ iff $A \in S$ or $A \in T$
- Example: If $S = \{A, B, C\}$ and $T = \{C, D, E\}$, then $S \cup T = \{A, B, C, D, E\}$
### Intersection

- The result of the intersection of two sets $S$ and $T$ is another set that contains only those elements that are elements of both $S$ and $T$.
- $A \in S \cap T$ iff $A \in S$ and $A \in T$
- Example: If $S = \{A, B, C\}$ and $T = \{C, D, E\}$, then $S \cap T = \{C\}$

![Intersection Diagram]

### Difference

- The result of the difference between set $S$ and set $T$ is another set that contains only those elements that are elements of $S$ but not of $T$.
- $A \in S \setminus T$ iff $A \in S$ and $A \notin T$
- Example: If $S = \{A, B, C\}$ and $T = \{C, D, E\}$, then $S \setminus T = \{A, B\}$

![Difference Diagram]

### Complement

- The complement of a set $S$ is the set that contains everything outside of $S$
- $A \in S^c$ iff $A \notin S$ iff $A \in U \setminus S$
- Example: If $S = \{A, B, C\}$ and $U = \{A, B, C, D, E, ..., Z\}$, then $S^c = \{D, E, F, ..., Z\}$

![Complement Diagram]
3 Relations and functions

3.1 Relations

Relations

- Given two sets $S$ and $T$, we may want to express some sort of relationship between their elements
- This is what a relation does - it relates the elements of a set with those of another (note: in general, relations can span any number of sets, not only 2)
- Relations are usually expressed as a set of tuples, each of which is of the form $(x, y, z)$ where $x \in$ the first set, $y \in$ the second set, etc.
- Example: $R = \{A, B, C\}, S = \{G, H, I\}, T = \{M, N, P\}$
  
  Then $(A, H, P)$ is one such tuple, and
  
  $\{(A, G, M), (A, H, P), (C, H, N)\}$ is a relation on $R, S$ and $T$
- The Cartesian product of two sets $S$ and $T$ is the set of all possible tuples constructed by taking as first argument an element of $S$, and as second component an element from $T$
- It is written as $S \times T$
- Example: $S = \{A, B\}, T = \{C, D\}$
  
  $S \times T = \{(A, C), (A, D), (B, C), (B, D)\}$
- Relations are clearly a subset of the cartesian products of the all the sets they range over
- Example: $R = \{A, B, C\}, S = \{G, H, I\}, T = \{M, N, P\}$
  
  $\{(A, G, M), (A, H, P), (C, H, N)\} \subseteq R \times S \times T$ is a relation on $R, S$ and $T$
- Usually, we consider n-ary relations over a single set $S$
  
  e.g. a binary relation over $S \times S$ ($S \times S$ is written $S^2$)
  
  e.g. a ternary relation over $S \times S \times S$ (written $S^3$)
  
  For a binary relation $R$, we often write $xRy$ for convenience to indicate that $(x, y) \in R$

3.2 Properties of relations

Properties of relations

- A binary relation $R$ over $S^2$ is reflexive if $(x, x) \in R$ for all $x \in S$
- A binary relation $R$ over $S^2$ is transitive if whenever $(x, y) \in R$ and $(y, z) \in R$, then also $(x, z) \in R$
- A binary relation $R$ over $S^2$ is symmetric if whenever $(x, y) \in R$ then also $(y, x) \in R$
3.3 Functions
Definition
- A function is a special type of relation between a set of elements called the domain and another set of elements called the range.
- It associates each element in the domain with exactly one element in the range - i.e. each element of the domain can only appear once as first coordinate in the relation.

4 Graphs
4.1 Definitions
Definitions
- A graph is a mathematical representation in which pairs of objects can be connected by links.
- The objects themselves are called vertices, and the links are called edges.
- Graphs have natural graphical representations - e.g.

![Graph Example](image)

- The notation we use for graphs is an ordered pair \((V, E)\), where \(V\) is the set of vertices, and \(E \subseteq V^2\) is the set of edges.
- If \((a, b) \in E\), it means there is an edge from \(a\) to \(b\) in the graph.
- Example: \(G = (V, E)\) with \(V = \{A, B, C, D, E\}\) and \(E = \{(A, B), (A, D), (B, C), (B, D), (D, A), (E, D)\}\).
Directed vs undirected graphs

- The examples of graphs we’ve seen so far were all directed graphs.
- In a directed graph, the order of vertices in each edge $e \in E$ matter - the edges “point” from the first coordinate to the second, and can only be traversed in that direction.
- By contrast, in an undirected graph, order does not matter - edges can be traversed in either direction (forwards or backwards).
- Any undirected graph $G = (V, E)$ can easily be converted into an equivalent directed graph $G' = (V, E')$ - for each edge $(a, b) \in E$, create edges $(a, b)$ and $(b, a)$ in $E'$ (skipping duplicates).

![An undirected graph](image)

![Corresponding directed graph](image)

Cycles

- Given a graph $G = (V, E)$, a path from vertex $a$ to vertex $b$ is a sequence of edges $e_1 = (a, v_1), e_2 = (v_1, v_2), e_3 = (v_2, v_3), \ldots, e_n = (v_{n-1}, b)$ such that each $e_i \in E$. The path has length $n$.
- If there is any vertex $v \in V$ such that there is a path of length $> 0$ from $v$ to itself, that path is called a cycle, and the graph is said to be cyclic.
4.2 Well-known graph problems

Some well-known graph problems

- The reachability problem simply asks: Can I reach vertex \( b \) from vertex \( a \) by following (directed) edges?

- The \textit{k-colourability problem} simply asks: Can I colour each vertex of the graph with a colour, using at most \( k \) colours, such that no pair of connected vertices have the same colour?

- A \textbf{Hamiltonian circuit} is a cycle that "visits" every vertex in the graph at least once.

- The \textbf{Hamiltonian circuit problem} simply asks: Does this graph have a Hamiltonian circuit?
Part II
Computability and Complexity

Introduction to Computability and Computational Complexity

1 Overview

Overview
In this part of the course, we examine two key questions in Computer Science

• There are (infinitely) many problems we can think of. Can they all be solved by a computer program
  or are the some that are not solvable on any computer, no matter how powerful?
• Assuming a problem is solvable, can a computer solve it in reasonable amount of time or what computing resources are required to solve it?

• The first is the question of computability
• The second is that of complexity

2 Computability

2.1 Introduction

Introduction

• In this section, we examine the question: Is a problem \( P \) solvable by a machine (computer)?
  If it is, we say the problem is computable

• We consider a problem to be solvable if we can come up with a method (algorithm) for solving it. The algorithm must
  – be correct: it must always give us the correct answer
  – be effective: it must be guaranteed to terminate. In other words, it must always return and give us an answer

• To determine whether a problem is computable, we check whether it is solvable on an abstract machine

• This abstract machine is a mathematical/theoretical construct that models computation, in general

• Thus, if the problem is solvable on this abstract machine, it is computable in general
Decision problems

- We consider here only decision problems.
- A decision problem is one that has a yes/no answer
- Examples:
  1. Give as input integer X, is X prime?
  2. Give as input a list Y, is Y sorted?
- All problems have a corresponding decision problem
- Example:
  
  Problem: Given a graph as input, what is the shortest path from A to B?
  Decision problem: Given a graph as input, is there a path of length n from A to B?

2.2 Formal languages

Formal languages

- A language can be seen as the (possibly infinite) set of strings (e.g. sentences) that are part of that language
- When we talk about a formal language, we’re talking at a syntactic level. We’re concerned with what strings in the language ”look like”: the types of strings (and their structure) that are accepted as part of the language
- So, a formal language is concerned with the form (i.e. shape/structure) of strings
- A formal language is defined by
  - An alphabet of symbols: these are used to build up the words/sentences of the language
  - A strict means of specifying what strings form part of the language

Example

- Consider the language $L = \{ab, abab, ababab, \ldots\}$
  
  Alphabet: \{a, b\}
  
  Specification: every string starting with ab, followed by zero or more occurrences of ab, is in $L$
Language recognisers/generators

- There are many ways of giving the "specification" of the language. Some of these describe how to generate strings in the language, while others say how to recognize them.

- A language generator can usually be used as a recogniser, and vice versa.

- Some of the well-known ways of specifying languages include:
  - Regular expressions
  - Finite state automata
  - Context free grammar (set of rules)
  - Turing machine (see later)

Example

- Consider again the language \( \mathcal{L} = \{ab, abab, ababab, \ldots\} \)

- Specified with a regular expression: \( ab(ab)^* \)

- An FSA

```
  a -- b
  \arrow{a} \arrow{b}
```

- A CFG
  
  \[
  S \rightarrow ab \\
  S \rightarrow abS 
  \]

Hierarchy of formal languages

- Different machines and formalisms have different power when it comes to language recognition/generation.

- The languages can be classified into a hierarchy (i.e. the Chomsky hierarchy):
  
  e.g. regular expressions and FSAs recognise the same class of languages: **regular languages**

  e.g. CFGs recognise/generate the class of **context-free languages**

  e.g. Linear bounded non-deterministic Turing machines (Turing machines with a tape of length \( kn \) for input size \( n \)) recognise the class of **context sensitive languages**
e.g. Deciders (halting Turing machines) recognise the class of \textbf{recursive languages}.

e.g. Arbitrary Turing machines recognise the class of \textbf{recursively-enumerable languages}.

![Diagram showing the relationship between different classes of languages: Type-0, recursively-enumerable (TM); Type-1, context-sensitive (linear bounded TM); Type-2, context-free (CFG); Type-3, regular (regular languages, FSA).]

\textbf{Note}: Each class is a strict subset of the one above it (i.e. Type-3 \textsubscript{⊂} Type-2 \textsubscript{⊂} Type-1 \textsubscript{⊂} Type-0).

\textbf{Decision problems as recognition}

- So, why do we care about languages - we’re supposed to be talking about computability.

- Well, the fact is that we can encode problems in such a way that they are essentially (string) recognition problems.

- i.e. encode the decision problem as a machine,

- and encode the input as a string that we feed into the machine.

- the machine accepts all strings corresponding to input on which the decision procedure answers \textbf{yes}, and rejects all other (answer \textbf{no}).

- Clearly, not all machines are equal in terms of the types of problems they can solve - i.e. a Turing machine can solve problems that an FSA cannot.

- We use the Turing machine as our abstract machine to model computation.

\subsection*{2.3 Turing machines}

\textbf{Turing machines}:

A Turing machine consists of

- An infinite \textbf{tape} that is divided into \textbf{cells}. The tape is arbitrarily extendable to the left and right.

- A \textbf{head} that can read or write one cell at a time. The head can be moved left or right.
A set of possible states, as well as a register storing the current state. One of the states is marked as the start state.

A table of transitions/actions that specifies, for the current machine state and the current symbol on the tape (under the head), whether to

- erase or write a symbol to the tape cell under the head
- move the head left or right
- stay in the current state or transition to a new state

A Turing machine starts in its start state, and begins processing the input (stored in the cells of the tape) as per the instructions in the transition/action table

If it reaches a state and/or reads a symbol for which there is no corresponding entry in the transition/action table, it halts

If/when it halts, the output appears on the tape

2.4 Computability/decidability

Computability/decidability

If a decision problem is computable, we say it is decidable

Given that we’ve chosen the Turing machine as our abstract model of computation, we now define computability as follows:

A problem $P$ is computable (or decidable) if

- We can build a Turing machine that,
- for any input,
- always halts
- and outputs the correct answer.

i.e. a problem is decidable if it can be answered by a Decider - a halting Turing machine. In other words, if it it corresponds to a recursive language.

Halting problem

Now that we have a definition of computability/decidability, it is natural to ask: are there any problems that are NOT computable?

The answer is yes - in fact, one of the first problems to be proven non-computable (in the 1930s) was the Halting problem

The Halting problem asks:

- Can we decide whether a Turing machine halts for given input?
Semi-decidable

- There are problems for which a Turing machine can be built that always halts and answers correctly for positive input (i.e. for when the answer is yes).
- But that do not necessarily halt when the answer should be no.
- These problems are called semi-decidable, and correspond to the class of recursively-enumerable languages.

3 Computational complexity

3.1 Concepts

3.1.1 Cost of computation

Complexity - intuitive idea

With computational complexity, we consider:
- The "cost" of computing the solution to some problem.
- How quickly that "cost" grows with larger input.
- What "cost" (and rate of growth) is acceptable.
- What factor contributes most to the "cost".
- What to do if the cost is too high.

Computer resources

There are finite resources available for computation (even in the cloud):
- CPU speed.
- Memory (e.g. RAM).
- Storage (disk space).
- Time (how long we're willing to wait for the answer).

Therefore, there is always some problem that is too "costly" to solve.

Cost of computation

We usually consider the "cost" of computation as:
- The amount of time required (TIME complexity).
- The amount of memory required (SPACE complexity).

... in the worst case ...
3.1.2 Best vs worst case

Worst vs best case
We can distinguish complexity i.t.o.

worst-case the “cost” of the computation, assuming the worst possible instance of the problem

best-case the “cost” in the best case

average-case the “cost”, assuming an average (or typical) instance of the problem

Worst vs best case - by example
Consider the following algorithm, which scans a list for a specific item

FindItemInList(List list, Item item)

Input: A List list and an Item item

Output: The position of item in list, or -1 if item does not occur in the list

begin
  pos = 0;
  foreach Item i in List list do
    pos++;
    if i = item then
      return pos;
  end

end

• We consider the cost of executing this algorithm on different input - i.e. on different instances of the problem.

• To simplify the analysis, we focus on the most expensive operations of the algorithm

• ... and on the operations that are representative of the “work” done by the algorithm

• ... in this case, the number of comparisons within the loop

• In general
  – graph algorithms - edge traversals
  – sorting algorithms - comparisons
  – numeric algorithms - arithmetic operations (esp. floating point)

• Each of the following is a different instance of the problem (represented by different input List list of length n and Item item)

  worst-case item does not appear in the list at all, then we need to perform n comparisons before returning −1

  almost-best-case item appears first in the list, then we need only a single comparison before returning 1
**best-case** list is the empty list, and we immediately return $-1$ after 0 comparisons

- What about the average case?

**Average case**

- Average case analysis is difficult, because we need to know the likelihood of a case occurring - i.e. probability distribution.

- However, in practise it is sometimes very useful to perform such analysis - there are problems that have prohibitive worst-case complexity, but perform well on average.

**This course**

In this course, we consider only worst-case complexity:

- guarantees termination within number of steps for all instances

- i.e. gives us an upper bound

### 3.1.3 Rate of growth

**Rate of growth**

- We think of the complexity of an algorithm as though it were a function (in the size of the input)

<table>
<thead>
<tr>
<th>n</th>
<th>1</th>
<th>10</th>
<th>100</th>
<th>1000</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$5n + 800$</td>
<td>805</td>
<td>850</td>
<td>1300</td>
<td>5800</td>
<td>50,800</td>
</tr>
<tr>
<td>$n^2$</td>
<td>1</td>
<td>100</td>
<td>10,000</td>
<td>1,000,000</td>
<td>1E8</td>
</tr>
<tr>
<td>$n^3$</td>
<td>1</td>
<td>1000</td>
<td>1,000,000</td>
<td>1E9</td>
<td>1E12</td>
</tr>
<tr>
<td>$2^n$</td>
<td>2</td>
<td>1024</td>
<td>1.267E30</td>
<td>1.071E301</td>
<td>1.995E3012</td>
</tr>
</tbody>
</table>

- We say an algorithm’s complexity function is _order_ $g$ if it’s rate of growth is no more than that of $g$

- $g$ then serves as an approximation (upper bound) of the algorithm’s rate of growth for large input

- written as $O(g)$

**Asymptotic growth rates**

Formally, we say that a function $f = O(g)$ if,

$$ (x \to \infty) \exists C, k . \forall x > k . |f(x)| < C \times g(x) $$

This means

- as $x$ tends to $\infty$ (i.e. $x$ is large)

- there is some $k$
• and some constant $C$ such that
• for all $x > k$
• $f(x) < C \times g(x)$

We call this the **asymptotic** upper bound for $f$ - it means $f$ grows no faster than $g$.

### Asymptotic growth rates

Analogously,

$f = \Omega(g)$ means that $f$ grows at least as fast as $g$

i.e. $g$ is a lower bound for $f$.

### Asymptotic growth rates - dominant term

• Typically when talking about complexity, we ignore the slower growing parts of the function, and talk only about the fastest growing term.

• This is because the fastest growing term dominates for large input.

• So

  - $4n^2 + 17n + 9 = \mathcal{O}(n^2)$
  - $n^4 + 2n^2 + 78n = \mathcal{O}(n^4)$
  - $20n + 1709 = \mathcal{O}(n)$
  - $143 = \mathcal{O}(1)$
  - $2^n + 12n^9 = \mathcal{O}(2^n)$

### Common growth rates

<table>
<thead>
<tr>
<th>$\mathcal{O}(1)$</th>
<th>constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{O}(\log n)$</td>
<td>logarithmic</td>
</tr>
<tr>
<td>$\mathcal{O}(n)$</td>
<td>linear</td>
</tr>
<tr>
<td>$\mathcal{O}(n \log n)$</td>
<td>$n \log n$</td>
</tr>
<tr>
<td>$\mathcal{O}(n^2)$</td>
<td>quadratic (polynomial)</td>
</tr>
<tr>
<td>$\mathcal{O}(n^3)$</td>
<td>cubic (polynomial)</td>
</tr>
<tr>
<td>$\mathcal{O}(n^k)$</td>
<td>polynomial</td>
</tr>
<tr>
<td>$\mathcal{O}(a^n)$</td>
<td>exponential</td>
</tr>
<tr>
<td>$\mathcal{O}(n!)$</td>
<td>factorial</td>
</tr>
</tbody>
</table>

Anything greater than exponential is called hyper-exponential (or just exponential).

20
3.2 Algorithm vs problem complexity

Complexity of algorithm vs complexity of problem

Algorithm complexity states the number of operations an algorithm (i.e. program) must execute (in worst case) to solve a problem.

Problem complexity states the underlying theoretical complexity of the problem itself - the minimum number of operations any algorithm would require to solve the problem (i.e. the problem cannot be solved in fewer operations).

The complexity of the problem represents a lower bound on the complexity algorithms that solve it - i.e. if a problem is known to be $\Omega(n^2)$, then there can exist no algorithm to solve the problem that is $O(n)$

Complexity of algorithm

Consider the problem of sorting a list (containing $n$ items). A (very inefficient) algorithm to do so is:

```plaintext
TerribleSort(List list)
begin
    foreach Item item in list do
        tmpList = list; // make temporary copy of list
        tmpList.remove(item); // remove item from tmplist
        sortedTmpList = TerribleSort(tmpList); // sort the rest
        if item > sortedTmpList[0] then
            outputList = []; // create a new empty list
            outputList.append(item);
            outputList.append(sortedTmpList);
        return sortedTmpList;
    end
end
```

Complexity of algorithm

- This algorithm, in the worst case, needs $n$ iterations to find the largest item.
- For each of those $n$ iterations, it makes a recursive call to sort the remaining $n-1$ items.
- Sorting the $n-1$ items requires:
  - $n-1$ iterations to find the (next) largest item
  - a recursive call for each to sort the remaining $n-2$ items
  - etc.
- Thus, in total, this algorithm requires $n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1 = n!$ iterations.
- Thus, this algorithm $= O(n!)$
Complexity of algorithm

- So, by the existence of this algorithm, we know that \( n! \) is an upper bound on the complexity of the problem of sorting a list
- i.e. sorting a list is \( \mathcal{O}(n!) \)
- BUT, can we do better?
- YES!
  - there are far better sorting algorithms
  - e.g. BubbleSort, InsertionSort are \( \mathcal{O}(n^2) \)
  - e.g. Heapsort, Quicksort are \( \mathcal{O}(n \log n) \)
- So, we’ve improved out upper bound to \( \mathcal{O}(n \log n) \). Can we do even better than that?
- NO! (see next slide)

Complexity of problem

- What is the inherent complexity of sorting a list? \( \Omega(n \log n) \)
- Each of the \( n \) list items must be handled, so \( \Omega(n) \)
- ... but that’s a little too coarse.
- Each of the \( n \) list items must also be inserted in the correct position in the sorted list
- What is the complexity of finding the correct position in the sorted list? \( \Omega(\log n) \)
  - Binary Search is \( \mathcal{O}(\log_2 n) \)
  - The basic idea here is as follows:
    * Intuitively, with binary search, we select the middle element of the list. If it is equal to the object we’re searching for, we return its position. If it is bigger, we “discard” all the items to its right, if it is smaller we “discard” all items to its left.
    * So, on each iteration, we essentially halve the length of the list left to search
    * So, if the list was initially \( n \) items long, we need at most about \( \log_2 n \) iterations
      - Assume \( n = 2^k \) (or \( 2^k + 1 \))
      - Then \( k = \log_2 n \)
      - and, \( n \) can halved \( k (= \log_2 n) \) times before we get 1
      - Thus, binary search is \( \mathcal{O}(\log_2 n) \)
    * This can also be seen by inspecting the search tree generated by binary search
      - The root of the tree is the middle item in the list
The left subtree contains all items < the root, and is rooted by the middle element in that list.

The right subtree contains all items < the root, and is rooted by the middle element in that list.

etc. (recursively)

Obs The tree has \( n \) nodes (one for each item in the list)

Obs The tree has a branching factor of 2 (it branches into at most 2 sub-branches at each node)

Obs The tree is balanced

Obs Searching for an item amounts to traversing the tree from the root downwards until the item is found (or a leaf is encountered). The longest such search is thus the depth of the tree, namely \( k = \log_2 n \)

\[ \therefore \text{binary search is } O(\log_2 n) \]

- So, QuickSort and MergeSort are pretty close to optimal

**Complexity: no known “good” algorithm**

What if there isn’t a known “good” algorithm?

\[ \quad \quad \quad \quad \quad \text{--- some algorithm } O(n^2) \]

\[ \quad \quad \quad \quad \quad \text{--- better algorithm } O(n^4) \]

\[ \quad \quad \quad \quad \quad \text{--- best known algorithm } O(n^2) \]

\[ \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{possible improvement} \]

\[ \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{ complexity of problem } O(n \log_2 n) \]

### 3.3 Complexity classes

Complexity classes
PTime

- *PTime* (usually known simply as *P*) is the class of all problems that
  - can be solved by a deterministic Turing machine
  - in an amount of time that is polynomially bounded
  - i.e. problems in *P* are $O(n^k)$

- *P* contains
  - linear, quadratic and cubic problems
  - $nLOGTime$, $LOGSpace$, $LOGTime$

- Typical problems in *P* include:
  - Graph reachability, 2-SAT, Matrix multiplication

**PTime example: graph reachability**

Given a directed graph $G = (V, E)$ and two vertices $a, b \in V$, determine whether there is a path from $a$ to $b$ in $G$.

**reachable** $(G = (V, E), a, b)$

```
begin
    mark vertex $a$;
    initialise $S = \{a\}$;
    while $S$ is not empty do
        choose vertex $t$ from $S$ and remove it from $S$;
        foreach $(w, u) \in E$ such that $w = t$ and $u$ is unmarked do
            mark $u$ and add it to $S$;
        end
    end
    if $b$ is marked then return True;
    else return False;
end
```
We consider the number of edges \((n = |E|)\) to be size of the input.

- **Complexity analysis: time**
  - Then, there are most \(n = |E|\) reachable nodes from \(a\), so the outer loop iterates at most \(n\) times
  - The inner loop must scan \(E\), so it requires \(n = |E|\) iterations
  - \(\therefore \mathcal{O}(n^2)\) in the size of \(E\)

- **Complexity analysis: space** We only need to store
  - which nodes are marked (these are \(\leq\) the number of nodes reachable from \(a\), so \(\leq n\))
  - the set \(S\), which is always a subset of the marked nodes, so \(\leq n\)

  \(\therefore \mathcal{O}(n)\) in the size of \(E\)

### 3.4 \(\mathcal{NP}\), \(\mathcal{NP}\)-Hard, \(\mathcal{NP}\)-Complete

**The class \(\mathcal{NP}\)**

- There are many problems for which we have no known polynomial algorithm for computing the answer
- But, if we were given an answer by some oracle, that answer could be checked in polynomial time

**Example: 3-colourability**

There is no known polynomial algorithm for 3-colouring a graph \(G = (V, E)\) - i.e. using only three colours, to "paint" each vertex \(v \in V\) such that no two adjacent vertices have the same colour.

However, if we can somehow "guess" a colouring, then that colouring can be checked in polynomial time - all that is required is to check that each vertex is assigned exactly one colour, and that no pair of adjacent vertices have the same colour (this requires \(|V| + |E|\) operations). \(\therefore\) checking the colouring can be done in polynomial time.

- "Guess and test a solution" instead of "search for a solution"
- Non-deterministic - we don’t specify how to guess a solution
- Doesn’t give a better way of finding answers
  - trying all possible answers is typically exponential, or worse
- but it does give us a way of specifying an important class of problems, namely \(\mathcal{NP}\)

**The class \(\mathcal{NP}\)**

A problem \(q\) is in \(\mathcal{NP}\) if there exists a algorithm that

- given any instance \(I\) of the problem \(q\)
- and a candidate solution \(k\),

checks in polynomial time whether \(k\) is a solution for \(I\).
\( \mathcal{P} \subseteq \mathcal{NP} \)

- Clearly, \( \mathcal{P} \subseteq \mathcal{NP} \)
  - for any \( q \in \mathcal{P} \), we can compute the answer in polynomial time
  - thus, checking a given solution \( k \) can be done in polynomial time (by computing and then comparing)

- But it is not known whether \( \mathcal{NP} \subseteq \mathcal{P} \) (and thus whether \( \mathcal{P} = \mathcal{NP} \))
  - most people suspect \( \mathcal{NP} \not\subseteq \mathcal{P} \) (and thus \( \mathcal{P} \neq \mathcal{NP} \))
  - i.e. that \( \mathcal{NP} \) is more difficult than \( \mathcal{P} \)
  - and that checking an answer is easier than finding it

**Reducibility**

- Assume that we have a problem \( A \), but we have no idea how to solve it (i.e. we know of no algorithm to solve instances of problem \( A \))

- However, after some careful reflection, we realise that we can take any instance \( I \) of problem \( A \), and transform it into \( J \), which is an instance of a different problem \( B \), such that
  - we already know how to solve instances of problem \( B \) (we have an algorithm)
  - whatever answer we get for \( J \) using this algorithm represents (and can be transformed back into) a correct answer for \( I \)

- Then we have a way of solving any instance of problem \( A \)
  - transform it into an instance of problem \( B \)
  - compute the answer
  - convert it back into the corresponding answer for \( A \)

- We say that problem \( A \) has been **reduced** (or is **reducible**) to problem \( B \)

**\( \mathcal{NP} \)-Hard**

- We now introduce the important class of \( \mathcal{NP} \)-Hard problems, the class of problems that are at least as difficult as the hardest problems in \( \mathcal{NP} \)

**The class \( \mathcal{NP} \)-Hard**

A problem \( q \) is in \( \mathcal{NP} \)-Hard if **every** problem in \( \mathcal{NP} \) is reducible to \( q \) in polynomial time.

**NOTE** Every problem in \( \mathcal{NP} \) must be reducible to \( q \)

**NOTE** The reduction of each problem may only take polynomial time

- All \( \mathcal{NP} \)-Hard problems are at least as difficult as \( \mathcal{NP} \)
• If a polynomial algorithm is found for any $\mathcal{NP}$-Hard problem, then every problem in $\mathcal{NP}$ can be solved in polynomial time (via reduction)
  
  – i.e. then $\mathcal{P} = \mathcal{NP}$

• Technically, $\mathcal{NP}$-Hard are those problems that have a polynomial-time Turing reduction from every problem in $\mathcal{NP}$

• i.e. polynomial-time reductions that can be executed by a Turing machine

• $\mathcal{NP}$-Hard can also include problems that are not decision problems

• In this course, we consider only decision problems, and do not further discuss Turing reductions

$\mathcal{NP}$-Complete

• The $\mathcal{NP}$-Complete problems are those $\mathcal{NP}$-Hard problems that are also in $\mathcal{NP}$

The class $\mathcal{NP}$-Complete

A problem $q$ is in $\mathcal{NP}$-Complete if

• every problem in $\mathcal{NP}$ is reducible to $q$ in polynomial time

• there is an algorithm that, when given an answer $k$ and an instance $I$ of $q$, checks whether $k$ is a solution to $I$ in polynomial time

• The $\mathcal{NP}$-Complete problems are all reducible to one another, and ”stand and fall together”

  – If any one of them is found to be in $\mathcal{P}$, then all of them are in $\mathcal{P}$
  ... and then also $\mathcal{P} = \mathcal{NP}$

  – If any one of them is found to be definitely NOT in $\mathcal{P}$, then none of them are in $\mathcal{P}$
... and then also $\mathcal{P} \neq \mathcal{NP}$

- As such, they have been heavily studied for almost 30 years
- There are 1000's of known $\mathcal{NP}$-Complete problems, including:
  - SAT (next slide)
  - $k$-SAT (for $k > 2$)
  - Travelling Salesman
  - Graph Colouring (for $> 2$ colours)
  - Hamiltonian Circuit

**Proving** $q \in \mathcal{NP}$-Complete

- To prove that a problem $q$ is in $\mathcal{NP}$-Complete, show that
  1. every problem in $\mathcal{NP}$ is reducible to $q$ in polynomial time
  2. answers to $q$ can be checked in polynomial time
- Part 2 is easy - we prove it by describing a polynomial algorithm that correctly checks any given answer and instance of $q$
- Part 1 is tricky - how do we show that EVERY problem in $\mathcal{NP}$ is reducible to $q$?
  - Firstly, note that for any problem $r \in \mathcal{NP}$-Complete, every problem in $\mathcal{NP}$ is reducible to $r$ in polynomial time
  - Thus, if we can reduce $r$ to $q$ in polynomial time, then we are able (by composition) to also reduce all problems in $\mathcal{NP}$ to $q$
  - So, to prove Part 1 above we reduce (in polynomial time) some known $\mathcal{NP}$-Complete problem (of which there are 1000's) to $q$
- But where did it all start? The first $\mathcal{NP}$-Complete problem was given by Cook in 1971.

**Cook’s Theorem**

Boolean satisfiability of a set of propositional logic clauses (SAT) is $\mathcal{NP}$-Complete

- any problem in $\mathcal{NP}$ can be reduced to SAT in polynomial time
- checking whether an assignment of truth values to propositional letters satisfies every clause in an instance of SAT can be done in polynomial time

Levin discovered the same independently in 1973, so this is also called the Cook-Levin Theorem.
\[ P = NP? \]

IF any one \(NP\)-Hard problem has a polynomial algorithm
THEN... \( P = NP \)

IF any one \(NP\)-Complete problem has a polynomial algorithm
THEN... **EVERY** \(NP\)-Complete problem has a polynomial algorithm
AND... \( P = NP \)

\[ P = NP \]

IF some \(NP\)-Complete problem is proven to \textbf{NOT} have a polynomial algorithm
THEN... **NO** \(NP\)-Complete problem can have a polynomial algorithm
AND... \( P \neq NP \)
4 References and further reading

References and further reading


Part III
Propositional Logic

Propositional Logic

1 Introduction

1.1 What is Logic?

What is Logic?

- Logic could be described as the formal study of inference and reasoning, and is concerned with the structure of statements and proofs.
- Logic provides a way of “standardising” inference and reasoning, thereby allowing for rigorous study of the correctness of arguments and proofs.
- Logic can also be seen as a branch of mathematics, but also (to some extent) as a formalisation of the underlying principles on which mathematical proof is based.

Why do we study it?

We study logic for a number of reasons

- It is often used as a tool to characterise problems in computer science (and in proofs).
- It has numerous applications in computer science, including programming languages, logical circuits, database theory, complexity theory, natural language semantics, etc.
- It forms the basis of most formal systems for knowledge representation and reasoning.
- and more ...

1.2 Propositional Logic - what does it mean?

Informal semantics: propositions

- Propositional Logic (PL) is a logic about propositions.
- A proposition represents some single fact in our “world”.
- e.g. A, B and C are propositions and
  - let A represent the fact “It is raining”
  - let B represent the fact “It is windy”
  - let C represent the fact ”Ronell is happy”
- A proposition is either true or false.
- e.g.
– A is True if it is raining, False if it is not
– B is True if it is windy, False if it is not
– C is True if Ronell is happy. If Ronell is not happy, C is False

Informal semantics: connectives

• We can say more complex things in propositional logic by combining propositions using connectives
• Legal connectives include things like "and", "or", "if ... then", "not"
• e.g.
  – A and B is True if it is both raining and windy, False otherwise
  – A or B is True if it is either raining or windy, or both. If it is neither raining, nor windy, then False
  – not B is True if it is not windy, False if it is
  – if A and B then not C (if it is rainy and windy, then Ronell is not happy) is False only if it is windy and raining and Ronell is happy, True otherwise

Representing knowledge/translating from natural language

• As seen in the examples above, we can use propositional logic to express facts about our "world".
• A few notes, though, about the meaning of the connectives:
  OR in PL is not "exclusive" - i.e. A or B is True if either A is True or B is True or both are True. However, in English OR often is "exclusive" - e.g. "I will go for a swim or I will go for a walk" implies that only one of the two cases can happen
  IF...THEN in PL does not imply causality, although it often does in natural language. Furthermore, if A then B in PL is True if A and B are both True, but also when A is False (regardless of whether B is true).

Ambiguity

• A further difficulty in translating from natural language is ambiguity
• i.e. natural language can be ambiguous
  e.g. I see the man with the telescope
  (1) Do I use a telescope to see the man?
  (2) Or do I see a man who happens to have a telescope?
• PL is not ambiguous - once we’ve decided on the intended meaning of the English sentence, this is expressed unambiguously in propositional logic
• Therefore, the same natural language sentence can have > 1 representation in propositional logic

32
2 Syntax

2.1 Formal language

Formal language

- With "the syntax of propositional logic", we’re talking about what strings of characters (called formulae) are considered legal/sensible in PL

- The syntax of propositional logic can thus be described as a formal language. The formal language $L$ of propositional logic is the set of all legal formulae in PL, and is specified by
  - an alphabet, containing all the individual symbols we can use
  - a grammar, given as a set of recursive rules describing how to build up new formulae in $L$ from smaller ones

... this grammar can also be used to test whether a string is a well-formed formula in $L$

2.2 Grammar: the language of propositional logic

Alphabet

- The language $L$ of PL has the following alphabet:
  - A finite (or countably infinite) set of proposition letters. By convention, we use $A, B, C, ...$
    possibly with subscripts: $A_1, A_2, A_3, ...$
  - The set of connectives $\{\land, \lor, \lnot, \rightarrow, \leftrightarrow\}$
  - The set of grouping symbols $\{(),\}$

- The connectives have names:
  - $\land$ conjunction
  - $\lor$ disjunction
  - $\lnot$ negation
  - $\rightarrow$ implication, or conditional
  - $\leftrightarrow$ bi-implication, biconditional

Grammar

- $L$ is equivalent to the set formulae of propositional logic. This set is the minimal set of strings such that
  - Each proposition letter is a formula
  - If $F$ is a formula, then so is $\lnot F$
  - If $F$ and $G$ are formulae, then so are $F \land G, F \lor G, F \rightarrow G$ and $F \leftrightarrow G$
In other words, \( \mathcal{L} \) is such that
- \( A, B, ... \in \mathcal{L} \), where \( A, B, ... \) are proposition letters
- if \( F \in \mathcal{L} \), then \( \neg F \in \mathcal{L} \)
- if \( F \in \mathcal{L} \) and \( G \in \mathcal{L} \), then
  * \( F \land G \in \mathcal{L} \)
  * \( F \lor G \in \mathcal{L} \)
  * \( F \rightarrow G \in \mathcal{L} \)
  * \( F \leftrightarrow G \in \mathcal{L} \)

Parentheses can be omitted along this order of precedence:

\[ \neg \land \lor \rightarrow \leftrightarrow \]

A propositional letter on its own is called a (propositional) atom

If \( A \) is a propositional atom, then \( A \) and \( \neg A \) are called literals (positive and negative literals, respectively)

3 Semantics

3.1 Introduction

What do we mean by semantics?

- In the previous section, we introduced the syntax of propositional logic - i.e. what propositional logic "looks" like
- However, we’ve not said anything formal about the meaning of propositional logic constructs
- In this section we give the formal semantics of propositional logic - i.e. we associate meaning with propositional logic formulae

3.2 Boolean valued logic

Two-valued logic

- Propositional logic is a two-valued logic - it only allows two possible values, namely
  - True
  - False
**Semantic principles**

- **Principle of bivalence**
  - Propositional logic semantics satisfy the principle of bivalence
  - Basically, this means that every propositional atom (or formula) must be either *True* or *False*
  - For propositional logic, this coincides nicely with the law of excluded middle, given syntactically by $A \lor \neg A$

- **Principle of contradiction**
  - Propositional logic semantics satisfy the principle of contradiction
  - Basically, this means that no pair of contradicting formulae can be simultaneously *True*
  - Which means that no proposition can be simultaneously *true* and *false*
  - For propositional logic, this coincides nicely with the law of contradiction, given syntactically by $\neg (A \land \neg A)$

**3.3 Interpretations**

**Interpretations**

- An interpretation is the way in which we formally assign meaning
- For propositional logic, an interpretation is a total mapping from the set of proposition letters to the set \{*True, False*\}
- i.e. it is a mapping that assigns to each propositional atom either the value *True*, or *False*
- An interpretation can be represented as a set of pairs
  - i.e. $\mathcal{I} = \{(A, \text{True}), (B, \text{False}), (C, \text{True})\}$
- Or, more conveniently, simply as the set of propositional atoms that are True
  - i.e. $\mathcal{I} = \{A, C\}$
- An interpretation assigns meaning to propositional atoms, but what about arbitrary formulae? An interpretation can be extended to arbitrary formulae
- If an interpretation $\mathcal{I}$ makes a formulae $F$ True, we say that $\mathcal{I}$ satisfies $F$ and write $\mathcal{I} \models F$. $\mathcal{I}$ is called a model for $F$
- If $I$ does not make a formulae $F$ True, we write $\mathcal{I} \not\models F$
- Extending an interpretation from the propositional atoms to arbitrary formulae is done (by structural recursion) as follows:
\[ I \models A \quad \text{iff} \quad A \text{ is a proposition letter and } I(A) = \text{True} \]
\[ I \not\models \neg F \quad \text{iff} \quad I \not\models F \]
\[ I \models F \land G \quad \text{iff} \quad I \models F \text{ and } I \models G \]
\[ I \models F \lor G \quad \text{iff} \quad I \models F \text{ or } I \models G, \text{ or both} \]
\[ I \models F \rightarrow G \quad \text{iff} \quad I \not\models F \text{ or } I \models G, \text{ or both} \]
\[ I \models F \leftrightarrow G \quad \text{iff} \quad I \models F \rightarrow G \text{ and } I \models G \rightarrow F \]

- This set of rules gives the formal semantics propositional logic

Example

- Consider the interpretations

\[ I_1 = \{A, B\} \]
\[ I_2 = \{A, C\} \]
\[ I_3 = \{B, C\} \]

and the formulae

\[ F = A \land B \]
\[ G = B \rightarrow A \]
\[ H = \neg C \]

- Then,

\[ I_1 \models F \quad I_1 \not\models G \quad I_1 \not\models H \]
\[ I_2 \not\models F \quad I_2 \models G \quad I_2 \not\models H \]
\[ I_3 \not\models F \quad I_3 \not\models G \quad I_3 \not\models H \]

3.4 Truth tables

Overview

- Another way of representing interpretations is to use truth tables

- A truth table has a column for each propositional atom and a row for each possible combination of truth values

- e.g. for the propositional letters \{A, B, C\}

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

Note: for \( n \) proposition letters, the truth table has \( 2^n \) rows
Connectives

- The semantics of the connectives can be nicely represented in a truth table

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>¬A</th>
<th>A ∧ B</th>
<th>A ∨ B</th>
<th>A → B</th>
<th>A ↔ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>2</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>3</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>4</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

Examples

The truth table for \( F = (C \rightarrow A) \land (B \rightarrow C) \) is

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>C → A</th>
<th>B → C</th>
<th>(C → A) ∧ (B → C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I₁</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>I₂</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>I₃</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>I₄</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>I₅</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>I₆</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>I₇</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>I₈</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

3.5 Propositional Logic Exercises I

1. Develop truth tables for the following:
   (a) \((A \land B) \lor \neg C\)
   (b) \((A \land B) \lor (\neg A \lor \neg B)\)
   (c) \((A \lor B) \lor \neg (A \lor (B \land C))\)

2. Translate the following English sentences into sentences of propositional logic. Use only the propositions we provide.

   For instance, the sentence, "It is either raining or snowing", with the logical propositions:
   - raining = “It is raining”
   - snowing = “It is snowing”

   should be answered by: raining ∨ snowing

   (a) “To become my girlfriend, you must be smart, pretty, and nice”
   - gf = “become my girlfriend”
   - s = “you are smart”
   - p = “you are pretty”
   - n = “you are nice”

   (b) “Unless I go to Korea or Japan, I will not be able to attend a World Cup match”
   - k = “I go to Korea”
3.6 Semantic properties of formulae

Validity, Falsifiability

- Recall: If \( \mathcal{I} \) satisfies \( F \) then \( \mathcal{I} \) is a model for \( F \) and we write \( \mathcal{I} \models F \)
- Recall: For \( n \) propositional letters, we have \( 2^n \) possible interpretations
- \( F \) is valid iff all interpretations satisfy \( F \) (i.e. \( \mathcal{I} \models F \) for all possible interpretations \( \mathcal{I} \))
  \[ F \text{ is called a tautology} \]
  We write this simply as \( \vdash F \)
- \( F \) is falsifiable iff it is not valid - i.e. iff there is at least one interpretation that does not satisfy \( F \) (i.e. \( \mathcal{I} \not\models F \) for some interpretation \( \mathcal{I} \))

Satisfiability

- \( F \) is satisfiable iff there is at least one interpretation that satisfies \( F \) (i.e. \( \mathcal{I} \models F \) for some interpretation \( \mathcal{I} \))
- \( F \) is unsatisfiable iff it is not satisfiable - i.e. iff no interpretation satisfies \( F \) (i.e. \( \mathcal{I} \not\models F \) for all interpretations \( \mathcal{I} \))
- If \( F \) is both satisfiable and falsifiable, then it is contingent

Two special formulae: \( \top \) and \( \bot \)

- There are two special forms of formulae that we often encounter. They’re important enough to be called laws, to have their own names, and to have their own symbols.
- The first is called contradiction, and is any formula of the form \( F \land \neg F \)
  - written as \( \bot \) ("bottom")
  - represents syntactically the notion of a contradiction
  - semantically, by the principle of contradiction, \( \bot \) must always be assigned the value False
  - ... the formal semantics of propositional logic ensure this - i.e.
The second is called excluded middle, and is any formula of the form $F \lor \neg F$

- written as $\top$ ("top")
- represents syntactically the notion that every formula is either True or False
- semantically, by the principle of bivalence, $\top$ must always be assigned the value True
- ... the formal semantics of propositional logic ensure this - i.e.

<table>
<thead>
<tr>
<th>$I_1$</th>
<th>True</th>
<th>False</th>
<th>$F \land \neg F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_2$</td>
<td>False</td>
<td>True</td>
<td>$F \land \neg F$</td>
</tr>
</tbody>
</table>

(as before, the notion of the truth table has been generalised slightly)

Entailment

- We generalise the notions of satisfaction and models to handle sets of formulae
  - Let $\mathcal{F}$ be a set of propositional logic formulae
  - We say an interpretation $\mathcal{I}$ satisfies $\mathcal{F}$ and is a model of $\mathcal{F}$ (written $\mathcal{I} \models \mathcal{F}$) iff $\mathcal{I}$ is a model for every formula $H \in \mathcal{F}$
- A set of formulae $\mathcal{F}$ logically entails a formula $G$ if every model of $\mathcal{F}$ is also a model for $G$.
- We write $\mathcal{F} \models G$
- If $\mathcal{F} \models G$, we also say that $G$ is a logical consequence of $\mathcal{F}$

Example

- We prove that modus ponens holds semantically for propositional logic
  - Modus ponens states that if we know $A \rightarrow B$, and we also know that $A$ holds, then we can infer $B$ (i.e. $\{(A \rightarrow B), A\} \models B$)
  - Let us check this with a truth table

<table>
<thead>
<tr>
<th>$I_1$</th>
<th>True</th>
<th>True</th>
<th>$A \rightarrow B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_2$</td>
<td>True</td>
<td>False</td>
<td>$A \rightarrow B$</td>
</tr>
<tr>
<td>$I_3$</td>
<td>False</td>
<td>True</td>
<td>$A \rightarrow B$</td>
</tr>
<tr>
<td>$I_4$</td>
<td>False</td>
<td>False</td>
<td>$A \rightarrow B$</td>
</tr>
</tbody>
</table>
• The models for $A \rightarrow B$ are $\{I_1, I_3, I_4\}$
• The models for $A$ are $\{I_1, I_2\}$
• So, the only model of $\{A, (A \rightarrow B)\}$ is $I_1$
• But $I_1$ is also a model for $B$, so $\{(A \rightarrow B), A\} \models B$

Deduction
The following theorem is useful when proving entailment

Deduction theorem
Let $F$ be a set of formulae, and $F$ and $G$ be formulae. Then $F \models F \rightarrow G$ iff $F \cup \{F\} \models G$

• Example: in order to prove $F \land G \models F \lor G$, it is sufficient to prove $\models (F \land G) \rightarrow (F \lor G)$

Another useful theorem is this one

Theorem
$F$ is valid iff $\neg F$ is unsatisfiable

• Example (continued from before): in order to prove $\models (F \land G) \rightarrow (F \lor G)$, it is sufficient to prove that $\neg((F \land G) \rightarrow (F \lor G))$ is unsatisfiable

Equivalence

• Two formulae $F$ and $G$ are semantically equivalent
  iff they have the same truth value under all interpretations
  iff $F \models G$ and $G \models F$

• We write $F \equiv G$

Useful (and named) equivalences
$F \land F \equiv F$  idempotency of $\land$

$F \lor F \equiv F$  idempotency of $\lor$

$F \land G \equiv G \land F$  commutativity of $\land$

$F \lor G \equiv G \lor F$  commutativity of $\lor$

$F \land (G \land H) \equiv (F \land G) \land H$  associativity of $\land$

$F \lor (G \lor H) \equiv (F \lor G) \lor H$  associativity of $\lor$

$F \land (G \lor H) \equiv (F \land G) \lor (F \land H)$  distributivity of $\land$

$F \lor (G \land H) \equiv (F \lor G) \land (F \lor H)$  distributivity of $\lor$

$(F \land G) \lor F \equiv F$  absorption

$(F \lor G) \land F \equiv F$  absorption

$\neg \neg F \equiv F$  double negation

$\neg (F \land G) \equiv \neg F \lor \neg G$  De Morgan’s Law

$\neg (F \lor G) \equiv \neg F \land \neg G$  De Morgan’s Law

$F \leftrightarrow G \equiv (F \rightarrow G) \land (G \rightarrow F)$  equivalence

$F \rightarrow G \equiv \neg F \lor G$  material implication

$F \rightarrow G \equiv \neg G \rightarrow \neg F$  contraposition

Example
We prove De Morgan’s Law ($\neg (F \land G) \equiv \neg F \lor \neg G$) using a truth table

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>G</th>
<th>$F \land G$</th>
<th>$\neg F$</th>
<th>$\neg (F \land G)$</th>
<th>$\neg F \lor \neg G$</th>
</tr>
</thead>
<tbody>
<tr>
<td>I₁</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>I₂</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>I₃</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>I₄</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

Substituting equivalent subformulae

- Let $F$ be a formula, and let $G$ be the result of substituting (in $F$) a subformula $H$ with an equivalent formula $H_=$

- Then $F \equiv G$

Example  Let $F$ be $A \land \neg (B \lor C)$

By substituting the subformula $\neg (B \lor C)$ by the equivalent (by De Morgan) $\neg B \land \neg C$, we get $A \land \neg B \land \neg C$.

claim  $A \land \neg (B \lor C) \equiv A \land (\neg B \land \neg C)$

proof  By truth table
A | B | C | ¬B ∧ ¬C | ¬(B ∨ C) | A ∧ ¬(B ∨ C) | A ∧ (¬B ∧ ¬C)
--- | --- | --- | --- | --- | --- | ---
True | True | True | False | False | False | False
True | True | False | False | False | False | False
True | False | True | False | False | False | False
True | False | False | True | True | True | True
False | True | True | False | False | False | False
False | True | False | False | False | False | False
False | False | True | False | False | False | False
False | False | False | True | True | True | True

4 Decidability and complexity

Decidable, NP-Complete

- For propositional logic, we can always find and answer by exhaustively trying all possible interpretations in a truth table
- Therefore, problems in propositional logic are decidable
- However, the exhaustive approach is exponential in complexity
- In fact, Cook showed in 1971 that SAT for propositional logic (the problem of checking whether a set of propositional logic formulae are satisfiable) is NP-Complete

4.1 Exercises II

1. Determine whether each of the following is valid, contingent or unsatisfiable.
   (a) ¬(B ∧ ¬C ∧ ¬B)
   (b) ¬(A ∨ B ∧ ¬C) ∨ (A ∨ B ∧ ¬C)
   (c) ¬(A ∧ (¬A ∨ (B ∧ C))) ∨ B

2. Show by truth tabling that:
   \[ A → B ∧ C \models A → C \]

3. Show that two formulae F and G are semantically equivalent (F ≡ G) iff F ↔ G is a tautology.

4. Let \( \tau \) and \( \Delta \) be sets of sentences in propositional logic, and let \( \gamma \) be an individual sentence in propositional logic. State whether each of the following statements is true or false.
   - If \( \tau \models \gamma \) and \( \Delta \models \gamma \) then \( \tau \cup \Delta \models \gamma \)
   - If \( \tau \models \gamma \) and \( \Delta \models \gamma \) then \( \tau \cap \Delta \models \gamma \)

5. Prove: F is valid iff \( \neg F \) is unsatisfiable

5 References and further reading

References and further reading

42


Part IV
First-Order Logic

1 Introduction
1.1 Weakness of propositional logic

Introduction

• In the previous part, we learnt about propositional logic and how it can be used to formalise natural language

• Propositional logic appears to be quite a powerful logic - it can express many natural language concepts and constructs

• However, there are many things that cannot be adequately expressed in propositional logic, some of which are discussed the following slides

• To address some of these deficiencies, we extend the logic - this leads us to the first-order logic - a more powerful logic - which we study in subsequent sections.

English ⇒ Propositional Logic

• Consider the English sentence: "Mary likes flowers and chocolate"

• To express this in propositional logic, we’d say something like $A \land B$, where $A$ stands for “Mary likes flowers” and $B$ for “Mary likes chocolate”

• However, there’s a problem here: there’s nothing connecting Mary” (in A) to “Mary” (in B)
  
  – $B$ could stand for something else, like “Sue likes flowers”?
  
  – There is no connection between the ”Mary“ in A and ”Sue“ in B
  
  – But propositional logic has no way of distinguishing this

• The problem is that there’s no connection between objects/individuals within propositional atoms in propositional logic (e.g. no way of linking “Mary” to ”Mary“ in our example)

• i.e. Propositional logic’s atomic unit is too coarse.

• The same holds for all objects in our propositions. Consider the sentence "Mary likes flowers and Sue likes flowers"

• Again, this is expressed as $A \land B$, but we lose the fact that both Mary and Sue like the same thing

• The same also goes for the predicate of the propositions - consider "John loves Mary and Peter loves Sue"
• This is also expressed as $A \land B$, and again we lose information

• As stated previously, propositional logic’s weakness here comes from the fact that its atoms are too coarse.

• Another problem in propositional logic is that there is no way of linking individuals that share some functional relationship.

• For instance, we cannot express the concept "John's father" in propositional logic - "John's father" clearly depends on “John”

• We also cannot speak about a group of individuals together - there’s no way to say "all Peter’s children" in propositional logic - at least not in a way that is robust across different interpretations

1.2 Required extensions

What is needed

In order to address the shortcomings of propositional logic, we need a logic that also includes the following

• A way of identifying distinct objects

• A way of describing and differentiating predicates

• A way of identifying individual objects in terms of others (via a function)

• A way of representing concepts like "all", "some", "none", etc. This also requires us to be able to talk about arbitrary objects, and thus requires some notion of a “placeholder” (or variable) for objects

These are exactly the extensions that give us first-order logic

1.3 First-order logic - what does it mean?

Propositional logic vs first-order logic

Epistemologically (what is knowledge, what can we know):

• Propositional logic considers truth and falsity

• First-order logic considers truth and falsity

Ontologically (what exists, what are the “things” we can talk about):

• Propositional logic considers facts (i.e. propositions)

• First-order logic considers objects, their properties/relations and functions over the objects
Informal semantics: objects, variables and functions

- First-order logic allows us to talk about individual objects from some domain
- These objects are represented in the logic by names - each such name corresponds uniquely to some object in the "real world"
- Examples: people, numbers, animals, cars, etc
- Variables in first-order logic act as placeholders for objects - they can be instantiated with any name representing an object from the domain
- Examples: the variable $x$ could stand for the object Ferrari (a car), or for the object John (a person), or any other object from our domain
- Functions in first-order logic allow us to talk about objects indirectly - i.e. we refer to them in terms of the other objects that determine them
- Examples: fatherOf(John) represents the object that is the father of John

Informal semantics: predicates

- A predicate in first-order logic is a way of specifying properties of objects, or relationships between them
- These predicates are represented in the logic using predicates names
- Each predicate also specifies how many "arguments" it expects
- Examples:
  - isRed(Ferrari) represents the property of being red (for the object Ferrari) - isRed expects 1 argument
  - onTopOf(Computer, Table) represents the idea that the object Computer is on top of the object Table - onTopOf expects 2 arguments
  - gives(John, Sue, Flowers) represents the idea that John gives Sue Flowers - gives expects 3 arguments
- Predicates (with the suitable number of arguments) correspond to the atoms of propositional logic

Informal semantics: quantifiers

- In first-order logic, we use two quantifiers to talk about objects as a group
- The first is called the existential quantifier. It represents that "there exists something (1 or more) with the specified property"
- This quantifier can also be used negatively to say "there is nothing with the specified property"
- Examples:
there exists x: isRed(x) means that there is at least one "thing" that is red
there does NOT exist x: isRed(x) means that there is nothing that is red

For the existential quantifier, we use the symbol $\exists$

The second quantifier is called the universal quantifier. It represents that "everything has the specified property"

This quantifier can also be used negatively to say "something does not have the specified property"

Examples:

- All x: isRed(x) means that everything is red
- NOT all x: isRed(x) means that there is at least one "thing" that is not red

For the universal quantifier, we use the symbol $\forall$

Informal semantics: connectives

As for propositional logic, we use connectives to combine formulae into more complex formulae

The connectives are the same as for propositional logic, and have the same intended meaning

Representing knowledge/translating from natural language

Let us now reconsider the examples we gave at the start of this section, to see how the extension to first-order logic helps

Eng "Mary likes flowers and chocolate"
FOL likes(Mary, Flowers) $\land$ likes(Mary, Chocolate)

Eng "Mary likes flowers and Sue likes flowers"
FOL likes(Mary, Flowers) $\land$ likes(Sue, Flowers)

Eng "John loves Mary and Peter loves Sue"
FOL loves(John, Mary) $\land$ loves(Peter, Sue)

Eng "John’s father"
FOL fatherOf(John)
Eng "John’s father is sleeping"
FOL \( isSleeping(fatherOf(John)) \)

Eng "All Peter’s children ...
FOL \( \forall x \ (childOf(x, Peter)) \)

Eng "All Peter’s children are sleeping"
FOL \( \forall x \ (childOf(x, Peter) \rightarrow isSleeping(x)) \)

2 Syntax

2.1 Grammar: the language of first-order logic

Alphabet

- The language \( \mathcal{L} \) of first-order logic has the following alphabet:
  - An countably infinite set of variables: we use \( x, y, z, ... \)
  - A set of function symbols: we use \( f, g, ... \)
  - A set of predicate symbols: we use \( p, q, r ... \)
  - For each of these, we possibly use subscripts: e.g. \( x_1, x_2, ... \)
  - The set of connectives \( \{ \land, \lor, \neg, \rightarrow, \leftrightarrow \} \)
  - The quantifiers \( \forall \) and \( \exists \)
  - Punctuation and grouping symbols: parentheses, comma

- Each function and predicate symbol has a fixed arity. To indicate this, they are sometimes written in the form \( p/n \) (predicate symbol \( p \) has arity \( n \))

- Nullary functions are called constants - they represent the names of individuals in our domain. Instead of writing constants as \( c() \), we usually just write \( c \)

Terms

- Terms are all the possible objects that first-order logic can talk about

- We usually use the symbols \( s, t, ... \) (possibly with subscripts) to denote arbitrary/generic terms

- The terms of first-order logic are defined as:
  - Every constant \( c \) is a term
  - Every variable \( x \) is a term
  - If \( f/n \) is a function symbol and \( t_1, t_2, ..., t_n \) are terms, then \( f(t_1, t_2, ..., t_n) \) is a term
Grammar

• $\mathcal{L}$, the language of first-order logic, is the minimal set of formulae such that
  
  – if $p/n$ is a predicate symbol, and $t_1$, $t_2$, ..., $t_n$ are terms, then $p(t_1, t_2, \ldots, t_n)$ is an atomic formula or atom
  
  – If $F$ is a formula, then so is $\neg F$
  
  – If $F$ and $G$ are formulae, then so are $F \land G$, $F \lor G$, $F \rightarrow G$ and $F \leftrightarrow G$
  
  – If $F$ is a formula, then so are $\exists x F$ and $\forall x F$

• Literals are atoms (positive literals) or negated atoms (negative literals)

• Parentheses can be omitted along this order of precedence:

| $\exists$, $\forall$ | $\neg$ | $\land$ | $\lor$ | $\rightarrow$ | $\leftrightarrow$ |

2.2 Variables

Bound and free variables

• Let $\text{var}(F)$ be the set of variables appearing in formula $F$

• $F$ is ground if $\text{var}(F) = \emptyset$

• In the formula $\forall x F$, $F$ is the scope of $\forall$, and $\forall$ is applied to $F$

• Similarly for $\exists x F$

• We say a variable $x$ in a $F$ is bound if it appears within the scope of some quantifier over that variable (i.e. $\forall x$ or $\exists x$)

• In the case of nesting of quantifiers, variables are bound by quantifiers of smallest scope (innermost quantifiers)

• A variable that is not bound is free

• A formula is closed if it contains no free variables (it’s then also called a sentence)

3 Semantics

3.1 Introduction

Introduction

• As for propositional logic, first-order logic is a two-valued logic

• First-order logic satisfies both the properties of bivalence and contradiction
3.2 Interpretations

Interpretations

• As for propositional logic, an interpretation is the way in which we formally assign meaning.

• For first-order logic, an interpretation $I$ on an alphabet $A$ is a non-empty domain $D$ and a mapping that associates
  - to each constant $c \in A$ an element $c_I \in D$
  - to each function symbol $f/n \in A$ a function $f_I : D^n \rightarrow D$
  - to each predicate symbol $p/n \in A$ a relation $p_I \subseteq D^n$

Valuations

• Because we also have variables in first-order logic, we need one additional thing before we can completely assign meaning - a valuation.

• A valuation is a mapping from the variables of $A$ to $D$ - i.e. it is an assignment of objects from our domain to the variables of our alphabet.

• For a valuation $\alpha$, interpretation $I$ and term $t$, the meaning of $\alpha_I(t)$ is
  - $c_I$ if $t$ is a constant $c$
  - $\alpha(x)$ if $t$ is a variable $x$
  - $f_I(\alpha_I(t_1),\ldots,\alpha_I(t_n))$ if $t$ is of the form $f(t_1,t_2,\ldots,t_n)$

Example

• Example: consider the term $g(c,f(x),y)$, and

  the interpretation $I = (D, \cdot)$ such that
  - $D = \mathbb{N}$
  - $c_I = 5$
  - $f_I(x) = x^2$
  - $g_I(x,y,z) = (x \times y) - z$

• Let $\alpha$ be a valuation such that $\alpha(x) = 2$ and $\alpha(y) = 20$

• Then,

$$\alpha_I(g(c,f(x),y)) = g_I(\alpha_I(c),\alpha_I(f(x)),\alpha_I(y))$$
$$= (\alpha_I(c) \times \alpha_I(f(x))) - \alpha_I(y)$$
$$= (c_I \times f_I(\alpha(x))) - \alpha(y)$$
$$= (5 \times (\alpha(x))^2) - 20$$
$$= (5 \times (\alpha(x))^2) - 20$$
$$= (5 \times 2^2) - 20$$
$$= 0$$
Semantic rules

- Let $\alpha$ be a valuation, $\mathcal{I}$ an interpretation with domain $D$, $x$ a variable, and $c \in D$
- $\alpha[x \mapsto c]$ is identical to $\alpha$, except that $x$ is mapped to $c$
- $\mathcal{I} \vDash \alpha F$ means that $F$ is satisfied by $\mathcal{I}$ and $\alpha$ (i.e. $F$ is made True by $\mathcal{I}$ and $\alpha$)

- The semantic rules for first-order logic are given below

$$
\begin{align*}
\mathcal{I} \vDash \alpha p(t_1, t_2, ..., t_n) & \quad \text{iff} \quad \langle \alpha I(t_1), \alpha I(t_2), ..., \alpha I(t_n) \rangle \in pI \\
\mathcal{I} \vDash \alpha \neg F & \quad \text{iff} \quad \mathcal{I} \nvDash \alpha F \\
\mathcal{I} \vDash \alpha F \land G & \quad \text{iff} \quad \mathcal{I} \vDash \alpha F \text{ and } \mathcal{I} \vDash \alpha G \\
\mathcal{I} \vDash \alpha F \lor G & \quad \text{iff} \quad \mathcal{I} \vDash \alpha F \text{ or } \mathcal{I} \vDash \alpha G, \text{ or both} \\
\mathcal{I} \vDash \alpha F \rightarrow G & \quad \text{iff} \quad \mathcal{I} \nvDash \alpha F \text{ or } \mathcal{I} \vDash \alpha G, \text{ or both} \\
\mathcal{I} \vDash \alpha F \leftrightarrow G & \quad \text{iff} \quad \mathcal{I} \vDash \alpha F \rightarrow G \text{ and } \mathcal{I} \vDash \alpha G \rightarrow F \\
\mathcal{I} \vDash \alpha \forall x F & \quad \text{iff} \quad \mathcal{I} \vDash \alpha[x \mapsto c] F \text{ for every } c \in D \\
\mathcal{I} \vDash \alpha \exists x F & \quad \text{iff} \quad \mathcal{I} \vDash \alpha[x \mapsto c] F \text{ for some } c \in D
\end{align*}
$$

Models

- If $F$ is a sentence, its meaning depends only on the interpretation - in these cases, we write $\mathcal{I} \vDash F$
- $\mathcal{I}$ is a model of $F$ iff $\mathcal{I} \vDash \alpha F$ for every valuation $\alpha$.
  - we write $\mathcal{I} \vDash F$
- $\mathcal{I}$ is a model of a set of formulae $\mathcal{F}$ iff it is a model for each $F \in \mathcal{F}$
  - we write $\mathcal{I} \vDash \mathcal{F}$

Example I

- Let $\mathcal{I} = (D, \cdot)$ be an interpretation with $D = \mathbb{N}$ and such that
  
  Constants: $c_I = 0$

  Functions: $f_I(x) = x + 1$ (i.e. the successor function)

  Predicates: $p_I = \{(1), (3), (5), ...\}$ (i.e. the odd numbers)

- The meaning of the formula $p(c) \land p(f(c))$ in $\mathcal{I}$ is then as follows:

$$
\begin{align*}
\mathcal{I} \vDash p(c) \land p(f(c)) & \quad \text{iff} \quad \mathcal{I} \vDash p(c) \text{ and } \mathcal{I} \vDash p(f(c)) \\
& \quad \text{iff} \quad (\alpha(x(c)) \in p_I \text{ and } (\alpha(x(f(c)))) \in p_I \\
& \quad \text{iff} \quad (0) \in p_I \text{ and } (0 + 1) \in p_I
\end{align*}
$$

- but $(0) \notin p_I$, so $\mathcal{I}$ is not a model for the formula

51
Example II

• Consider the following interpretations on alphabet $A = \{j, p, m, s\}$

$I_1 = (D, \cdot_1)$
$I_2 = (D, \cdot_2)$
$I_3 = (D, \cdot_3)$

- where $D = \{John, Mary, Peter, Sue\}$

- and for each interpretation $I_i$

  $I_i(j) = John$, $I_i(p) = Peter$, $I_i(m) = Mary$, $I_i(s) = Sue$

  isMarried$_{I_i} = \{John, Peter, Mary, Sue\}$

  spouseOf$_{I_i}(x)$ returns the spouse of $x$ (Mary is married to John, Sue is married to Peter)

- and loves$_{I_1} = \{(John, Mary), (Mary, John), (Peter, Sue), (Sue, Peter)\}$

- and loves$_{I_2} = \{(Peter, Sue), (Sue, Peter), (Mary, John)\}$

- and loves$_{I_3} = \{(John, Mary), (Mary, John), (Peter, Sue), (Sue, Peter), (Peter, Mary)\}$

• and consider the formulae

$F = \forall x(isMarried(x) \rightarrow (loves(spouseOf(x), x) \land loves(x, spouseOf(x))))$

$G = \forall x(isMarried(x) \rightarrow \neg\exists y(loves(x, y) \land y \neq spouseOf(x)))$

• Then,

$I_1 \vDash F \quad I_1 \vDash G$

$I_2 \not\vDash F \quad I_2 \vDash G$

$I_3 \vDash F \quad I_3 \not\vDash G$

3.3 Exercises I

1. Translate the following formulas into natural language (English), according to their intuitive intended meaning:

   (a) $\forall x(male(x) \lor female(x))$

   (b) $\forall x((father(x) \rightarrow male(x)) \land (mother(x) \rightarrow female(x)))$

   (c) $\neg\exists x(father(x) \land mother(x))$

2. Translate the following English sentences into first-order logic formulas. Invent a suitable vocabulary.

   (a) Some students take Formal Logics

   (b) No good student flunks an exam

   (c) Brothers are siblings (express the fact that sibling is a symmetric relation)

   (d) Ones mother is ones female parent

   (e) A cousin is a child of a parents sibling

52
(f) Every person has only one mother

(g) (a famous quotation): you can fool some people all of the time, and all
of the people some of the time, but you cannot fool all of the people all
of the time.

3. Argue why $\exists x \forall y F(x, y)$ is not logically equivalent to $\forall y \exists x F(x, y)$. Comment
on the formula $F(x, y) = mother(x, y)$.

3.4 Semantic properties of formulae

Validity, Falsifiability

- Recall: If $I$ satisfies $F$ for all valuations $\alpha$, then $I$ is a model for $F$ and we
  write $I \models F$
- $F$ is valid iff all interpretations are models for $F$ (i.e. $I \models F$ for all possible
  interpretations $I$)
  $F$ is called a tautology
  We write this simply as $\models F$
- $F$ is falsifiable iff it is not valid - i.e. iff there is at least one interpretation
  that does not satisfy $F$ (i.e. $I \not\models F$ for some interpretation $I$)

Satisfiability

- $F$ is satisfiable iff there is at least one interpretation that satisfies $F$ (i.e. $I \models F$ for some interpretation $I$)
- $F$ is unsatisfiable iff it is not satisfiable - i.e. iff no interpretation satisfies $F$
  (i.e. $I \not\models F$ for all interpretations $I$)
- If $F$ is both satisfiable and falsifiable, then it is contingent

Two special formulae: $\top$ and $\bot$

- As in propositional logic, we have the two special formulae $\bot$ ("bottom")
  and $\top$ ("top")
- As in propositional logic, $\bot$ is always False and $\top$ is always True

Entailment

- A set of formulae $\mathcal{F}$ logically entails a formula $G$ if every model of $\mathcal{F}$ is also
  a model for $G$.
- We write $\mathcal{F} \models G$
- If $\mathcal{F} \models G$, we also say that $G$ is a logical consequence of $\mathcal{F}$
Deduction
The following theorem is useful when proving entailment

Deduction theorem
Let $\mathcal{F}$ be a set of formulae, and $G$ and $H$ be formulae Then $\mathcal{F} \vdash G \rightarrow H$ iff $\mathcal{F} \cup \{G\} \vdash H$

Another useful theorem is this one

Theorem
$F$ is valid iff $\neg F$ is unsatisfiable

Example

• Prove that $p(A) \land q(B) \vdash p(A) \lor q(B)$
  • By the deduction theorem, it is sufficient to prove
    $\vdash (p(A) \land q(B)) \rightarrow (p(A) \lor q(B))$
  • By the second theorem on the previous slide, it is sufficient to prove that
    $\neg((p(A) \land q(B)) \rightarrow (p(A) \lor q(B)))$
    is unsatisfiable

Now,

$I \models \neg((p(A) \land q(B)) \rightarrow (p(A) \lor q(B)))$
iff $I \notmodels (p(A) \land q(B)) \rightarrow (p(A) \lor q(B))$
iff $I \models (p(A) \land q(B))$ and $I \notmodels (p(A) \lor q(B))$
iff $I \models p(A)$ and $I \models q(B)$ and $I \notmodels p(A)$ and $I \notmodels q(B)$
contradiction

\[ \therefore \neg((p(A) \land q(B)) \rightarrow (p(A) \lor q(B))) \text{ is unsatisfiable} \]
\[ \therefore \vdash (p(A) \land q(B)) \rightarrow (p(A) \lor q(B)) \]
\[ \therefore p(A) \land q(B) \vdash p(A) \lor q(B) \]

Equivalence

• Two formulae $F$ and $G$ are semantically equivalent
  iff they have the same truth value under all interpretations
  iff $F \vdash G$ and $G \vdash F$

• We write $F \equiv G$
Useful (and named) equivalences

\[ F \land F \equiv F \] idempotency of \( \land \)

\[ F \lor F \equiv F \] idempotency of \( \lor \)

\[ F \land G \equiv G \land F \] commutativity of \( \land \)

\[ F \lor G \equiv G \lor F \] commutativity of \( \lor \)

\[ F \land (G \land H) \equiv (F \land G) \land H \] associativity of \( \land \)

\[ F \lor (G \lor H) \equiv (F \lor G) \lor H \] associativity of \( \lor \)

\[ F \land (G \lor H) \equiv (F \land G) \lor (F \land H) \] distributivity of \( \land \)

\[ F \lor (G \land H) \equiv (F \lor G) \land (F \lor H) \] distributivity of \( \lor \)

\[ (F \land G) \lor F \equiv F \] absorption

\[ (F \lor G) \land F \equiv F \] absorption

\[ \neg F \equiv F \] double negation

\[ \neg(F \land G) \equiv \neg F \land \neg G \] De Morgan’s Law

\[ \neg(F \lor G) \equiv \neg F \land \neg G \] De Morgan’s Law

\[ F \iff G \equiv (F \rightarrow G) \land (G \rightarrow F) \] equivalence

\[ F \iff G \equiv \neg(F \land \neg G) \land (G \land \neg F) \] contraposition

\[ \forall x(F) \equiv \neg(\exists x(\neg F)) \] quantifier duality

\[ \exists x(F) \equiv \neg(\forall x(\neg F)) \] quantifier duality

\[ (\forall x(F)) \land (\forall x(G)) \equiv \forall x(F \land G) \] (1)

\[ (\exists x(F)) \lor (\exists x(G)) \equiv \exists x(F \lor G) \] (2)

\[ \forall x(\forall y(F)) \equiv \forall y(\forall x(F)) \] (3)

\[ \exists x(\exists y(F)) \equiv \exists y(\exists x(F)) \] (4)

\[ (\forall x(F)) \land G \equiv \forall x(F \land G) \] (5)

\[ (\forall x(F)) \lor G \equiv \forall x(F \lor G) \] (6)

\[ (\exists x(F)) \land G \equiv \exists x(F \land G) \] (7)

\[ (\exists x(F)) \lor G \equiv \exists x(F \lor G) \] (8)

* provided that \( G \) does not contain a free occurrence of \( x \)

Example

We show, via equivalences, that

\[ (\forall x \ p(x)) \lor (\exists y \neg q(y)) \equiv \forall x \exists y (\neg p(x) \lor \neg q(y)) \]

\[ (\forall x \ p(x)) \lor (\exists y \neg q(y)) \equiv \forall x \left( p(x) \lor (\exists y \neg q(y)) \right) \] equivalence (6) above

\[ \equiv \forall x \left( (\exists y \neg q(y)) \lor p(x) \right) \] commutativity of \( \lor \)

\[ \equiv \forall x \left( \exists y (\neg q(y) \lor p(x)) \right) \] equivalence (8) above

\[ \equiv \forall x \exists y \left( q(y) \rightarrow p(x) \right) \] material implication

\[ \equiv \forall x \exists y \left( \neg p(x) \rightarrow \neg q(y) \right) \] contraposition
3.5 Herbrand interpretations

Herbrand interpretations

- **Herbrand interpretations** help us to compute logical entailment - instead of searching for models in all possible interpretations, we search only a subset of those, namely the Herbrand interpretations.

- Consider a first-order logic language $\mathcal{L}$, with an alphabet $A$.
  - The **Herbrand universe** $U^A_H$ of $\mathcal{L}$ is the set of all ground terms that we can construct from $A$.
  - The **Herbrand base** $B^A_H$ of $\mathcal{L}$ is the set of all ground atoms that we can construct from $A$.

- Example: Assume $A$ contains the set of constants $\{a, b\}$, the set of function symbols $\{f/1\}$, and the set of predicate symbols $\{p/1\}$.
  - Then the Herbrand universe is $U^A_H = \{a, b, f(a), f(b), f(f(a)), f(f(b)), \ldots\}$.
  - and the Herbrand base is $B^A_H = \{p(a), p(b), p(f(a)), p(f(b)), p(f(f(a))), p(f(f(b))), \ldots\}$.

- A **Herbrand interpretation** for $\mathcal{L}$ is an interpretation $I = (D, \cdot)$ such that:
  - $D = U^A_H$.
  - Every constant $c$ is mapped to itself (i.e. $c_I = c$).
  - For every function symbol $f/n$, we have a function $f_I : (U^A_H)^n \rightarrow U^A_H$, that maps $(t_1, t_2, \ldots, t_n)$ to $f(t_1, t_2, \ldots, t_n)$ (i.e. to the term in $U^A_H$).
  - Every predicate symbol $p/n$ is mapped to $p_I \subseteq (U^A_H)^n$.

The first 3 points above (domain, constants and function symbols) yield a pre-interpretation - this pre-interpretation is unique for each alphabet.

- A **Herbrand model** for a formula $G$ (or for a set of formulae $F$), is a Herbrand interpretation that is a model of $G$ (or $F$).

- A Herbrand interpretation can be seen as subset of the Herbrand base $B^A_H$.

Conjunctive normal form

- A formula $F$ is in **conjunctive normal form** (CNF) if it is of the form $\forall x_1 \forall x_2 \ldots \forall x_n (F)$, and $F$ is a formula that is a conjunction of disjunctions of literals (positive or negative).

- e.g.
  - $\forall x \forall y ((p(x) \lor \neg p(y)) \land (q(x) \lor r(x) \lor q(y) \lor r(y)))$ is in CNF.
  - $\forall x \exists y ((p(x) \lor p(y)) \land (q(x) \lor r(x)))$ is NOT in CNF.
  - $\forall x (\neg p(x) \land p(y))$ is in CNF.
  - $\forall x (\neg p(x) \land p(y))$ is NOT in CNF.
• CNF is important to us because of the following theorem

**Theorem: CNF and Herbrand interpretations**
A first-order logic formula $F$ in CNF is unsatisfiable iff $F$ is false under all Herbrand interpretations

• It turns out that there is a (mechanical) way to convert any first-order logic formula into CNF

• However, it depends on a transformation step (called Skolemisation) that is not **model preserving** (i.e. the resultant CNF formula is not necessarily equivalent to the one we started with)

• However, the form of Skolemisation used is **satisfiability preserving** (i.e. the resultant CNF formula is (un)satisfiable iff the original one is)

• Skolemisation works as follows:
  
  1. $(\exists x_1 \exists x_2 ... \exists x_n)(\forall y)F$ is rewritten as
     
     $(\exists x_1 \exists x_2 ... \exists x_n)F[y/f(x_1, x_2, ..., x_n)]$
     
     where $f$ is a new function symbol not appearing in $F$

  2. $(\forall x_1 \forall x_2 ... \forall x_n)(\exists y)F$ is rewritten as
     
     $(\forall x_1 \forall x_2 ... \forall x_n)F[y/f(x_1, x_2, ..., x_n)]$
     
     where $f$ is a new function symbol not appearing in $F$

• $f$ is called the **skolem function**, or **skolem constant** if $n = 0$

• The first form of Skolemisation is **validity preserving**

• The second form is **satisfiability preserving** and is the form of Skolemisation that we use when converting to CNF.

**Testing logical consequence**
The following approach describes how we can test entailment:

• Checking logical consequence can be converted into checking validity, via the deduction theorem

• Validity, in turn, can be the same as checking unsatisfiability of the complement

• The complement can be transformed into CNF (which is equi-(un)satisfiable)

• The CNF can be checked for unsatisfiability under all Herbrand interpretations (using a negative calculus)

• **BUT** this is not a decision procedure, because the calculus is not guaranteed to terminate (see next slide)
4 Decidability

Undecidable

- We saw that propositional logic is decidable
- Bad news for first-order logic, though - in general, first-order logic is undecidable
- However, logical entailment in first-order logic is actually semi-decidable - i.e. there are procedures that
  - if $F \models G$, always terminate and answer yes
  - if $F \not\models G$, may terminate and answer no, or may not answer at all

4.1 Exercises II

1. Use equivalences to show that $\forall y(\exists x p(x, y) \rightarrow \neg q(y))$ is logically equivalent to $\neg \exists y \exists x (p(x, y) \land q(y))$

2. Use equivalences to show that $\forall y(\exists x (p(x, y) \rightarrow \neg q(y)))$ is logically equivalent to $\neg \exists y \forall x (p(x, y) \land q(y))$

3. Show whether the following entailments hold. If the entailment does not hold, provide a counter-example.
   (a) $\forall x(\neg p(x) \lor q(x)) \models \forall x(p(x) \rightarrow q(x))$
   (b) $\forall x \exists y(p(x, y)) \models \exists y \forall x(p(x, y))$

4. Show the Herbrand universe and Herbrand base for the following formulas:
   (a) $\text{odd}(s(0)) \land \forall x ((\text{odd}(x) \rightarrow \text{odd}(s(s(x)))))$
   (b) $\text{has}(@\text{owner}(\text{car}), \text{car}) \land \forall x (\text{has}(x, \text{car}) \rightarrow \text{happy}(x))$
   (c) $\neg p(a) \land \exists x r(x)$

5 References and further reading

References and further reading


Acknowledgement

Some of the examples are exercises used in this course were based on those used by Davide Martinenghi in his course on Computational Logic.
Part V
Solutions

Solutions to exercises

A Propositional logic

A.1 Exercises I

1. Develop truth tables for the following:

(a) \((A \land B) \lor \neg C\)

Answer:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>((A \land B) \lor \neg C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

(b) \((A \land B) \lor (\neg A \lor \neg B)\)

Answer:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>((A \land B) \lor (\neg A \lor \neg B))</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

(c) \((A \lor B) \lor \neg (A \lor (B \land C))\)

Answer:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>((A \lor B) \lor \neg (A \lor (B \land C)))</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
2. Translate the following English sentences into sentences of propositional logic. Use only the propositions we provide.

For instance, the sentence, "It is either raining or snowing", with the logical propositions:

- \( r \) = “It is raining”
- \( s \) = “It is snowing”

should be answered by: \( r \lor s \)

(a) “To become my girlfriend, you must be smart, pretty, and nice”
- \( g \) = “become my girlfriend”
- \( s \) = “you are smart”
- \( p \) = “you are pretty”
- \( n \) = “you are nice”

Answer: \( g \to (p \land s \land v) \)

(b) “Unless I go to Korea or Japan, I will not be able to attend a World Cup match”
- \( k \) = “I go to Korea”
- \( j \) = “I go to Japan”
- \( w \) = “I will be able to attend a World Cup match”

Answer: \( \neg(k \lor j) \to \neg w \)

(c) “I am willing to help whenever you have a problem, except when I’m sleeping or very busy”
- \( w \) = “I am willing to help”
- \( h \) = “You have a problem”
- \( a \) = “I am sleeping”
- \( b \) = “I am very busy”

Answer: \( (h \land \neg a \land \neg b) \to w \)

A.2 Exercises II

1. Determine whether each of the following is valid, contingent or unsatisfiable.

(a) \( \neg (B \land \neg C \land \neg B) \)

Answer:

<table>
<thead>
<tr>
<th>( B )</th>
<th>( C )</th>
<th>( \neg (B \land \neg C \land \neg B) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>
∴ it is valid (and thus not falsifiable, not contingent, and not unsatisfiable)

(b) \( \neg(A \lor B \land \neg C) \lor (A \lor B \land \neg C) \)

Answer:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>( \neg(A \lor B \land \neg C) \lor (A \lor B \land \neg C) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

∴ it is valid (and thus not falsifiable, not contingent, and not unsatisfiable)

(c) \( \neg(A \land (\neg A \lor (B \land C))) \lor B \)

Answer:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>( \neg(A \land (\neg A \lor (B \land C))) \lor B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

∴ it is valid (and thus not falsifiable, not contingent, and not unsatisfiable)

2. Show by truth tabling that:

\[ A \rightarrow B \land C \models A \rightarrow C \]

Answer:
The models of $A \rightarrow B \land C$ are $I_1$, $I_5$, $I_6$, $I_7$, and $I_8$. Each of these also satisfy $A \rightarrow C$, so $\therefore A \rightarrow B \land C \models A \rightarrow C$

Alternatively: To prove that $A \rightarrow B \land C \models A \rightarrow C$, it is sufficient to prove that $\vdash (A \rightarrow B \land C) \rightarrow (A \rightarrow C)$.
This is proven in the truth table $\therefore A \rightarrow B \land C \models A \rightarrow C$

3. Show that two formulae $F$ and $G$ are semantically equivalent ($F \equiv G$) iff $F \leftrightarrow G$ is a tautology.

**Answer:** If $(F \equiv G)$, then

$$F \models G \quad \text{and} \quad G \models F$$

By the Deduction theorem

$$\vdash F \rightarrow G \quad \text{and} \quad \vdash G \rightarrow F$$

$\therefore \vdash F \leftrightarrow G$

Similarly, if $(F \leftrightarrow G)$ is a tautology, then

$$\vdash F \rightarrow G \quad \text{and} \quad \vdash G \rightarrow F$$

This implies that $F \models G \quad \text{and} \quad G \models F$

$\therefore F \equiv G$

4. Let $\tau$ and $\Delta$ be sets of sentences in propositional logic, and let $\gamma$ be an individual sentence in propositional logic. State whether each of the following statements is true or false.

- **If $\tau \models \gamma$ and $\Delta \models \gamma$ then $\tau \cup \Delta \models \gamma$**

**Answer:** True

Let $M(\tau)$ and $M(\Delta)$ be the sets of models of $\tau$ and $\Delta$, respectively, and $M(\tau \cup \Delta)$ be the set of models of their union. Similarly, $M(\gamma)$ is the set of models for $\gamma$

Now, if some interpretation $I$ is a model of $\tau \cup \Delta$, then it must satisfy every formula in $\tau$, as well as every formula in $\Delta$.

In other words, if $I \in M(\tau \cup \Delta)$, then $I \in M(\tau)$ and $I \in M(\Delta)$

i.e. $M(\tau \cup \Delta) \subseteq M(\tau) \cap M(\Delta)$

63
But every model for $\tau$ is also a model for $\gamma$ because $\tau \models \gamma$, i.e. $M(\tau) \subseteq M(\gamma)$

Also, every model for $\Delta$ is also a model for $\gamma$ because $\Delta \models \gamma$, i.e. $M(\Delta) \subseteq M(\gamma)$

\[ \therefore M(\tau \cup \Delta) \subseteq M(\tau) \cap M(\Delta) \subseteq M(\gamma) \]

\[ \therefore \tau \cup \Delta \models \gamma \]

- If $\tau \models \gamma$ and $\Delta \models \gamma$ then $\tau \cap \Delta \not\models \gamma$

**Answer:** False

It should be intuitively clear after the previous example. A counter-example suffices:

Let $\tau = \{ A, A \to B \}$ and $\Delta = \{ A, B \}$, and let $\gamma$ be the formula $B$

Then $\tau \cap \Delta = \{ A \}$.

Clearly, $\tau \models \gamma$ and $\Delta \models \gamma$, but $\tau \cap \Delta \not\models \gamma$

5. Prove: $F$ is valid iff $\neg F$ is unsatisfiable

**Answer:** $F$ is valid iff for every interpretation $I$, we have that $I \models F$

if $\text{every } I \not\models \neg F$

if there is no $I$ such that $I \models \neg F$

if $\neg F$ is unsatisfiable.

**B First-order logic**

**B.1 Exercises I**

1. Translate the following formulas into natural language (English), according to their intuitive intended meaning:

   (a) $\forall x (\text{male}(x) \lor \text{female}(x))$

   **Answer:** Everyone is either male or female.

   (b) $\forall x ((\text{father}(x) \to \text{male}(x)) \land (\text{mother}(x) \to \text{female}(x)))$

   **Answer:** All fathers are male and all mothers are female.

   (c) $\neg \exists x (\text{father}(x) \land \text{mother}(x))$

   **Answer:** No-one is both a father and a mother.

2. Translate the following English sentences into first-order logic formulas. Invent a suitable vocabulary.

   (a) Some students take Formal Logics

   **Answer:** $\exists x (\text{student}(x) \land \text{takes}(x, \text{FormalLogic}))$

   (b) No good student flunks an exam
Answer: \(\neg\exists x, y (\text{good student}(x) \land \text{flunks}(x, y))\)

or \(\neg\exists x, y (\text{good student}(x) \land \text{exam}(y) \land \text{flunks}(x, y))\)

(c) Brothers are siblings (express the fact that sibling is a symmetric relation)
Answer: \(\forall x, y (\text{brother}(x, y) \rightarrow \text{brother}(y, x))\)

or \(\forall x, y (\text{brother}(x, y) \rightarrow \text{sibling}(x, y))\)
\(\forall x, y (\text{sibling}(x, y) \rightarrow \text{sibling}(y, x))\)

Answer: \(\forall x, y (\text{mother}(x, y) \rightarrow (\text{parent}(x, y) \land \text{female}(x)))\)

(e) A cousin is a child of a parents sibling
Answer: \(\forall w, x, y, z (\text{cousin}(x, y) \leftrightarrow (\text{parent}(z, x) \land \text{parent}(w, y) \land \text{sibling}(w, z)))\)

(f) Every person has only one mother
Answer: \(\forall x, y, z ((\text{mother}(x, y) \land \text{mother}(z, y)) \rightarrow x = z)\)

Answer: \(\forall t_1 \exists p_1 \text{fool}(t_1, p_1) \land \exists p_2 \forall p_2 \text{fool}(t_2, p_2) \land \neg \forall t_3 \forall p_3 \text{fool}(t_2, p_3)\)

3. Argue why \(\exists x \forall y F(x, y)\) is not logically equivalent to \(\forall y \exists x F(x, y)\). Comment on the formula \(F(x, y) = \text{mother}(x, y)\).
Answer: \(\exists x \forall y F(x, y)\) means that there’s at least one object that links to every object in our domain via \(F\)
\(\forall y \exists x F(x, y)\) means that every object from our domain is linked to by a least one object via \(F\) (but they aren’t necessarily all linked to the same object)

\(\exists x \forall y \text{mother}(x, y)\) means that there’s at least one person that is the mother of everyone (incl. herself)
\(\forall y \exists x \text{mother}(x, y)\) means that everyone has at least one mother (but not necessarily that everyone has the same mother)

B.2 Exercises II

1. Use equivalences to show that \(\forall y ((\exists x p(x, y)) \rightarrow \neg q(y))\) is logically equivalent to \(\neg \exists y \exists x (p(x, y) \land q(y))\)
Answer:
\[
\forall y (\exists x \, p(x, y)) \rightarrow \neg(q(y)) \equiv \neg \exists y (\exists x \, p(x, y) \rightarrow \neg q(y)) \, \text{quantifier duality}
\]
\[
\equiv \neg \exists y (\neg (\exists x \, p(x, y)) \lor \neg q(y)) \, \text{material implication}
\]
\[
\equiv \neg \exists y (\exists x \, p(x, y) \land \neg q(y)) \quad \text{De Morgan}
\]
\[
\equiv \neg \exists y (\exists x \, p(x, y) \land q(y)) \, \text{double negation (x2)} \quad (7)
\]

2. Use equivalences to show that \(\forall y (\exists x (p(x, y) \rightarrow \neg q(y)))\) is logically equivalent to \(\neg \exists y \forall x (p(x, y) \land q(y))\)

Answer:
\[
\forall y (\exists x (p(x, y) \rightarrow \neg q(y))) \equiv \neg \exists y (\exists x (p(x, y) \rightarrow \neg q(y))) \, \text{quantifier duality}
\]
\[
\equiv \neg \exists y (\forall x (p(x, y) \rightarrow \neg q(y))) \quad \text{quantifier duality}
\]
\[
\equiv \neg \exists y \forall x (p(x, y) \rightarrow \neg q(y)) \, \text{double negation}
\]
\[
\equiv \neg \exists y \forall x (\neg p(x, y) \lor \neg q(y)) \quad \text{material implication}
\]
\[
\equiv \neg \exists y \forall x (\neg p(x, y) \land \neg q(y)) \quad \text{De Morgan}
\]
\[
\equiv \neg \exists y \forall x (p(x, y) \land q(y)) \quad \text{double negation (x2)}
\]

3. Show whether the following entailments hold. If the entailment does not hold, provide a counter-example.

(a) \(\forall x (\neg p(x) \lor q(x)) \vdash \forall x (p(x) \rightarrow q(x))\)

Answer: By material implication, we know that
\[
\forall x (\neg p(x) \lor q(x)) \equiv \forall x (p(x) \rightarrow q(x))
\]
Therefore, by definition,
\[
\forall x (\neg p(x) \lor q(x)) \vdash \forall x (p(x) \rightarrow q(x))
\]

(b) \(\forall x \exists y (p(x, y)) \vdash \exists y \forall x (p(x, y))\)

Answer: Counter-example: \(I = (D, \cdot)\) with
\(D = \mathbb{N}\)
\(p_I = "<"\) (the “less than” relation)
Now, \(\forall x \exists y p(x, y)\) says that every integer is “less than” at least one other integer - this formula is satisfied by \(I\)
Now, \(\exists y \forall x p(x, y)\) says that there is some integer that is “less than” every integer (incl. itself) - this is clearly not satisfied by \(I\)
\[
\therefore \forall x \exists y (p(x, y)) \nvdash \exists y \forall x (p(x, y))
\]

4. Show the Herbrand universe and Herbrand base for the following formulas:

(a) \(\text{odd}(s(0)) \land \forall x ((\text{odd}(x) \rightarrow \text{odd}(s(s(x)))))\)

Answer:
\(U_H = \{0, s(0), s(s(0)), \ldots\}\)
\(B_H = \{\text{odd}(0), \text{odd}(s(0)), \text{odd}(s(s(0))), \ldots\}\)
(b) $\text{has(owner(car), car)} \land \forall x (\text{has(x, car)} \rightarrow \text{happy(x)})$

**Answer:**

$U_H = \{\text{car, owner(car), owner(owner(car)), ...}\}$

$B_H = \{\text{has(car, car), has(car, owner(car)), has(owner(car), car), has(owner(car), owner(car)), ...}\} \cup \{\text{happy(car), happy(owner(car)), happy(owner(owner(car))), ...}\}$

(c) $\neg p(a) \land \exists x p(x)$

**Answer:**

$U_H = \{a\}$

$B_H = \{p(a)\}$