

# Complexity of the bi-implication fragment of Intuitionistic logic

Bachelor's thesis  
15 ECTS

Sanne Brinkhorst

*Supervisors:*  
Jeroen Goudsmit  
Albert Visser

August 10, 2011

## Preface

This thesis has not one but two problems it investigates. Both are about intuitionistic logic and its complexity, but the problems are quite different. The first is a comparison between the complexity of the full propositional language and fragments with limited connectives, in this case bi-implication. For classical logic it is proven that the bi-implication fragment is easier\* than the full language, but for intuitionistic logic this is not proven. I expect that it is also possible to have an algorithm in a smaller complexity class for intuitionistic logic, but I did not have enough time and/or knowledge to investigate this further.

The second part is a working-out of a proof that intuitionistic logic is PSPACE-hard. This is a very elegant proof by Švejdar based on the structure of Kripke models, the models we use to represent the semantics of intuitionistic logic. He does not only prove that intuitionistic logic is PSPACE-complete for the full language, but also for the implicational fragment. Statman [5] proved this earlier, but he used proof theoretical methods and not model theoretical methods.

Not all classical axioms are valid in intuitionistic logic. These restrictions might be useful for knowledge systems in artificial intelligence. In classical logic there might be too much propositions that are accepted as true, even the ones that we don't want to accept. Furthermore complexity problems are very important in artificial intelligence. A knowledge system is not very useful if it is very slow, so better algorithms can improve them a great deal.

I wish to thank my supervisors Jeroen Goudsmit and Albert Visser for their ideas and advice. I started with an ambitious time scheme and without their support and patience I would not have finished it.

Thanks to my boyfriend Alexander for the support in practical and abstract matters. Without you I would not have taken the step to study artificial intelligence, a choice that changed my life for the better. Thanks to my parents, family and friends for keeping up with someone who always unjustly thinks that she will have more time for you in the future.

---

\*in case complexity classes P and NP are not equal

# Contents

<b>1</b>	<b>Intuitionistic Logic</b>	<b>5</b>
1.1	Introduction . . . . .	5
1.2	Natural deduction . . . . .	6
1.3	Kripke Models . . . . .	9
1.4	Soundness of Kripke models . . . . .	11
<b>2</b>	<b>Decision problems and Complexity</b>	<b>13</b>
2.1	Complexity . . . . .	14
2.2	Complexity classes . . . . .	15
2.3	Reducibility . . . . .	16
2.4	NP: SAT and TAUT . . . . .	17
2.5	PSPACE: INTAUT and TQBF . . . . .	18
<b>3</b>	<b>Bi-implicational fragment</b>	<b>21</b>
3.1	Deciding bi-implicational formulas in classical logic . . . . .	24
3.2	Complexity of classical decision algorithm . . . . .	27
3.3	Intuitionistic decision algorithm? . . . . .	28
<b>4</b>	<b>Complexity of intuitionistic logic</b>	<b>31</b>
4.1	Reduction from TQBF to INTTAUT . . . . .	31
4.2	PSPACE-completeness of $\mathcal{L}_{\rightarrow}$ . . . . .	33
<b>A</b>	<b>Turing machines</b>	<b>37</b>



# Chapter 1

## Intuitionistic Logic

### 1.1 Introduction

In classical logic, reasoning is based on the assumption that a statement is either true or false. The truth of propositions is based on the truth values of the propositional variables. In this text we will use the set  $\text{Var}$  of propositional variables and lower case latin letters, preferably  $p, q, r$  for variables. Every propositional variable  $\text{Var}$  has a Boolean value. Boolean values are the set  $\mathbb{B} = \{0, 1\}$  such that if a variable is true, the Boolean value is 1 and if the variable is false the value is 0. We can also assign Boolean values to propositions, which are sentences of variables and logical connectives. Boolean values are always computed modulo 2 because they can not be bigger than 1.

In this text, propositional formulas will be built up from the binary connectives  $\vee, \wedge$  and  $\rightarrow$ , the unary connective  $\neg$ , variables and the symbols  $\top$  for truth and  $\perp$  for falsity.

**Definition 1.1** (Propositional language).  $\mathcal{L}_p = \text{Var} \mid \top \mid \perp \mid \mathcal{L}_p \rightarrow \mathcal{L}_p \mid \mathcal{L}_p \wedge \mathcal{L}_p \mid \mathcal{L}_p \vee \mathcal{L}_p \mid \neg \mathcal{L}_p$

A valuation function  $v$  is a function  $\text{Var} \rightarrow \mathbb{B}$  which returns for every variable its Boolean value. To calculate the Boolean value of a proposition we use  $\llbracket - \rrbracket$ , a function of  $(\text{Var} \rightarrow \mathbb{B}) \rightarrow \mathbb{B}$ . It calculates the Boolean value of a connective based on the valuation function of its arguments.

$$\begin{aligned} \llbracket p \rrbracket &: v \mapsto v(p) \\ \llbracket \psi \wedge \phi \rrbracket &: v \mapsto \min(\llbracket \psi \rrbracket(v), \llbracket \phi \rrbracket(v)) \\ \llbracket \psi \vee \phi \rrbracket &: v \mapsto \max(\llbracket \psi \rrbracket(v), \llbracket \phi \rrbracket(v)) \\ \llbracket \psi \rightarrow \phi \rrbracket &: v \mapsto \llbracket \neg \psi \vee \phi \rrbracket(v) \\ \llbracket \neg \psi \rrbracket &: v \mapsto (\llbracket \psi \rrbracket(v) + 1) \\ \llbracket \perp \rrbracket &: v \mapsto 0 \\ \llbracket \top \rrbracket &: v \mapsto 1 \end{aligned}$$

With these formulas we can calculate the truth value of any proposition, if we know the valuation function of the variables. We can also decide if a proposition is a tautology, a proposition that is true for every valuation or whether it is satisfiable, meaning that there is at least one valuation for which the proposition is true. Both can be calculated by trying every possible valuation. Because every variable has 2 possible values, a proposition with  $n$  variables has  $2^n$  possible valuations. This can be seen from the domain of  $v$ , being  $\mathbb{B}^{\text{Var}}$ .

If one uses the subset  $V$  of  $\text{Var}$  which contains only the variables used in the proposition, the number of functions is equal to  $|\mathbb{B}|^{|V|}$ , which is  $2^n$  with  $n = |V|$ .

If  $\llbracket \psi \rrbracket(v)$  is true, we write  $v \models \psi$ . If  $v \models \psi$  for all  $v$ , i.e. it is a tautology, we write  $\models \psi$ .

**Example 1.1.**  $p$ : it is raining

We can conclude that  $p \vee \neg p$  is true without knowing anything about the current weather. The possible valuations are  $p$  is true and  $p$  is false, in both cases  $p \vee \neg p$  is true. The proposition does not contain any information about the weather.

In intuitionistic logic a formula is only valid if we can construct a proof for it.\* A proof of the conjunction  $A \wedge B$  is given by a pair of proofs  $(a, b)$  such that  $a$  proves  $A$  and  $b$  proves  $B$ . The disjunction differs even more from classical logic. A proof of the disjunction  $A \vee B$  is a pair of which the first element contains the information which of the disjuncts is correct and the second element gives the proof of the correct disjunct. For example the first element of the proof  $(p, q)$  is given by a number  $p \in \{0, 1\}$  such that if  $p = 0$  then  $q$  is a proof of  $A$  and if  $p = 1$  then  $q$  is a proof of  $B$ .

A proof  $p$  of an implication  $A \rightarrow B$  is a construction that takes a proof  $a$  of  $A$  and returns a proof  $p(a)$  of  $B$ .

**Example 1.2.**

$A$ : “There exist at least twenty twin primes above  $65516468355 \cdot 2^{333333} \pm 1$ ”

$B$ : “There exist at least nineteen twin primes above  $65516468355 \cdot 2^{333333} \pm 1$ ”

If we can prove  $A$  we have also a proof of  $B$  in a very simple way. A proof of twenty of more twin primes above a certain number is also the proof of at least nineteen twin primes above that number. At the moment we have no idea of the truth of both  $A$  and  $B$  and we can certainly not prove it.† But we know that the implication  $A \rightarrow B$  is intuitionistically valid. In classical logic  $A \rightarrow B$  is valid if and only if  $\neg A \vee B$  and we do not know whether  $A$  and  $B$  are true or false. Both in classical and intuitionistic logic it is possible to rewrite  $A \rightarrow B$  to  $\neg B \rightarrow \neg A$ . A proof of  $\neg B$  in the example is one that proves there are eighteen or less twin primes above  $65516468355 \cdot 2^{333333} \pm 1$ . Any proof of nineteen or more twin primes results in a contradiction. This proof of  $\neg B$  also is in contradiction with any proof of twenty or more twin primes and is therefore also a proof of  $\neg A$ .

A proof of a negation  $\neg A$  is a proof that transforms every proof of  $A$  in a contradiction. This may also be written as a special case of the implication:  $p : A \rightarrow \perp$ . In both writings proof  $p$  applied to  $a : A$  gives us  $p(a) : \perp$ . Now we can give an intuitionistic interpretation of  $p \vee \neg p$ : “we can either prove  $p$  or every proof for  $p$  gives us a contradiction and we know which one is the case.”

## 1.2 Natural deduction

In the previous section we used the semantics of the connectives to decide the validity of propositions. Another way to decide whether a proposition is a

\*From here on most of the information about intuitionistic logic is based on [6] and [7].

†The biggest known twin prime at the moment of writing, see [1]

tautology or not is proving it in a deductive system. Below are the rules for propositional logic which we will use in this thesis.<sup>‡</sup> For formulas we use the capital latin letters, preferably  $A, B, C$ . These can stand for any proposition in  $\mathcal{L}_p$ .

**Definition 1.2.**  $\Gamma \vdash \psi$  if there is a deduction for  $\psi$  with only assumptions from  $\Gamma$ . A proposition  $\psi$  is valid if  $\vdash \psi$ , meaning that it can be deduced without assumptions.

$\Gamma$  is a finite set of assumptions. The base case is  $\Gamma, A \vdash A$ .

$$\begin{array}{c}
\frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \wedge B} \wedge I \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge E_l \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge E_r \\
\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee I_l \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee I_r \qquad \frac{\Gamma \vdash A \vee B \quad \Delta, A \vdash C \quad \Lambda, B \vdash C}{\Gamma, \Delta, \Lambda \vdash C} \vee E \\
\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \rightarrow I \qquad \frac{\Gamma \vdash A \rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} \rightarrow E \\
\frac{\Gamma \vdash \perp}{\Gamma \vdash A} \perp E \\
\frac{\Gamma, A \vdash \perp}{\Gamma \vdash \neg A} \neg I \qquad \frac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash \perp} \neg E
\end{array}$$

For deductions of propositions without a set of assumptions we use derived rules without context. We do not write the assumptions down every step, but we prove the conclusion under these assumptions. Assumptions will be marked with a number and the retraction will be marked with the same number next to name of the rule. The base case is  $[A]$ , a marked assumption.

---

<sup>‡</sup>There are rules for first order logic, see [6, page 7].

$$\begin{array}{c}
\frac{A \quad B}{A \wedge B} \wedge I \qquad \frac{A \wedge B}{A} \wedge E_l \quad \frac{A \wedge B}{B} \wedge E_r \\
\\
\frac{A}{A \vee B} \vee I_l \quad \frac{B}{A \vee B} \vee I_r \qquad \frac{\begin{array}{c} [A] \quad [B] \\ \mathcal{D} \quad \mathcal{D}' \\ A \vee B \quad C \quad C \end{array}}{C} \vee E \\
\\
\frac{\begin{array}{c} [A] \\ \mathcal{D} \\ B \end{array}}{A \rightarrow B} \rightarrow I \qquad \frac{A \rightarrow B \quad A}{B} \rightarrow E \\
\\
\frac{\perp}{A} \perp E \\
\\
\frac{\begin{array}{c} [A] \\ \mathcal{D} \\ \perp \end{array}}{\neg A} \neg I \qquad \frac{\neg A \quad A}{\perp} \neg E
\end{array}$$

**Example 1.3.** If we have a formula that requires a pair  $A \wedge B$  as input and we have one half of this pair ( $A$ ) we can construct a new formula  $B \rightarrow C$  that only requires the second half of the pair ( $B$ ) as input. This is known as Currying.<sup>§</sup>

$$\frac{\frac{\frac{{}^1(A \wedge B) \rightarrow C \quad \frac{{}^2A \quad {}^3B}{A \wedge B} \wedge I}{C} \rightarrow E}{B \rightarrow C} \rightarrow I, 3}{A \rightarrow (B \rightarrow C)} \rightarrow I, 2}{((A \wedge B) \rightarrow C) \rightarrow (A \rightarrow (B \rightarrow C))} \rightarrow I, 1$$

**Example 1.4.** If we can deduce  $C$  from  $A$  and  $B$  and we can deduce  $B$  from  $A$  we can also deduce  $C$  from  $A$ . From the previous rule we know that the result is equivalent to  $((A \wedge B) \rightarrow C) \wedge (A \rightarrow B) \wedge A \rightarrow C$ , a formula of which it might be easier to see that it is true.

$$\frac{\frac{\frac{{}^1A \rightarrow (B \rightarrow C) \quad {}^2A}{B \rightarrow C} \rightarrow E \quad \frac{{}^3A \rightarrow B \quad {}^2A}{B} \rightarrow E}{C} \rightarrow I, 2}{A \rightarrow C} \rightarrow I, 3}{(A \rightarrow B) \rightarrow (A \rightarrow C)} \rightarrow I, 1}{(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))} \rightarrow I, 1$$

**Example 1.5.** A distribution of a conjunction over disjunctions is intuitionistically valid.

<sup>§</sup>After the mathematician Haskell Curry. He reinvented the process invented by Moses Schönfinkel. *Schönfinkeling* did not really catch on.



$$\frac{\frac{A \wedge (B \vee C)}{B \vee C} \wedge E_r \quad \frac{\frac{\frac{{}^1A \wedge (B \vee C)}{A} \wedge E_l \quad {}^2B}{A \wedge B} \wedge I \quad (A \wedge B) \vee (A \wedge C) \vee I_l}{(A \wedge B) \vee (A \wedge C)} \vee I_l \quad \frac{\frac{\frac{{}^1A \wedge (B \vee C)}{A} \wedge E_l \quad {}^3C}{A \wedge C} \wedge I \quad (A \wedge B) \vee (A \wedge C) \vee I_r}{(A \wedge B) \vee (A \wedge C)} \vee I_r}{(A \wedge B) \vee (A \wedge C)} \vee E2,3}{A \wedge (B \vee C) \rightarrow (A \wedge B) \vee (A \wedge C)} \rightarrow I,1$$

**Example 1.6.** If we can deduce something from either  $A$  or  $B$ , it has to be deducible from both  $A$  and  $B$ . We do not know beforehand which of them will be the case, so both have to imply  $C$ .

$$\frac{\frac{{}^1(A \vee B) \rightarrow C \quad \frac{{}^2A}{A \vee B} \vee I_l}{C} \rightarrow I,2 \quad \frac{{}^1(A \vee B) \rightarrow C \quad \frac{{}^3B}{A \vee B} \vee I_r}{C} \rightarrow I,3}{(A \rightarrow C) \wedge (B \rightarrow C) \wedge I}{((A \vee B) \rightarrow C) \rightarrow ((A \rightarrow C) \wedge (B \rightarrow C))} \rightarrow I,1$$

### 1.3 Kripke Models

In intuitionistic logic it is not possible to make truth tables, because not everything classically valid is also intuitionistic valid. Instead we can use Kripke models. Kripke models are a useful way to prove that a proposition is false. By constructing models one can show that a proposition is not true. Infinitely many frames and models can be constructed, so it is not always easy to find the model that shows the contradiction.

Kripke frames for intuitionistic logic are sets of nodes  $K$  with a partial order  $\leq$  on these nodes.<sup>¶</sup> A model is a frame with variables assigned to nodes  $k \in K$ . If a variable  $p$  is true in node  $k$  we write  $k \Vdash p$  ( $k$  forces  $\phi$ ). One can see this model as a time model with possible futures. The bottom node, in this text indicated as  $\mathbf{0}$ , represents our current knowledge. Accessible nodes represent possible future situations with the knowledge we have at that moment. Once we have accepted knowledge in a world, it holds for all accessible future nodes.

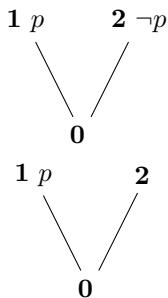
Figure 1.1: Countermodel for  $p \vee \neg p$



An example is the frame  $\langle K, \leq \rangle$  with  $K = \{0, 1\}$  and  $0 \leq 1$ . We do not know anything now, but in the future we might know  $p$ . We write this as  $1 \Vdash p$ . Now we know  $0 \not\Vdash \neg p$  because  $1 \Vdash p$  and if  $0 \Vdash \neg p$  would hold, we would have

<sup>¶</sup>A partial order is a reflexive, transitive and anti-symmetric relation. The  $\leq$  relation on natural numbers is a partial order.

a contradiction in 1. If we might know  $p$  in the future, we can not know  $\neg p$  now. Both  $p$  and  $\neg p$  are not the case in 0, later we will see that we can conclude  $0 \not\vdash p \vee \neg p$  from this information and that we have a countermodel for  $p \vee \neg p$ . This model is not the only possible situation. In the future we also might know  $\neg p$ .

Figure 1.2: Countermodel for  $\neg p \vee \neg\neg p$ 

In the model in figure 1.2 we have a countermodel for  $\neg p \vee \neg\neg p$ . We add another node to the model in figure 1.1, being node 2 for which  $\neg p$  holds. If in world 0  $\neg\neg p$  would hold, we would have a contradiction in world 2, therefore  $0 \not\vdash \neg\neg p$ . Because of  $1 \Vdash p$  we already know  $0 \not\vdash \neg p$  and therefore  $0 \not\vdash \neg p \vee \neg\neg p$ .

However, we do not actually assign propositions to nodes, but only variables. Therefore we do not assign  $\neg p$  to node 2, but we do simply not assign  $p$  which means that  $2 \not\vdash p$ .

**Definition 1.3.** A Kripke model is a triple  $\langle K, \leq, \Vdash \rangle$  with a set of nodes  $K$ , a partial ordering  $\leq$  on these nodes and a binary relation  $\Vdash$  on  $K \times \text{Var}$ . This relation is such that if  $\forall k, k' k \Vdash p$  and  $k' \geq k$  then  $k' \Vdash p$ .

We can extend the definition of Kripke models to sentences.

$$\begin{aligned} k \Vdash \phi \wedge \psi &:= k \Vdash \phi \text{ and } k \Vdash \psi. \\ k \Vdash \phi \vee \psi &:= k \Vdash \phi \text{ or } k \Vdash \psi. \\ k \Vdash \phi \rightarrow \psi &:= \text{for all } l \geq k \text{ if } l \Vdash \phi \text{ then also } l \Vdash \psi. \\ k \Vdash \neg \phi &:= k \Vdash \phi \rightarrow \perp \\ k \not\vdash \perp & \end{aligned}$$

**Definition 1.4.** A proposition is valid on a model if it is valid on all nodes in that model.  $K \Vdash \phi$  if and only if  $\forall k \in K k \Vdash \phi$

**Definition 1.5.** A proposition is valid if it is true for all models.  $\Vdash \phi$  if and only if  $\forall K, K \Vdash \phi$ .

**Lemma 1.1** (Preservation of truth). If  $k \Vdash \psi$  and  $k' \geq k$  then also  $k' \Vdash \psi$ .

*Proof.* We have  $k$  and  $k'$  such that  $k \leq k'$  and  $k \Vdash \psi \wedge \phi$ . Now  $k \Vdash \psi$  and  $k \Vdash \phi$ . From the definition of Kripke models we know  $k' \Vdash \psi$  and  $k' \Vdash \phi$ . Therefore also  $k' \Vdash \psi \wedge \phi$ .

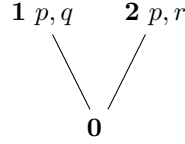
We have  $k$  and  $k'$  such that  $k \leq k'$  and  $k \Vdash \psi \vee \phi$ . This is valid if  $k \Vdash \psi$  or  $k \Vdash \phi$ . In the case  $k \Vdash \psi$  also  $k' \Vdash \psi$  and therefore  $k' \Vdash \psi \vee \phi$ . If  $k \not\vdash \psi$  then  $k \Vdash \phi$  is the case. Then  $k' \Vdash \phi$  and  $k' \Vdash \psi \wedge \phi$ .

The definition of the implication and negation is such that it is easy to see that it preserves truth. Note that  $\leq$  is transitive, i.e.  $k \leq k'$  and  $k' \leq k''$  also means  $k \leq k''$ .  $\square$

Another example is that of  $((p \rightarrow (q \vee r)) \rightarrow ((p \rightarrow q) \vee (p \rightarrow r)))$  in figure 1.3. If this were true, all worlds  $w \Vdash p$  would either all force  $q$  or all force  $r$ . We take a model  $K \Vdash p \rightarrow (q \vee r)$  and show  $\exists k \in K \ k \not\Vdash (p \rightarrow q) \vee (p \rightarrow r)$  and therefore  $K \not\Vdash ((p \rightarrow (q \vee r)) \rightarrow ((p \rightarrow q) \vee (p \rightarrow r)))$ . Because we have found this countermodel we also know  $\not\Vdash ((p \rightarrow (q \vee r)) \rightarrow ((p \rightarrow q) \vee (p \rightarrow r)))$ .

Figure 1.3 shows a countermodel. The valuation is such that  $0 \Vdash p \rightarrow (q \vee r)$ . Because  $1 \Vdash p$  and  $1 \not\Vdash r$ , we know  $0 \not\Vdash p \rightarrow r$ . Similarly we can see  $0 \not\Vdash p \rightarrow q$  because of the valuation in 2. Both sides of the disjunction are not valid and therefore the disjunction is not valid either.  $0 \not\Vdash (p \rightarrow q) \vee (p \rightarrow r)$ .

Figure 1.3: Countermodel for  $((p \rightarrow (q \vee r)) \rightarrow ((p \rightarrow q) \vee (p \rightarrow r)))$



In intuitionistic logic  $A \rightarrow \neg\neg A$  is valid as in classical logic. We can show this with natural deduction. Later we will see that Kripke models are sound and therefore if we can deduce something with natural deduction it also holds in all Kripke models.

*Proof.*

$$\frac{\frac{\frac{1 \neg A \quad 2 A}{\perp} \neg E}{\neg\neg A} \neg I, 1}{A \rightarrow \neg\neg A} \rightarrow I, 2$$

$\square$

The other way around,  $\neg\neg A \rightarrow A$  is not valid for all  $A$ . This  $A$  is a metavariable that can be any sentence in  $\mathcal{L}_p$  and when choosing  $A = p$  we can make a countermodel. We show that there is a model  $K$  with a node  $k \in K$  such that  $k \Vdash \neg\neg p$  but also  $k \not\Vdash p$ . Take the frame with worlds 0 and 1 with  $0 \leq 1$  and  $1 \Vdash p$ . Now  $0 \Vdash \neg\neg p$  but not  $0 \Vdash p$ . There are two possibilities for a world  $l \geq 0$ , being  $l = 0$  such that  $0 = l$  and  $l = 1$  such that  $0 < l$ . In the first case there is an accessible world, being 1, in which  $p$  is true. In the second case there is also an accessible world, again being 1, in which  $p$  is true. Therefore  $0 \Vdash \neg\neg p$  and  $0 \not\Vdash \neg\neg p \rightarrow p$ .

## 1.4 Soundness of Kripke models

Kripke models are sound if everything that can be proven by natural deduction also holds on Kripke models. We will show this with an induction on the structure of the deduction.

Figure 1.4: Countermodel for  $\neg\neg p \rightarrow p$ 

**Theorem 1.1** (Soundness of Kripke models). If  $\Gamma \vdash \psi$  then  $\Gamma \Vdash \psi$ .

**Definition 1.6.**  $\Gamma \Vdash \psi$  if and only if in each model  $K$  and each node  $k \in K$ , if  $k \Vdash \gamma$  for all  $\gamma \in \Gamma$  then also  $k \Vdash \psi$ .

*Proof.* Let  $\Gamma$  a set of formula's and  $\psi$  a formula. Suppose  $\Gamma \vdash \psi$ . We proceed by using the structure of the proof of  $\psi$ .

We distinguish several cases. The base case is when  $\psi$  is in  $\Gamma$ . Suppose that  $\Gamma = \Delta, \psi$ . For all Kripke models  $K$  and nodes  $k \in K$  where  $k \Vdash \gamma$  for all  $\gamma \in \Gamma$ . Because  $\psi \in \Gamma$  we know  $k \Vdash \psi$ . Therefore  $\Gamma \Vdash \psi$ .

*Case  $\psi = \phi \wedge \chi$ ,  $\wedge$  introduction:* Suppose  $\Gamma \vdash \phi \wedge \chi$ . From the deduction we know  $\Gamma \vdash \phi$  and  $\Gamma \vdash \chi$ . With induction we know  $\Gamma \Vdash \phi$  and  $\Gamma \Vdash \chi$ . For an arbitrary model  $K$  and an arbitrary node  $k$  we know that if  $k \Vdash \phi$  and  $k \Vdash \chi$  then also  $k \Vdash \phi \wedge \chi$ . From this it follows that  $\Gamma \Vdash \phi \wedge \chi$ .

*Case  $\psi = \alpha \vee \beta \rightarrow \gamma$ ,  $\vee$  elimination:* Suppose  $\Gamma, \Delta, \Lambda \vdash \alpha \vee \beta \rightarrow \gamma$ . Let  $K$  be a model. Suppose for every  $k$  in the model  $k \Vdash \Gamma, \Delta, \Lambda$ . In particular  $k \Vdash \Gamma$  and consequently  $k \Vdash \alpha \vee \beta$ . We distinguish two situations. In the first  $k \Vdash \alpha$  and from  $k \Vdash \Delta, \alpha$  it follows that  $k \Vdash \gamma$ . In the other situation  $k \Vdash \beta$  and  $k \Vdash \Lambda, \beta$  and hence  $k \Vdash \gamma$ . Consequently  $\Gamma, \Delta, \Lambda \Vdash \alpha \vee \beta \rightarrow \gamma$ .

*Case  $\alpha \rightarrow \beta$ ,  $\rightarrow$  elimination:* Suppose  $\Gamma \vdash \alpha \rightarrow \beta$  and consequently  $\Gamma, \alpha \vdash \beta$ . Let  $K$  be a model. Suppose for every  $k$  in the model  $k \Vdash \gamma$  for all  $\gamma \in \Gamma$  and thus  $k \Vdash \alpha \rightarrow \beta$ . For every  $k \Vdash \Gamma, \alpha$  also  $k \Vdash \beta$ , a consequence of the definition of  $k \Vdash \alpha \rightarrow \beta$ . Concluding  $\Gamma \Vdash \alpha \rightarrow \beta$  and  $\Gamma, \alpha \Vdash \beta$ .

The proof of the soundness for Kripke models for the other deduction rules are quite similar. If we use the empty set  $\emptyset$  for the context we know that if  $\vdash \psi$  also  $\Vdash \psi$ .  $\square$

## Chapter 2

# Decision problems and Complexity

In logic we often want to know whether a proposition is true or false. This is a decision problem. In decision problems there is a question that has to be answered about an input. The question has only two possible answers, mostly yes or no.

**Example 2.1.** Is this proposition for this valuation true?

Input:  $p \vee q$  with  $p = \top$  and  $q = \perp$

Answer: Yes

If there is an algorithm to answer such a question, the problem is decidable. Some problems are not decidable because there is no algorithm that gives a yes or a no.

**Example 2.2.**  $n$  and  $m$  is the biggest pair of twin primes

“On input  $\langle n, m \rangle$  with  $n, m$  a pair of twin primes:

1. Try to find a bigger twin prime pair. If you find this, *reject*.
2. Otherwise, *accept*

The problem with this algorithm is that there is no way to decide whether you are not going to find a bigger twin prime pair. You only know whether you have found one or you have not found one yet. If we run this algorithm on  $\langle 65516468355 \cdot 2^{333333} - 1, 65516468355 \cdot 2^{333333} + 1 \rangle$  it is possible that the algorithm never ends. If there is no bigger pair of twin primes the algorithm will never get out of step 1 and keep looking for a bigger pair.

At the moment we can not decide this problem, but it is not necessarily impossible. We simply do not have a decision procedure. There are also problems which will always be undecidable. A decision algorithm for those problems will always lead to a contradiction. Not just because the algorithm is not good enough and we have to find a better one, but because we can find an instance of the problem that cannot be solved by any algorithm that decides this kind of problems. An example  $A_{tm}$  which contains the encodings  $\langle T, w \rangle$  of pairs of a Turing machine  $T$  and an input  $w$  for which  $T$  accepts when running on  $w$ .

**Example 2.3** (Example from [4]). We have a hypothetical Turing machine  $H$  that decides whether a Turing machine  $M$  accepts on input  $w$ . We construct Turing machine  $M$ .

$M =$  “On input  $\langle N \rangle$ :

1. Run  $H$  on  $\langle N, \langle H \rangle \rangle$ .
2. If  $H$  rejects, *accept*. Otherwise, *reject*

Now run  $M$  on  $\langle M \rangle$ . This accepts if  $H$  rejects on  $\langle M, \langle M \rangle \rangle$  and rejects if  $H$  accepts  $\langle M, \langle M \rangle \rangle$ . Machine  $M$  always does the opposite of what  $H$  says it does. Therefore we know that  $H$  cannot decide this problem.

## 2.1 Complexity

Not every decidable problem is equally hard to solve. Complexity theory investigates the amount of time or memory that is used to solve a problem.\*

**Example 2.4.** String  $s$  does contain at least one  $a$ .

**Algorithm** “On input  $s$ :

1. Start with the first character of  $s$ .
2. Check the character. If this is an  $a$ , *accept*.
3. If there is a next character, do step 2 for the next character.
4. Otherwise, *reject* ”

This is an algorithm that always halts. It quits when an  $a$  is encountered and if no  $a$  is encountered, the algorithm halts too, because it returns “no” when the end of the string is reached. If you run this algorithm on a number of strings with the same length, for example strings with 10 characters, the run time of the algorithm will not be equal for every string. The string that starts with an  $a$  will be decided really quick, the algorithm stops at the first step. If the string contains no  $a$ 's, step 2 and 3 will be repeated for every character of the string and after that the algorithm will reject. But these steps will only be repeated for as many times as there are characters in the string. The algorithm starts at one side of the string and only moves to the other end and never goes back to a character already tested.

In complexity analysis we are interested in the worst case, which is the longest possible running time for a string of certain length. But the exact running time of a algorithm is often a complex expression and more precise than useful. An algorithm that runs in  $n + 100$  time is for small  $n$ 's slower than one in  $n^2$ , but for large inputs it will definitely be faster. We are interested in the behavior of the running time for large inputs. For large  $n$ 's the biggest polynomial of the function is the most important. Therefore we use an estimate of the function called the big-O notation.

**Definition 2.1.** Let  $f, g$  be functions  $\mathbb{N} \mapsto \mathbb{R}_{>0}$ .  $f(n) = O(g(n))$  if  $c, n_0 \in \mathbb{N}_{>0}$  exist such that for every  $n \geq n_0$

$$f(n) \leq cg(n)$$

With this definition we can conclude  $n^{100} \neq O(n^{99})$  because no matter how big we choose  $c$ , from a certain  $n$  (being  $n = c$ )  $n^{100}$  is always bigger than  $cn^{99}$ . The other way around  $n^{99} = O(n^{100})$  because  $n^{100}$  grows faster than  $n^{99}$ . Choosing  $n^{100}$  as an estimation for  $n^{99}$  is not a very useful thing to do.

---

\*Most information in this chapter is from [4].

Algorithms with running times  $n^2$  and  $n^5$  are both in  $O(n^5)$ , but the first is definitely a faster algorithm. Therefore it is common to pick the smallest possible function for  $g(n)$ . The big-O notation is transitive.

**Lemma 2.1** (Big-O is transitive). If  $f(n)$  is in  $O(g(n))$  and  $g(n)$  is in  $O(h(n))$  then  $f(n)$  is also in  $O(h(n))$ .

*Proof.* Let  $f(n) \leq c_1 \cdot g(n)$  for all  $n \geq n_1$ . Let  $g(n) \leq c_2 \cdot h(n)$  for all  $n \geq n_2$ . If we multiply both sides of a  $\leq$  relation with the same positive constant, the relation still holds. Therefore  $c_1 \cdot g(n) \leq c_1 \cdot c_2 \cdot h(n)$  for all  $n \geq n_2$ . The  $\leq$  relation is transitive. From that we can see  $f(n) \leq c_1 \cdot c_2 \cdot h(n)$ . This holds for all  $n \geq \max(n_1, n_2)$ .  $\square$

## 2.2 Complexity classes

The problem in example 2.4 is a linear time problem. There is a linear relation between the length of the input and the time needed to calculate the answer. Space complexity is expressed as a function of  $n$  (with  $n$  the length of the input) that tells us the number of tape spaces that a Turing machine needs to use to decide the problem in the worst case. When using pseudo-code algorithms the space complexity is expressed in used memory in terms of the size of the input.

**Definition 2.2.**  $\text{TIME}(f(n))$  is the class of problems decidable in  $O(f(n))$  time on a deterministic Turing machine.

$\text{NTIME}(f(n))$  is the class of problems decidable in  $O(f(n))$  time on a non-deterministic Turing machine.

**Definition 2.3.** Class P is the class of all functions decidable on a deterministic Turing machine in polynomial time.

$$P = \bigcup_k \text{TIME}(n^k).$$

**Definition 2.4.** Class NP is the class of all functions decidable on a non-deterministic Turing machine in polynomial time.

$$NP = \bigcup_k \text{NTIME}(n^k).$$

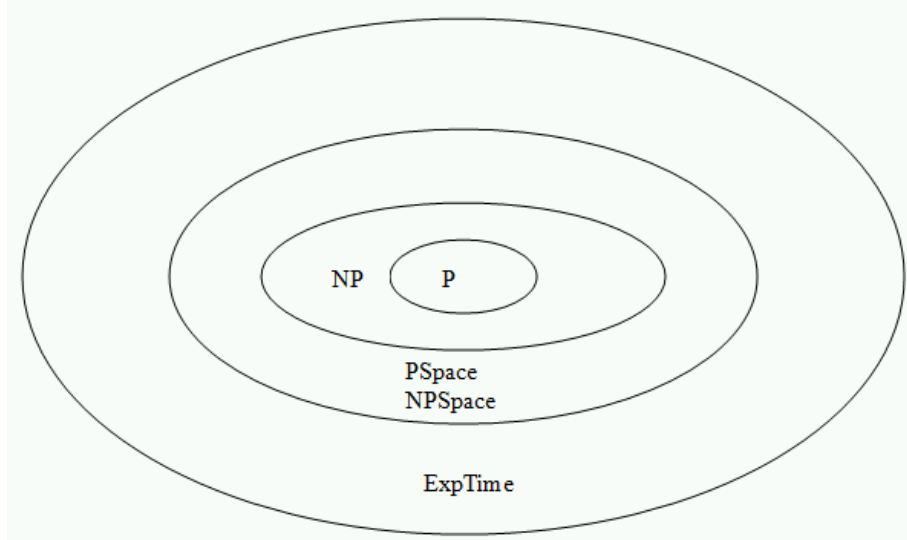
**Definition 2.5.**  $\text{EXPTIME} = \bigcup_c \text{TIME}(c^n)$

The classes for complexity are related. If a problem is in P because we can solve it in polynomial time on a deterministic Turing machine, we also can solve it in polynomial time on a non-deterministic Turing Machine. Every problem in P is also in NP. From this we can conclude  $P \subseteq NP$ .<sup>†</sup>

Besides time complexity there also is space complexity. The space complexity is the maximum number of cells a Turing Machine scans when calculating the output. This is expressed by a function  $f(n)$  with  $n$  the length of the input.

**Definition 2.6.**  $\text{SPACE}(f(n))$  = the class of problems decidable in  $O(f(n))$  space on a deterministic Turing Machine.

<sup>†</sup>There are two possibilities of this situation, being  $P = NP$  or  $P \subset NP$ . At the moment there is no proof for either of the situations. It is one of the major open problems in mathematics.

Figure 2.1: The complexity classes with  $P \neq NP$ 

As with time, there is the class NSPACE for the problems that are decidable on non-deterministic Turing Machines. Savitch's theorem tells us that  $NSPACE(f(n)) \subseteq SPACE(f^2(n))$ .<sup>‡</sup>

**Definition 2.7.**  $PSPACE = \bigcup_k SPACE(n^k)$

We could define NPSPACE the same way with PSPACE but this is useless. A polynomial squared is still a polynomial and therefore  $PSPACE = NPSPACE$ . In relation to the time complexity classes  $NP \subseteq PSPACE$ , because in every time step there can be set one place step, so in  $n^k$  time the maximal number of steps in one way that can be taken are  $n^k$ .

## 2.3 Reducibility

A reduction is a way to use the solution of one problem to solve another. For example, you're at home and you want to know whether the supermarket is open. You can go there and see for yourself. But you can also look up the opening hours and find out what time and day it is now, for example on the internet. If the supermarket is far away the last solution is a quicker one, only if you have access to the internet at home, which is most likely. But if you only have access to the internet at the library next to the supermarket, this solution is not really useful anymore.

**Example 2.5.** Solving CLIQUE with HALF-CLIQUE

A clique is a subgraph wherein every node is connected to every other node in the clique. CLIQUE is the language of all  $\langle G, k \rangle$  with  $G$  a graph with a clique containing  $k$  nodes. HALF-CLIQUE is the language of encodings of graphs  $\langle G \rangle$

<sup>‡</sup>The proof for this is quite long, see for example [4, page 309]



with a clique containing half of the total number of nodes

Solving HALF-CLIQUE when you know how to solve CLIQUE is not that hard. You count the nodes, divide this number by two and see if there is a clique this size by running CLIQUE on  $\langle G, n/2 \rangle$ . But it is also possible to solve CLIQUE with HALF-CLIQUE using the following Turing machine:

M= "On input  $\langle G, k \rangle$

1. Count the number of nodes, call this  $n$
2. If  $k = 2n$  run HALF-CLIQUE on  $\langle G \rangle$
3. If  $k < 2n$  add  $m - 2k$  fully connected nodes and run HALF-CLIQUE on  $\langle G \rangle$
4. If  $k > 2n$  add  $2k - m$  unconnected nodes and run HALF-CLIQUE on  $\langle G \rangle$
5. If any of these steps accepts, *accept*. Otherwise, *reject*.

The constraint that a reduction has to take place in polynomial time is to make sure the amount of time to reduce the problem to a different one and to solve the new problem combined is not in a different class.

**Definition 2.8.** A problem  $p$  is complete for class  $X$  if:

1.  $p$  is in class  $X$
2. all problems in class  $X$  are reducible in polynomial time to  $p$ .

The first part can be proved by an algorithm that solves the problem in the time or space corresponding to class  $X$ . I.e. if we have an algorithm that solves problem  $p$  in  $\text{NTIME}(n^k)$  for some  $k \geq 0$  then  $p$  is in NP. A problem is  $X$ -hard if it is at least as hard as all other  $X$ -complete problems, which means that we can reduce every known  $X$ -complete problem to the new problem. Luckily for us this is a transitive relation so if we prove that  $p$  is at least as hard as one  $X$ -complete problem, it is at least as hard as all other problems in  $X$ . For example if we can solve a problem by reducing it to CLIQUE, we can also solve it by reducing it to HALF-CLIQUE, for example by reducing the reduction again. If we know that CLIQUE is NP-complete (which is the case) we now know that HALF-CLIQUE is NP-complete because we can reduce CLIQUE to HALF-CLIQUE.

A reduction has to be in polynomial time. This way, the solution of a problem in NP by reducing it and solving the reduced problem is still in NP. If the reduction would be in exponential time the solution is no longer guaranteed to be in NP.

## 2.4 NP: SAT and TAUT

The satisfiability problem is an example of an NP-complete problem. A boolean formula is satisfiable if there is at least one valuation of the variables that makes the formula true.

**Example 2.6.**  $p \vee q$

This is a satisfiable formula. There are three valuations that make the formula true, being only  $p$  is true, only  $q$  is true or  $p$  and  $q$  are both true.

**Example 2.7.**  $p \wedge \neg p$

This formula is not satisfiable. There is one variable and whether this is true or false, the conjunction is false.

A slightly different but related problem is that of the tautology. A boolean formula is a tautology if all possible valuations make the formula true. Example 2.6 is not a tautology because there is one valuation that makes the formula false, being  $p$  and  $q$  both false.

SAT is the set of all encodings of satisfiable formulas and TAUT is the set of all encodings of tautologies.  $\text{TAUT} \subset \text{SAT}$  because if a formula is true for all valuations, there is also at least one valuation that makes the formula true.

## 2.5 PSPACE: INTAUT and TQBF

INTAUT is the set of encodings of tautologies in intuitionistic logic. This can be solved via the Kripke models. Ladner [3] gives an algorithm that solves S4 models for modal logic, models with partial order frames, in PSPACE.

SAT and TAUT are sets of formula's in propositional logic. Formula's in first order logic, predicate calculus, are harder to calculate. In predicate logic we use the same symbols as in propositional logic, extended with the quantifiers  $\exists$  and  $\forall$ . The existential quantifier  $\exists$  has the meaning that  $\exists x \phi$  is valid if there is a value for  $x$  such that  $\phi$  is valid. The universal quantifier  $\forall$  has the meaning that  $\forall x \phi$  is valid if  $\phi$  is valid for all values of  $x$ .

$$\begin{aligned} \llbracket \exists x \phi \rrbracket &: v \mapsto \max(\llbracket \phi \rrbracket v \langle x \mapsto 0 \rangle, \llbracket \phi \rrbracket v \langle x \mapsto 1 \rangle) \\ \llbracket \forall x \phi \rrbracket &: v \mapsto \min(\llbracket \phi \rrbracket v \langle x \mapsto 0 \rangle, \llbracket \phi \rrbracket v \langle x \mapsto 1 \rangle) \end{aligned}$$

Here  $v \langle x \mapsto 0 \rangle$  is a normal valuation  $v$  but with  $x$  false and  $v \langle x \mapsto 1 \rangle$  is the same valuation but with  $x$  true.

The set TQBF is the set of True fully Quantified Boolean Formulas. Fully quantified Boolean formulas are formulas with variables that are Booleans and every variable is in the scope of a quantifier. A formula  $\phi$  can be evaluated and if it is true the encoding of  $\phi$  is in TQBF. Every QBF can be written with all quantifiers at the beginning.

On input  $\langle \phi \rangle$ , with  $\phi$  a fully quantified Boolean formula:

1. If  $\phi$  has no quantifiers, evaluate  $\phi$  and *accept*
2. If  $\phi$  is  $\exists x \psi$ , evaluate  $\psi$  once with  $x$  is true and once with  $x$  is false. If either of these evaluations accepts, *accept*
3. If  $\phi$  is  $\forall x \psi$ , evaluate  $\psi$  once with  $x$  is true and once with  $x$  is false. If both evaluations accept, *accept*

The evaluation of QBF's is in exponential time but in polynomial space. Every quantified variable can be true or false and both cases will be evaluated, making the time exponential. But for every variable only the value has to be stored and this takes only constant place. The space complexity is in  $n^k$ .

The reduction of other problems from PSPACE to TQBF can be made for all problems in PSPACE at once because it is based on the contents of the tape of the evaluations of these problems. The tape space is in  $O(n^k)$ , but the time is polynomial. We can make a list of the states, with on the first line the begin state and on the last line the accepting state. This list is  $O(2^{c(n)})$  long. We are going to construct a quantified Boolean formula that represents the list of states.

We can construct a formula  $\phi_{c_1, c_2, t}$  which is true if we can go from state  $c_1$  to a new configuration  $c_2$  in at most  $t$  steps. The base cases are  $\phi$  is true for  $\phi_{c_1, c_2, 1}$  if and only if  $c_2$  follows from a single transition on  $c_1$ . The other case is  $\phi_{c_1, c_2, 0}$  is true if and only if  $c_1 = c_2$ . For values of  $t$  bigger than 1 we decide the validity of  $\phi$  with a conjunction of steps taken from the first to the last state.

Every decidable problem has a start configuration and an accept configuration on the tape. We want to decide  $\psi_{c_{\text{begin}}, c_{\text{accept}}, t}$  with  $t$  at least as big as the maximal number of possible configurations of the tape. We choose  $t$  a power of 2 because this will make our calculation easier. If we go from  $c_{\text{begin}}$  to  $c_{\text{accept}}$  there is a configuration  $m$  halfway. It is possible to get from  $c_{\text{begin}}$  to  $m$  and we can also get from  $m$  to  $c_{\text{accept}}$ , both in half the time that it takes to get from  $c_{\text{begin}}$  to  $c_{\text{accept}}$ .

$$\psi_{c_1, c_2, t} \text{ if and only if } \exists m \psi_{c_1, m, \frac{t}{2}} \text{ and } \psi_{m, c_2, \frac{t}{2}}$$

We divide the list in halves and decide for both halves whether they are true. This way you split  $t$  in half every step, but you also double the length of the formula. With  $t$  bigger than the maximal number of steps and the problem being in EXPTIME, the formula will be exponentially big. A reduction has to be in polynomial time, so this is not the solution.

We want a formula  $\phi$  which only makes one new  $\phi'$  and not two. We do this by adding a second quantifier.

$$\phi_{c_1, c_2, t} = \exists m \forall (c_3, c_4) \in \{(c_1, m), (m, c_2)\} [\phi_{c_3, c_4, \frac{t}{2}}]$$

This is only valid if there is a  $m$  such that both  $\phi_{c_1, m, \frac{t}{2}}$  and  $\phi_{m, c_2, \frac{t}{2}}$  are valid. The set notation is not a valid notation for quantified Boolean formula's. We can replace this by the notation

$$\forall (c_3, c_4) \exists m [((c_3, c_4) = (c_1, m) \vee (c_3, c_4) = (m, c_2)) \rightarrow \phi_{c_3, c_4, \frac{t}{2}}]^\S$$

If we choose  $t = 2^{cf(n)}$  with  $c$  a constant and  $f(n)$  a polynomial, the size of the formula  $\phi_{c_{\text{begin}}, c_{\text{end}}, t}$  will be in  $O(f(n))$ . A recursive step adds a constant number of configurations to the formula, so this is linear with the length of the input. There will be  $\log(t)$  recursions because every step halves the list of configurations, so this is  $\log(2^{cf(n)}) = O(f(n))$ . Choosing the right  $m$  for each formula is non-deterministic. Savitch's theorem tells us that every non-deterministic Turing machine that solves a problem in  $f(n)$  can be converted to a deterministic Turing machine that solves the problem in  $f^2(n)$ . Because  $f(n)$  is a polynomial function,  $f^2(n)$  will also be polynomial. Therefore we have a reduction in polynomial time and space and is TQBF PSPACE-complete.

---

<sup>§</sup>This is equivalent.  $\forall x \in \{y, z\} Q(x)$  can be rewritten to  $\forall x [(x = y \vee x = z) \rightarrow Q(x)]$ . The second notation requires that the implication is true for all possible values of  $x$  and not only  $x = y$  and  $x = z$ , but if the assumption is false (i.e.  $x \neq y$  and  $x \neq z$ ) the implication is always true. This notation still uses a pair notation which is also not defined for quantified Boolean formula's. This can be rewritten with quantifiers for both  $c_3$  and  $c_4$ , but the pair notation is easier to read.



## Chapter 3

# Bi-implicational fragment

For this part we use the following language:

**Definition 3.1.**  $\mathcal{L}_{\leftrightarrow} = \text{Var} \mid \mathcal{L}_{\leftrightarrow} \leftrightarrow \mathcal{L}_{\leftrightarrow} \mid \top$

We can interpret this as:

$$\llbracket \phi \leftrightarrow \psi \rrbracket : v \mapsto (\llbracket \phi \rrbracket(v) + \llbracket \psi \rrbracket(v) + 1)$$

In classical logic the following rules can be derived for the bi-implicational fragment:

$$\begin{array}{cc} \frac{B \leftrightarrow A}{A \leftrightarrow B} & \frac{A \leftrightarrow A}{\top} \\ \text{symmetry} & \text{self-cancellation} \\ \\ \frac{A \leftrightarrow \top}{A} & \frac{(A \leftrightarrow B) \leftrightarrow C}{A \leftrightarrow (B \leftrightarrow C)} \\ \text{identity} & \text{associativity} \end{array}$$

All these rules can be proven in classical logic with truth tables, see table 3.1

With commutativity and associativity we can easily check the truth of propositions. For example the following deduction:

**Example 3.1.**

$$\begin{array}{l} \frac{(a \leftrightarrow b) \leftrightarrow (b \leftrightarrow a)}{(a \leftrightarrow b) \leftrightarrow (a \leftrightarrow b)} \\ \frac{((a \leftrightarrow b) \leftrightarrow a) \leftrightarrow b}{((a \leftrightarrow (b \leftrightarrow a)) \leftrightarrow b)} \\ \frac{(a \leftrightarrow (a \leftrightarrow b)) \leftrightarrow b}{((a \leftrightarrow a) \leftrightarrow b) \leftrightarrow b} \\ \frac{(a \leftrightarrow a) \leftrightarrow (b \leftrightarrow b)}{\frac{\top \leftrightarrow \top}{\top}} \end{array}$$

The same proposition can be deduced in different ways. For example  $((a \leftrightarrow b) \leftrightarrow a) \leftrightarrow b$  in the third row may be directly rewritten to  $(a \leftrightarrow (a \leftrightarrow b)) \leftrightarrow b$  which makes

Table 3.1: Truth tables for classical deduction rules for  $\mathcal{L}_{\leftrightarrow}$ 

p	$p \leftrightarrow \top$	p
0	0	0
1	1	1

p	$\top$	$p \leftrightarrow p$
0	1	1
1	1	1

p	q	$(p \leftrightarrow q)$	$(q \leftrightarrow p)$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	1	1

p	q	r	$(p \leftrightarrow q) \leftrightarrow r$	$p \leftrightarrow (q \leftrightarrow r)$
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

the deduction one step shorter. Because every rule can be used in both directions, the deduction is also valid in both directions. In this chapter are single lines used between the rows of deductions, but these can be read as double lines.

Because of the associativity the parentheses in a proposition can be moved to any position. We can conclude that the parentheses in a classical bi-implication formula contain no information.

**Definition 3.2** (Equivalence). Formulas  $\psi$  and  $\phi$  are equivalent if  $\Vdash \psi \leftrightarrow \phi$ . This relation is reflexive, transitive and symmetric.

**Lemma 3.1.** For every  $\phi$  there is a  $\psi$  such that  $\phi \leftrightarrow \psi$  and in  $\psi$  all parentheses are in the rightmost position

*Proof.* If we only use the associativity rule that writes the parentheses to the right we will end with a formula with the parentheses in the rightmost position.  $(A \leftrightarrow B) \leftrightarrow C \rightsquigarrow A \leftrightarrow (B \leftrightarrow C)$ . Not all formula's are in this form, so we add the rules that if  $A \rightsquigarrow B$  then also  $(A \leftrightarrow C) \rightsquigarrow (B \leftrightarrow C)$  and  $(C \leftrightarrow A) \rightsquigarrow (C \leftrightarrow B)$ . For every formula  $A$  there is a reflexive transitive closure  $\rightsquigarrow^*$  such that  $A \rightsquigarrow^* A'$ .

This  $A'$  is unique for every  $A$ , no  $A$  has two rightmost notations. If there would be two rightmost notations,  $A \rightsquigarrow^* B, B'$  then  $B$  and  $B'$  would be equivalent save the brackets. In that case there is a  $C$  such that  $B, B' \rightsquigarrow C$ . But  $B$  and  $B'$  are obtained by a reflexive transitive closure, so  $C$  would have to be reached by zero steps. Therefore  $B$  and  $B'$  are the same.  $\square$

Multiple  $\phi$ 's may have the same  $\psi$  and are therefore equivalent. If  $\phi \leftrightarrow \psi$  and  $\phi' \leftrightarrow \psi$  then also  $\phi \leftrightarrow \phi'$ .

**Example 3.2.**

$$\frac{\frac{\frac{(p \leftrightarrow q) \leftrightarrow r}{(p \leftrightarrow q) \leftrightarrow (r \leftrightarrow s)}}{p \leftrightarrow (q \leftrightarrow (r \leftrightarrow s))}}$$

$$\frac{(p \leftrightarrow q) \leftrightarrow (r \leftrightarrow s)}{p \leftrightarrow (q \leftrightarrow (r \leftrightarrow s))}$$

From now on, parentheses will only be used for clarity. If there are no parentheses, assume the rightmost notation.

The commutativity allows us to swap every variable in a formula with any other in that formula.

**Lemma 3.2.** Every  $\Phi = A \leftrightarrow \phi \leftrightarrow \beta_1 \leftrightarrow \dots \leftrightarrow \beta_n \leftrightarrow \psi \leftrightarrow \Gamma$  can be rewritten to  $\Psi = A \leftrightarrow \psi \leftrightarrow \beta_1 \leftrightarrow \dots \leftrightarrow \beta_n \leftrightarrow \phi \leftrightarrow \Gamma$

*Proof.*

The proof is an induction on  $n$ , the number of elements between  $\phi$  and  $\psi$ . The subformula  $\phi \leftrightarrow \beta_1 \leftrightarrow \dots \leftrightarrow \beta_n \leftrightarrow \psi$  can be rewritten without changing anything in the subformula's  $A$  and  $\Gamma$ . The first and last element of the subformula can be exchanged if

$$\phi \leftrightarrow \beta_1 \leftrightarrow \dots \leftrightarrow \beta_{n-1} \leftrightarrow \psi$$

can be rewritten to

$$\psi \leftrightarrow \beta_1 \leftrightarrow \dots \leftrightarrow \beta_{n-1} \leftrightarrow \phi$$

Two variables next to each other can be exchanged, therefore every

$$\phi \leftrightarrow \beta_1 \leftrightarrow \dots \leftrightarrow \beta_{n-1} \leftrightarrow \beta_n \leftrightarrow \psi$$

can be rewritten to

$$(\phi \leftrightarrow \beta_1 \leftrightarrow \dots \leftrightarrow \beta_{n-1} \leftrightarrow \psi) \leftrightarrow \beta_n$$

Now ignoring the last variable we rewrite this to

$$(\psi \leftrightarrow \beta_1 \leftrightarrow \dots \leftrightarrow \beta_{n-1} \leftrightarrow \phi) \leftrightarrow \beta_n$$

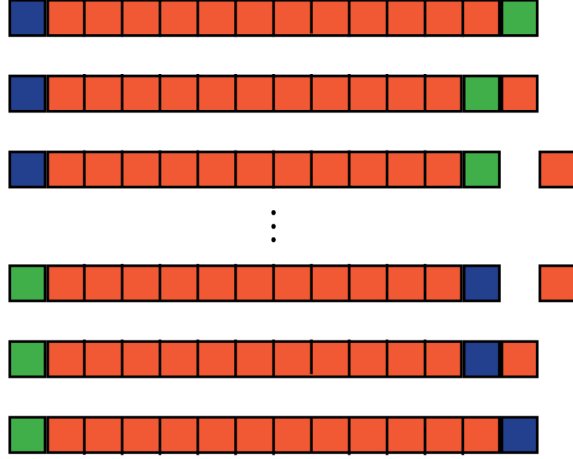
which can be rewritten to

$$\psi \leftrightarrow \beta_1 \leftrightarrow \dots \leftrightarrow \beta_{n-1} \leftrightarrow \beta_n \leftrightarrow \phi$$

Every recursive step has a formula of length  $n - 1$ . If a formula contains two occurrences of variables these can be exchanged because of the symmetry rule  $\frac{B \leftrightarrow A}{A \leftrightarrow B}$ .  $\square$

Figure 3.1 is a visual example. You can see this problem as a game with bricks. You have to swap the first and the last brick (the green and the blue ones) and are only allowed to swap two bricks next to each other.

Figure 3.1: Visual example of exchanging elements of a formula.



### 3.1 Deciding bi-implicational formulas in classical logic

The proposition  $(p \leftrightarrow ((q \leftrightarrow q) \leftrightarrow (r \leftrightarrow p))) \leftrightarrow ((r \leftrightarrow (p \leftrightarrow q)) \leftrightarrow (q \leftrightarrow p))$  is true. The deduction for this is quite long but simple. It starts with repeatedly applying the associativity and commutativity rules and so rewriting the formula to a version in which the appearances of the variables are grouped. In the deduction below the associativity and commutativity steps are omitted, this would take many steps and we have seen in section 3 that this is a valid deduction. After that we apply the rule  $\frac{A \leftrightarrow A}{\top}$  and have a proposition only consisting of  $\leftrightarrow$  and  $\top$ . The special case of the rule,  $\frac{\top \leftrightarrow \top}{\top}$  is valid and can be applied several times to conclude  $\top$ .

**Example 3.3.**

$$\frac{\frac{(p \leftrightarrow ((q \leftrightarrow q) \leftrightarrow (r \leftrightarrow p))) \leftrightarrow ((r \leftrightarrow (p \leftrightarrow q)) \leftrightarrow (q \leftrightarrow p))}{\dots}}{\frac{((p \leftrightarrow p) \leftrightarrow (p \leftrightarrow p)) \leftrightarrow (((q \leftrightarrow q) \leftrightarrow (q \leftrightarrow q)) \leftrightarrow (r \leftrightarrow r))}{(\top \leftrightarrow \top) \leftrightarrow ((\top \leftrightarrow \top) \leftrightarrow \top)}}{\dots}$$

$$\frac{}{\top}$$

In another case we can see that  $(p \leftrightarrow ((q \leftrightarrow q) \leftrightarrow (r \leftrightarrow p))) \leftrightarrow ((r \leftrightarrow p) \leftrightarrow (q \leftrightarrow p))$  is not a tautology.

**Example 3.4.**



$$\frac{\frac{\frac{(p \leftrightarrow ((q \leftrightarrow q) \leftrightarrow (r \leftrightarrow p))) \leftrightarrow ((r \leftrightarrow p) \leftrightarrow (q \leftrightarrow p))}{\dots}}{((p \leftrightarrow p) \leftrightarrow (p \leftrightarrow p)) \leftrightarrow (((q \leftrightarrow q) \leftrightarrow q) \leftrightarrow (r \leftrightarrow r))}}{(\top \leftrightarrow \top) \leftrightarrow ((\top \leftrightarrow q) \leftrightarrow \top)}}{\dots} \frac{}{q}$$

What matters in deciding a proposition is the number of appearances of a variable in that proposition. If that number is even for all variables the proposition is true. If one variable appears an odd number, the proposition is not a tautology. Now we have to formalize this.

For the proof of the algorithm we use the space  $\mathbb{E}$ , which is the space of odd and even numbers modulo 2,  $\mathbb{E} = \{0, 1\}$  where the following rules apply:

$$0 + 0 = 0$$

$$1 + 1 = 0$$

$$0 + 1 = 1 = 1 + 0$$

Like normal operations on natural numbers addition in  $\mathbb{E}$  is commutative and associative.\*

We will use the operator  $\langle - \rangle : \mathcal{L}_{\leftrightarrow} \rightarrow \mathbb{X}$ . In  $\mathbb{X}$  are functions that tell us whether a variable in  $\psi$  appears odd or even in  $\psi$ . A function in  $\mathbb{X}$  is at finitely many places unequal to 0. I.e. in a formula  $\psi$  is a finite number of variables with odd appearances. Every variable appears at most once and the variables are ordered. (There is a bijection between  $\mathbb{N}$ , the natural numbers, and  $\text{Var}$ .)

**Definition 3.3.**

$$\mathbb{X} = \sum_{p \in \text{Var}} p \cdot \mathbb{E}$$

What we want of formula of the type  $\mathcal{L}_{\leftrightarrow} \rightarrow \{\top, \perp\}$ , something that maps a sentence in  $\mathcal{L}_{\leftrightarrow}$  to truth values. We can do this by using an extra step:  $\mathcal{L}_{\leftrightarrow} \rightarrow \mathbb{X}$  and  $\mathbb{X} \rightarrow \mathbb{B}$

**Definition 3.4.** The function  $\langle - \rangle : \mathcal{L}_{\leftrightarrow} \rightarrow \mathbb{X}$  is defined:

$$\begin{aligned} \langle p \rangle &= 1 \cdot p \\ \langle A \leftrightarrow B \rangle &= \langle A \rangle + \langle B \rangle \\ \langle \top \rangle &= 0 \end{aligned}$$

**Example 3.5.**

$$\begin{aligned} &\langle (p \leftrightarrow (q \leftrightarrow p)) \leftrightarrow p \rangle \\ &= \langle p \leftrightarrow (q \leftrightarrow p) \rangle + \langle p \rangle \\ &= (\langle p \rangle + \langle q \leftrightarrow p \rangle) + \langle p \rangle \\ &= (\langle p \rangle + (\langle q \rangle + \langle p \rangle)) + \langle p \rangle \\ &= (p + (q + p)) + p \\ &= p + q + p + p \\ &= p + p + p + q \\ &= p + q \end{aligned}$$

Now we have a mapping from the formula to an expression in  $\mathbb{X}$ . The final step is to map  $\mathbb{X}$  to truth-values.

---

\*Bi-implicational formulas are also associative and commutative. Because of this it does not matter whether we use these properties before or after the mapping of the formula to  $\mathbb{X}$ .

**Definition 3.5.** The interpretation of sentences in  $\mathbb{X}$  to Boolean values is such that 0 in  $\mathbb{X}$  maps to 1 in the Boolean values. Everything else maps to 0.

$\langle \Phi \rangle$  translates a formula  $\Phi$  from  $\mathcal{L}_{\leftrightarrow}$  to space  $\mathbb{X}$ . It is not possible to translate something in  $\mathbb{X}$  back to exactly the same formula it was  $\mathcal{L}_{\leftrightarrow}$ . All pairs of the same variable are removed. For the translation from  $\mathbb{X}$  to  $\mathcal{L}_{\leftrightarrow}$  we use the operator  $\llbracket \cdot \rrbracket$ .

**Definition 3.6.** The function  $\llbracket - \rrbracket : \mathbb{X} \rightarrow \mathcal{L}_{\leftrightarrow}$  is defined:

$$\begin{aligned} \llbracket 0 \rrbracket &= \top \\ \llbracket x \rrbracket &= x \quad \text{with } x \in \text{Var} \\ \llbracket x + b \rrbracket &= \llbracket x \rrbracket \leftrightarrow \llbracket b \rrbracket \end{aligned}$$

Note that  $\llbracket a + b \rrbracket$  translates to  $a \leftrightarrow b$  only if  $a \neq b$ . The space  $\mathbb{X}$  is defined such that every variable occurs at most once, so the case  $a = b$  is not possible.

**Example 3.6.** Translating example 3.5 back to  $\mathcal{L}_p$ .

$$\begin{aligned} &\llbracket q + p \rrbracket \\ = &\llbracket p \rrbracket \leftrightarrow \llbracket q \rrbracket \\ = &p \leftrightarrow q \end{aligned}$$

This means that  $\llbracket \langle \Phi \rangle \rrbracket$  is not always equal to  $\Phi$ . This is because the information in  $\mathbb{X}$  only tells us whether a variable has odd or even occurrences.  $\llbracket \langle \Phi \rangle \rrbracket$  gives us a  $\Phi'$  with all pairs of occurrences of the same variable removed. We can deduce  $\Phi'$  from  $\Phi$  with the deduction rules in section 3 and know from that  $\Phi \leftrightarrow \Phi'$ .

**Lemma 3.3.** For all  $x \in \mathbb{X}$   $\llbracket x \rrbracket$  is a tautology if and only if  $x = 0$ .

*Proof.*  $\llbracket 0 \rrbracket = \top$  and  $\top$  is a tautology.

Let  $\llbracket x \rrbracket$  be a tautology with  $x = p + x'$  and  $p \in \text{Var}$ . We know that  $p$  is not in  $x'$  and therefore not in  $\llbracket x' \rrbracket$ . Now choose a valuation  $v$  such that  $v(p)$  is not equal to  $v(\llbracket x' \rrbracket)$ . It follows that  $v(p) \neq v(\llbracket x' \rrbracket)$  and therefore  $v \neq \llbracket x \rrbracket$ . Consequently  $\llbracket x \rrbracket$  is not a tautology, so  $x$  has to be 0.  $\square$

**Definition 3.7.**  $| - |$  is the number of variables used in a proposition. It is not the number of occurrences of variables, i.e.  $|p \leftrightarrow p \leftrightarrow q| = 2$ .

**Lemma 3.4.** For all propositions  $\Phi$  is  $\llbracket \langle \Phi \rangle \rrbracket$  equivalent to  $\Phi$ .

*Proof.* This is proven by an induction on the number of used variables.

If  $|\Phi| = 0$  then  $\langle \Phi \rangle = 0$  because no variable in  $\text{Var}$  is ever used. Because  $\llbracket 0 \rrbracket = \top$  we know  $\llbracket \langle \Phi \rangle \rrbracket = \top$  which is equivalent to  $\top$ .

Let  $|\Phi| = n$  with variables  $p_1 \dots p_n$ . Now we have a proposition  $\Psi$  such that  $|\Psi| = n - 1$  and  $\Psi$  uses the variables  $p_2 \dots p_n$ .  $P$  is a proposition with only the variable  $p_1$ .  $P = \overbrace{p_1 \leftrightarrow \dots \leftrightarrow p_1}^{m \text{ occurrences of } p}$  and  $m$  is equal to the number of occurrences of  $p_1$  in  $\Phi$ .

$$\forall \Phi \exists \Psi, P \quad \Phi \leftrightarrow (\Psi \leftrightarrow P)$$

$$\langle \Phi \rangle = \langle \Psi + P \rangle$$

$$\langle \Phi \rangle = \langle \Psi \rangle + \langle P \rangle$$

If  $m$  is even then  $\langle P \rangle = 0$  and  $\langle \Phi \rangle = \langle \Psi \rangle + \langle P \rangle = \langle \Psi \rangle$ .

$$\llbracket \langle \Phi \rangle \rrbracket = \llbracket \langle \Psi \rangle \rrbracket$$

$p_1$  is even in  $\Phi$  and also even in  $\llbracket \langle \Phi \rangle \rrbracket$ . If  $\llbracket \langle \Psi \rangle \rrbracket \leftrightarrow \Psi$  then also  $\llbracket \langle \Phi \rangle \rrbracket \leftrightarrow \Phi$ .

If  $m$  is odd  $\langle P \rangle = p_1$  and  $\langle \Phi \rangle = \langle \Psi \rangle + p_1$ .

$$\llbracket \langle \Phi \rangle \rrbracket = \llbracket \langle \Psi \rangle + p_1 \rrbracket$$

$$\llbracket \langle \Phi \rangle \rrbracket = \llbracket \langle \Psi \rangle \rrbracket \leftrightarrow p_1$$

$p_1$  is odd in  $\Phi$  and also odd in  $\llbracket \langle \Phi \rangle \rrbracket$ . If  $\llbracket \langle \Psi \rangle \rrbracket \leftrightarrow \Psi$  then also  $\llbracket \langle \Phi \rangle \rrbracket \leftrightarrow \Phi$ . □

**Lemma 3.5.** If  $\Phi$  and  $\Psi$  are equivalent then  $\langle \Phi \rangle = \langle \Psi \rangle$ .

*Proof.*  $\llbracket \langle \Phi \leftrightarrow \Psi \rangle \rrbracket \leftrightarrow (\Phi \leftrightarrow \Psi)$  is a tautology. If  $\Psi$  and  $\Phi$  are equivalent,  $\Psi \leftrightarrow \Phi$  is a tautology and equivalent to  $\llbracket \langle \Psi \leftrightarrow \Phi \rangle \rrbracket$ , which is also a tautology, because it is equivalent to a tautology. Because of lemma 3.3 and the definition of  $\langle - \rangle$  we know  $\langle \Phi \leftrightarrow \Psi \rangle = 0 = \langle \Psi \rangle + \langle \Phi \rangle$ , which means that every variable appears even. This is only the case if a variable appears even in both  $\langle \Psi \rangle$  and  $\langle \Phi \rangle$  or odd in both functions. □

**Lemma 3.6** (Soundness). If  $\Phi$  is a tautology,  $\langle \Phi \rangle = 0$ .

*Proof.* From lemma 3.4 we know that  $\Phi \leftrightarrow \llbracket \langle \Phi \rangle \rrbracket$ . Because  $\Phi$  is a tautology,  $\llbracket \langle \Phi \rangle \rrbracket$  is also a tautology. Lemma 3.3 gives us that for every  $x$ , if  $\llbracket \langle x \rangle \rrbracket$  is a tautology,  $\langle x \rangle = 0$ . Therefore  $\langle \Phi \rangle = 0$ . □

**Lemma 3.7** (Completeness). If  $\langle \Phi \rangle = 0$ ,  $\Phi$  is a tautology.

*Proof.* From lemma 3.3 we know that if  $\langle \Phi \rangle = 0$  then  $\llbracket \langle \Phi \rangle \rrbracket$  is a tautology. From lemma 3.4 we know that  $\llbracket \langle \Phi \rangle \rrbracket$  is equivalent to  $\Phi$ . Because  $\Phi$  is equivalent to a tautology it is also a tautology. □

## 3.2 Complexity of classical decision algorithm

**Theorem 3.1** (Decision of the bi-implication fragment for classical logic). There exists a  $n \log n$ -time algorithm for deciding TAUT for the  $\mathcal{L}_{\leftrightarrow}$ -fragment for classical logic.

*Proof.* This is the algorithm and its complexity.

“On input  $\langle \Phi \rangle$ :

1. Sort the list with the variables first and then the  $\leftrightarrow$  and  $O(n \log n)^\dagger$   
parentheses
2. Check if every variable has an even number of occurrences.  $O(n)$   
If there is a variable that has an odd number of occurrences,  
*reject*
3. When reaching the  $\leftrightarrow$  *accept*”  $O(1)$  □

---

<sup>†</sup>There is a wide variety of sorting algorithms, for example merge sort which is in  $O(n \log n)$

From lemma's 3.1 and 3.2 we know that parentheses and the order of the variables have no influence on the validity of a proposition in  $\mathcal{L}_{\leftrightarrow}$ . Therefore we know that we can sort the input without loss of information. This makes the second step of the algorithm easier.

Because the list is sorted we can check the occurrences of the variables by scanning the list once. If we have a sequence of  $p$ 's and then a sequence of  $q$ 's we will never encounter another  $p$ . So if the sequence of  $p$ 's is odd when we encounter a  $q$ , we will never find another  $p$  to make it even. If  $p$  is odd,  $\mathbb{X} = p$  and that will map to  $\perp$ , even if all other variables are even. Therefore the algorithm can reject when finding that a variable occurs odd.

Deciding TAUT for the complete propositional language is NP-hard. For the bi-implication fragment there is a  $n \log n$ -algorithm, meaning that this is in class P. If  $P \neq NP$  then deciding  $\mathcal{L}_{\leftrightarrow}$  is strict easier than  $\mathcal{L}_p$ .

### 3.3 Intuitionistic decision algorithm?

We want to find a similar way to solve  $\mathcal{L}_{\leftrightarrow}$  for intuitionistic logic. The algorithm for classical logic greatly relied on associativity, allowing us to pair variables.

The rule  $\frac{(A \leftrightarrow B) \leftrightarrow C}{A \leftrightarrow (B \leftrightarrow C)}$  is not true in intuitionistic logic. A counterexample can be given with  $A = p$  and  $B = \perp = C$ . If intuitionistic logic was associative then

$$((p \leftrightarrow (\perp \leftrightarrow \perp)) \leftrightarrow ((p \leftrightarrow \perp) \leftrightarrow \perp))$$

should be true. We will show this is not true by rewriting both sides of the bi-implication.

$$\frac{p \leftrightarrow (\perp \leftrightarrow \perp)}{\frac{p \leftrightarrow \top}{p}}$$

$$p \leftrightarrow ((p \leftrightarrow \perp) \leftrightarrow \perp)$$

maps to

$$p \leftrightarrow ((p \leftrightarrow \perp) \rightarrow \perp \wedge \perp \rightarrow (p \leftrightarrow \perp))$$

Anything can be deduced from  $\perp$  i.e.  $\frac{\perp}{A}$  is a tautology and therefore we can remove the part  $\perp \rightarrow (p \leftrightarrow \perp)$  because of  $\frac{A \wedge \top}{A}^\ddagger$

$$p \leftrightarrow ((p \leftrightarrow \perp) \rightarrow \perp)$$

$$p \leftrightarrow ((p \rightarrow \perp \wedge \perp \rightarrow p) \rightarrow \perp)$$

Again  $\perp \rightarrow p$  can be removed and  $p \rightarrow \perp$  infers  $\neg p$ .

$$p \leftrightarrow (\neg p \rightarrow \perp)$$

$$p \leftrightarrow \neg\neg p$$

---

<sup>‡</sup>The proof of  $A \leftrightarrow A \wedge \top$  is left to the reader.

The bi-implication in intuitionistic logic is associative if and only if  $p \leftrightarrow \neg\neg p$ , which maps to  $p \rightarrow \neg\neg p \wedge \neg\neg p \rightarrow p$ . The first part of the conjunction is true. For the second part we have a countermodel in figure 1.4. Due to the soundness theorem the bi-implication is not associative.

Some rules do shorten a proposition. In the next chapter we will see that solving INTTAUT for  $\mathcal{L}_p$  is a PSPACE-complete problem. If we can shorten the input by polynomial time operations, the problem is most likely solved in less time. Some patterns are equivalent to shorter patterns, but these patterns will not always occur and we cannot exchange the variables to make these patterns. Therefore this method is not guaranteed to work on all propositions.

For example  $A \leftrightarrow (A \leftrightarrow B)$  is equivalent to  $B$ , because if  $A$  is true it is equivalent to  $B$  and if  $B$  is true it is a tautology. The structure  $(A \leftrightarrow B) \leftrightarrow (B \leftrightarrow A)$  is a tautology, something that can be proven with natural deduction. When exploring longer formulas like  $A \leftrightarrow (B \leftrightarrow (C \leftrightarrow (C \leftrightarrow (B \leftrightarrow A))))$  it seems that odd and even appearances of variables are still important in intuitionistic logic, but this is only a guess. Without using associativity the proof of this would be quite different from that of classical logic.



## Chapter 4

# Complexity of intuitionistic logic

In [2] Švejdar provides a reduction of quantified Boolean formula's to intuitionistic logic. He does this by constructing an intuitionistic formula  $A^*$  from the quantified Boolean formula  $A$ .

### 4.1 Reduction from tqbf to IntTaut

We consider quantified Boolean formula's of the form  $A = Q_m p_m \dots Q_1 p_1 B(p)$  with  $B(p)$  a Boolean proposition using the variables  $p_1$  to  $p_m$  and no quantifiers. Every variable  $p_n$  is bound by a quantifier  $Q_n$ . We want to construct an intuitionistic formula  $A^*$ . The construction of this formula will be done by constructing  $A_0^*$  and  $A_{j+1}^*$  for all  $j < m$ . Finally  $A^*$  is  $A_m^*$ . If  $A^*$  has a countermodel,  $A$  is not valid.

In classical logic we have a countermodel of a QBF if and only if there is an evaluation  $e$  of the variables  $p_{j+1} \dots p_m \xrightarrow{e} \{\top, \perp\}^m$  such that  $e \not\models Q_{j-1} p_{j-1} \dots Q_1 p_1 B(p)$ . This means that  $B(p)$  with all variables bigger than  $j$  with a valuation and the variables 1 to  $j$  without valuation leads to a contradiction.

If we have a formula defined by

$$D_0 = \perp, D_{n+1} = (p_{n+1} \rightarrow D_n) \vee (\neg p_{n+1} \rightarrow D_n)$$

we can find a counterexample for  $D_{n+1}$  by showing that neither  $(p_{n+1} \rightarrow D_n)$  nor  $(\neg p_{n+1} \rightarrow D_n)$  is true. The formula will contain two occurrences of  $D_n$ . This means that the formula grows exponentially in  $n$  while we want it to be in polynomial space.

To avoid the second appearance of  $D_n$  we introduce a new variable, being  $q_{n+1}$ , and rewrite  $D$  to a new version  $E$  with a single occurrence of  $E_n$  in  $E_{n+1}$ .

$$E_0 = \perp, E_{n+1} = (E_n \rightarrow q_{n+1}) \rightarrow (p_{n+1} \rightarrow q_{n+1}) \vee (\neg p_{n+1} \rightarrow q_{n+1})$$

This version of the formula contains a disjunction, but we can avoid using that by using another variable  $s$ . In section 4.2 we will need a formula without disjunction.

**Definition 4.1.**  $A_0^*$  is equal to  $B(\underline{p})$

For  $j > 0$  and  $Q_j = \exists A_j^*$  is:

$$(A_{j-1}^* \rightarrow q_j) \wedge ((p_j \rightarrow q_j) \rightarrow s_j) \wedge ((\neg p_j \rightarrow q_j) \rightarrow s_j) \rightarrow s_j$$

For  $j > 0$  and  $Q_j = \forall A_j^*$  is:

$$(A_{j-1}^* \rightarrow q_j) \wedge ((p_j \rightarrow q_j) \wedge (\neg p_j \rightarrow q_j) \rightarrow q_j) \rightarrow q_j$$

**Definition 4.2.**  $k \Vdash e$  if and only if  $k \Vdash p \iff e(p) = \top$  for all  $p$  in the domain of  $e$ .

**Lemma 4.1.** Let  $0 \leq j \leq m$  and let  $e$  be an evaluation of atoms  $p_{j+1}, \dots, p_m$ . Then  $e \not\models Q_j p_j \dots Q_1 p_1 B(\underline{p})$  if and only if  $A_j^*$  has a Kripke counter-example  $K$  with for every node  $k \Vdash e$  for all atoms  $p_i$  with  $i > j$ .

First we consider  $A_0^*$ . A counter-example is a valuation  $e$  such that  $e \not\models B(\underline{p})$ . If we have this valuation we can also construct a Kripke counter-example. This is the model with one node  $k$  such that  $k \Vdash e$ . If we have a Kripke model  $K$  for  $A_0^*$  this is not necessarily a one-point model, so we have to prove for all points that we can make  $e$ .

**Lemma 4.2.**

$$\forall p \in \text{Var} [\forall k \in K e \models p \iff k \Vdash p]$$

Consider  $k' \geq k$  and assume  $k \models A \rightarrow B$  and we want to show  $k \Vdash A \rightarrow B$ . If  $k' \Vdash A$  then also  $e \models A$ . Because of  $e \models A \rightarrow B$  also  $e \models B$ . Now  $k' \Vdash B$  and therefore  $k' \Vdash A \rightarrow B$ .

Now we want an induction on  $j$ . We distinguish for  $j > 0$  the cases  $Q_j = \exists$  and  $Q_j = \forall$ .

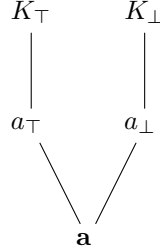
*Constructing a countermodel for the case  $Q_j = \exists$*  If we have a classical counterexample  $e \not\models \exists p_j Q_{j-1} p_{j-1} \dots Q_1 p_1 B(\underline{p})$  we know that  $Q_{j-1} p_{j-1} \dots Q_1 p_1 B(\underline{p})$  is false both with  $v(p_j) = \top$  and  $v(p_j) = \perp$ . This means that we will make two separate countermodels and combine these later. For the case  $v(p_j) = \top$  we will construct sub-model  $K_\top$  with root  $a_\top$ . On this root  $e_\top$  is valid.  $e_\top$  is the valuation function for  $p_m$  to  $p_{j+1}$  extended with  $p_j$ . Model  $K_\top$  is a counterexample to  $A_{j-1}^*$ . Similarly is  $K_\perp$  with root  $a_\perp$  and  $e_\perp$  constructed. This is another counter-model to  $A_{j-1}^*$ . Note that  $e_\perp$  has the same valuation as  $e_\top$  for  $p_m$  to  $p_{j+1}$  but is extended with  $\neg p_j$ . The two models are combined into one model  $K$  by making a new root  $a$  that has access to both  $a_\top$  and  $a_\perp$ . In root  $a$  variables are assigned such that  $a \Vdash e$ .

*Constructing a countermodel for the case  $Q_j = \forall$*

For the universal quantifier  $\forall$  the construction of a counter-model to  $A_j^*$  is similar. There is a counter-model if either  $e_\top \not\models Q_{j-1} p_{j-1} \dots Q_1 p_1 B(\underline{p})$  or  $e_\perp \not\models Q_{j-1} p_{j-1} \dots Q_1 p_1 B(\underline{p})$ . Either  $K_\top$  with root  $a_\top$  or  $K_\perp$  with root  $a_\perp$  will be constructed and be accessible from the new root  $a$  with  $a \Vdash e$ .

In the process of construction we used variables  $p, q$  and  $s$  and we have to assign values to these for the new node  $a$ . For  $p_{j+1}$  to  $p_m$  we assign the truth values according to  $e$ . The variables  $p_1 \dots p_j, q_1 \dots q_{j-1}$  and  $s_1 \dots s_{j-1}$  are negative in



Figure 4.1: Visualisation of  $A_j^*$ 

$a$ . These variables will be used in the counter-models  $A_i^*$  for  $i < j$  and if we would assign these to  $a$  we would not be able to choose a different valuation later because of the preservation of truth. The variable  $q_j$  is in  $a$  equal to the truth-value of  $A_{j+1}^*$  and  $s_j$  has the same value as  $(p_j \rightarrow q_j) \vee (\neg p_j \rightarrow q_j)$ .

Now we want to show that we have constructed a countermodel to  $A_j^*$ .

$$((A_{j-1}^* \rightarrow q_j) \wedge ((p_j \rightarrow q_j) \rightarrow s_j) \wedge ((\neg p_j \rightarrow q_j) \rightarrow s_j)) \rightarrow s_j$$

We have  $a \Vdash A_{j-1}^* \rightarrow q_j$  because  $q_j$  is defined with the same truth value as  $A_{j-1}$ . We also know  $((p_j \rightarrow q_j) \rightarrow s_j) \wedge ((\neg p_j \rightarrow q_j) \rightarrow s_j)$  to be true, because it is equivalent to  $((p_j \rightarrow q_j) \vee (\neg p_j \rightarrow q_j)) \rightarrow s_j$  and the meaning of  $s_j$  is defined as equivalent to  $(p_j \rightarrow q_j) \vee (\neg p_j \rightarrow q_j)$ .

We can show that  $a \not\Vdash (p_j \rightarrow q_j) \vee (\neg p_j \rightarrow q_j)$ , because  $a_{\top} \not\Vdash p_j \rightarrow q_j$  and  $a_{\perp} \not\Vdash \neg p_j \rightarrow q_j$ . From this we know that  $s_j$  is false. This makes  $A_j^*$  false because it is an implication with the arguments true but the conclusion false. We have constructed a counter-model from counterexample  $e$ .

*Deducing the valuation from counter-model  $K$  for  $Q_j = \exists$*  If we have a counter-model  $K$  to  $A_j$  with  $j > 0$  and  $Q_j = \exists$  we also have a valuation  $e$ . Model  $K$  has a root  $a$  with  $a \not\Vdash A_j^*$  and also  $a \not\Vdash s_j$ . From this we know  $a \not\Vdash p_j \rightarrow q_j$  because  $a \Vdash (p_j \rightarrow q_j) \rightarrow s_j$ . This means that there is a node  $b \geq a$  such that  $b \not\Vdash p_j \rightarrow q_j$  because  $b \Vdash p_j$  and also  $b \not\Vdash q_j$ . We have defined  $q_j$  by  $A_{j-1}^* \rightarrow q_j$  thus  $b \not\Vdash A_{j-1}^*$ . Thus we have a counterexample  $b$  for  $A_{j-1}^*$  with  $p_j$  true on all nodes. The node we now call  $b$  was called  $a_{\top}$  by constructing the model. The same way we can find a node  $a_{\perp} \geq a$  with  $a_{\perp} \Vdash \neg p_1$  and  $a_{\perp} \not\Vdash A_{j-1}^*$ .

We now have  $e_{\perp} \not\Vdash Q_{j-1}p_{j-1} \dots Q_1p_1B(\underline{p})$  and  $e_{\top} \not\Vdash Q_{j-1}p_{j-1} \dots Q_1p_1B(\underline{p})$  and from that we conclude  $e \not\Vdash \exists p_j \dots Q_1p_1\bar{B}(\underline{p})$ .

For the case  $Q_j = \forall$  the valuation  $e$  can be found in a similar way.

## 4.2 pspace-completeness of $\mathcal{L}_{\rightarrow}$

For the reduction we had no restrictions on which connectives could be used. It is possible to make the same reduction with only the language of variables and implications.

**Definition 4.3.**  $\mathcal{L}_{\rightarrow} = \text{Var} \mid \mathcal{L}_{\rightarrow} \rightarrow \mathcal{L}_{\rightarrow}$

In example 1.3 we have seen that the conjunction can be avoided in intuitionistic logic. We already avoided the use of disjunctions in our reduction. If we rewrite the negation form  $\neg A$  to  $A \rightarrow \perp$ , we only have variables,  $\rightarrow$ ,  $\top$  and  $\perp$ . The latter two can be replaced by yet another variable,  $r_j$  with the constant value true or false.

The result of this is that the implicational fragment of the language is also PSPACE-complete.

# Bibliography

- [1] Chris K. Caldwell. The top twenty prime pages.
- [2] Vítězslav Švejdar. On the polynomial-space completeness of intuitionistic propositional logic. *Archive for Mathematical Logic*, 42:711–716, 2003.
- [3] Richard Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM journal of computing*, pages 467–480, 1977.
- [4] Michael Sipser. *Introduction to the Theory of Computation*. Thomson course technology, 2 edition, 2006.
- [5] Richard Statman. Intuitionistic propositional logic is polynomial-space complete. *Theoretical Computer Science*, 9:67–72, 1979.
- [6] D. van Dalen. *Intuitionistic Logic*, pages 224 – 258. Blackwell, Oxford, 2001.
- [7] D. van Dalen and A.S. Troelstra. *Constructivism in Mathematics*, volume 121 of *Studies in Logic*. North-Holland, Amsterdam, 1988.



# Appendix A

## Turing machines

A Turing machine is a simple but powerful model of computation. It has unlimited memory without restrictions and is able to calculate the same things a normal computer can do. It has a read-write head and a tape from which the read-write head can read and write. Based on what the head reads and the current state of the machine there is a state transition, an output is written on the tape and the head moves to the left or right based on the transition function.

**Definition A.1.** A Turing machine is a 7 -tuple  $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ , where

1.  $Q$  is the set of states,
2.  $\Sigma$  is the input alphabet with  $\sqcup \notin \Sigma$ ,
3.  $\Gamma$  is the tape alphabet with  $\sqcup \in \Gamma$  and  $\Sigma \subseteq \Gamma$ ,
4.  $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L\}$  is the transition function,
5.  $q_0 \in Q$  is the start state,
6.  $q_{accept} \in Q$  is the accept state, and
7.  $q_{reject} \in Q$  is the reject state, where  $q_{accept} \neq q_{reject}$ .

A nondeterministic Turing machine is a Turing machine with transition function  $\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$ . This means that for every step there are multiple possibilities for the next step. The evaluation process of a non-deterministic Turing machine can be seen as a tree whose branches represent possible choices. A non-deterministic Turing machine accepts if there is at least one branch in the tree that accepts.

A language is decidable if and only if there is a nondeterministic Turing machine that decides it.

Time complexity of an algorithm on a Turing machine is given in the number of steps that is taken. Assumed is that every step (consisting of reading the input, writing the output, going to the new state and moving to the left or right) takes a finite and equal amount of time.

In computational complexity we do not actually construct Turing machines for every problem. The Church-Turing thesis states that every informal algorithm equals a Turing machine algorithm.\* If we have an algorithm to sort comparable elements, we might have a hard time writing the input and states

---

\*The Church-Turing thesis is based on two separate papers by Alonzo Church and Alan Turing. Church used  $\lambda$ -calculus to formalize algorithms and Turing used automatic-machines for this.

of a Turing machine that executes this algorithm. Therefore we use a more abstract definition. We do not define the states and input precisely, we simply say they exist. If we have a list  $L$  that we want to sort, we use the encoding  $\langle L \rangle$  of this list without specifying how it is encoded. The machine  $S$  that sorts the list requires that the list is encoded in that way. Instead of transitions we give more general descriptions of the behaviour of  $S$ . We simply state what the algorithm does, not how it is done. For example "if  $L$  contains more than one element, divide  $L$  in two parts  $L_1$  and  $L_2$  and apply  $S$  to  $\langle L_1 \rangle$  and  $\langle L_2 \rangle$ ." This does not only make it easier for the writer, but a reader can also see quite fast whether an algorithm works and if the complexity analysis is right.