COMP0009 Logic & Databases
Formal Reasoning

Robin Hirsch

October 7, 2020

Contents

1 Reading

Web pages such as wikepedia, stanford encylopedia of philosophy, etc. have really good material and are recommended. If you prefer a text book, here are some to consider.

- '*Logic: an introduction to elementary logic*' by W Hodges, Penguin, 1977. Cheap and accessible.
- Logic and Discrete Mathematics: A Concise Introduction* by Willem Conradie and Valentin Goranko, and published by John Wiley, 2015.
- '*A friendly introduction to mathematical logic*' by C Leary, Prentice Hall, 2000.
- '*First-order logic*' by R Smullyan, Springer-Verlag, Berlin, 1968.
- '*Formal logic, its scope and limits*', McGraw Hill, New York, 1967.
- '*Discrete mathematics*' by Richard Johnsonbaugh, Pearson Prentice-Hall, 6th edition 2005. Good for most of the theory strand of your degree, but does not cover tableau theorem proving, nor complexity theory.
- '*Logic for computer science*' by S Reeves and M Clarke, Addison-Wesley, 1990. Out of print.
- 'Discrete mathematics with applications' by S Epp, PWS Publishing Co., 1995. Covers much of COMP102p, COMP104p, COMP202p and MATH6301. Does not cover tableau. Accessible.
- '*Discrete mathematics for computer scientists*' by John Truss, Addison-Wesley, 1999.
- '*Mathematical logic and computability*' by J Keisler, Schaum International Series, 1996. Tough but covers much.
- '*Logic for maths and computer science*' by S Burris, Prentice Hall, 1998. No tableau, no complexity or computability, but otherwise fine.
- '*Introduction to automata theory, languages and computation*' by J Hopcroft, R Motwani and J Ullman, Addison-Wesley, third edition, 2007. Good for automata theory and regular languages. Also useful for computability and complexity (3rd or 4th year course). Nothing on predicate logic.

2 Logic: Introduction and background

Last year you studied two widely used logics: propositional logic and predicate logic. You started by accurately defining the syntax of formulas and we focused on the semantics, or meaning, of formulas. You saw that predicate logic is much more expressive than propositional logic. In this course we will focus on algorithms that can be used to automatically calculate the meaning of a formula. In particular we will devize algorithms that determine whether a formula is valid or not. We will see that whereas for propositional logics there are perfectly good algorithms to determine validty, for predicate logic no such algorithm has been found, nor will any be found in the future (assuming the *Church Turing thesis*, more on that next year).

A logic usually comprises three parts:

- A language.
- A meaning.
- An inference system.

The formal aspects of the first category is called *syntax.* In a logic you should be able to say what the sentences or formulas are, what the structure of a sentence is, etc., etc. We consider syntax to be abstract rules about strings of abstract symbols. There is no meaning attached to them. In natural language the rules of grammar are to do with the syntax of sentences. Compilers deal with the syntax of programs. The second category is sometimes called *semantics* and it concerns the meaning or interpretation of the formulas in the language. In a sense, semantics takes us beyond the form of an argument and starts to deal with the content. Questions to do with semantics include: what does this formula mean? is this sentence true? is it valid or satisfiable? To understand the semantics of a computer program you need to know what the program actually does. The final category is an inference or proof system. This is usually a purely syntactic device that can be used to calculate the meaning of a formula. So it is a kind of bridge between syntax and semantics.

In this course we deal only with *formal logic.* That means that each of the three parts should be formal and unambiguous. So the language should be specified by definite rules which tell you whether a formula is correct (in the language) or not. And there should be a clear, unambiguous way of interpreting a formula, i.e. a way of giving it meaning. And the inference system should make no use of intuition or anything which is not clearly specified in the logic. Since you are all computer scientists, the concept of a formal language should be familiar: every programming language that I've heard of is a formal one. This contrasts with natural language and everyday reasoning which are partly *informal.*

Our insistence on using a formal logic does restrict the kind of thing we can talk about and the kind of argument we can put. In philosophy there have been criticisms of formal logic in that it has a tendency to reduce the complexity of the real world to rather simple strings of symbols and rules of deduction. There is a tendency in formal logic to insist that things are always clear-cut whereas in the real world this is not so. However, the restriction to formal logic is necessary if we wish to automate our procedures. It is ideal for computer science. My conclusion is that formal logic can be a very useful tool to clarify the reasoning process, but that you must be prepared to modify the logic or replace it by a new logic, if it becomes clear that it is inadequate to represent and reason about the problem in hand.

Formal logic originated with Aristotle (and Euclid). In response to a crisis in Greek mathematics, he proposed a formal system of *syllogisms.* Aristotle's system held sway for over 2,000 years, but in the nineteenth century mathematicians, in particular De Morgan who was working at UCL, and Frege who was working somewhere else, developed a more modern system of logic. Frege's quantifier logic has dominated the subject ever since and is now called predicate logic or first-order logic. During the first half of the twentieth century predicate logic had profound implications for mathematics and philosophy. One of the more significant philosophical developments in logic was called *logicism*, a school of thought developed by Frege and then Russell and Whitehead. The logicists wanted to base the whole of mathematics and perhaps the whole of science on simple logical foundations. Frege wrote

> The firmest method of proof is obviously the purely logical one, which, disregarding the particular characteristics of things, is based solely upon the laws on which all knowledge rests. (Preface to [Fre72] page 103)

He then continues, in the same article, to attempt to demonstrate that arithmetic and probably geometry, differential and integral calculus can be handled by this very rigorous method of deduction. To quote Frege again, 'arithmetic is a branch of logic and need not borrow any ground of proof whatever from experience or intuition.'

This attempt to place mathematics on a rigorous base was promoted particularly by the mathematician David Hilbert. Hilbert's program (his *Entscheidungsproblem*) asked that mathematics, or at least arithmetic, should be expressed in a formal language (such as Frege's predicate logic) and that mathematical proofs should be based on finitary methods (such as a proof system with finitely many axioms). Hilbert identified the following key problems:

- Show that the finitary proof system does not contradict itself, i.e. prove that it is consistent,

- Show that the finitary proof system can prove any true statement, i.e. prove that the system is complete,
- Find an *algorithm* that can tell if a formula is true or not.

It is interesting to note that the logicist program was destroyed not by an argument in philosophy but in logic. Gödel's famous incompleteness theorem states, roughly, that in any consistent, recursively enumerable, formal system, adequate for arithmetic there would be true statements for which there exist no proof. The incompleteness theorem defeated logicism by showing that the first two items (above) in Hilbert's program could not both be achieved. Later Turing showed that the third item was also impossible.

Later in this century, logic played an important part in formal language theory. Its use in computer science is more recent. First-order logic

- provides formal program specifications (Z)
- provides a framework for proving program correctness, though often modal logic is used instead,
- is used as actual programming languages, in particular prolog is based on a fragment of first-order logic (the Horn-clause fragment)
- is used for knowledge representation
- is used for databases (SQL)
- gives an important approach to artificial intelligence
- does several other things I can't think of right now.

3 Revision

Remember propositional and predicate logic? Well, never mind, here's a reminder.

3.1 Propositional Logic

Syntax A *proposition* is just a letter $p, q, r, \ldots$ or sometimes $p_0, p_1, p_2, \ldots$. The *propositional connectives* are $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction), $\rightarrow$ (implication) and $\leftrightarrow$ (two-way implication). All but negation are binary connectives, negation is unary.

A propositional *formula* (fm) is defined by

$$\begin{aligned} fm \quad ::= & \quad prop|\neg fm|(fm \vee fm)|(fm \wedge fm) \\ & \quad |(fm \rightarrow fm)|(fm \leftrightarrow fm) \end{aligned}$$

where *prop* is any propositional letter.

Semantics A *valuation* is a function v from the propositions into $\{\top, \bot\}$. A valuation v can be extended to a unique *truth function* v' defined on all possible formulas. A truth function v' satisfies

$$\begin{aligned} v'(\neg\phi) = \top & \quad \Leftrightarrow \quad v'(\phi) = \bot \\ v'(\phi \vee \psi) = \top & \quad \Leftrightarrow \quad v'(\phi) = \top \text{ or } v'(\psi) = \top \\ v'(\phi \wedge \psi) = \top & \quad \Leftrightarrow \quad v'(\phi) = v'(\psi) = \top \\ v'(\phi \rightarrow \psi) = \top & \quad \Leftrightarrow \quad v'(\psi) = \top \text{ or } v'(\phi) = \bot \\ v'(\phi \leftrightarrow \psi) = \top & \quad \Leftrightarrow \quad v'(\phi) = v'(\psi) \end{aligned}$$

for all formulas ϕ, ψ.

A formula ϕ is *valid* if for all valuations v we have $v(\phi) = \top$. The notation $\models \phi$ means that ϕ is valid.

A formula ϕ is *satisfiable* if for at least one valuation v we have $v(\phi) = \top$.

The result of applying v to ϕ depends only on the propositional letters that occur in ϕ.

3.2 Predicate Logic

Syntax A first order language $L(C, F, P)$ is determined by a set C of constant symbols, a set F of function symbols and a non-empty set P of predicate symbols. Each function symbol and predicate symbol has an associated *arity*, a natural number. We may write f^n to indicate that f is an n-ary function symbol, similarly p^n is an n-ary predicate symbol. There is a countably infinite set V of variable symbols. A *term* for $L(C, F, P)$ is defined by

$$\begin{array}{rcl} term & = & c\ (c \in C) \mid v\ (v \in V) \mid f^n(term_0, \ldots, term_{n-1})\ (f^n \in F) \\ atom & = & p^n(term_0, \ldots, term_{n-1})\ (p^n \in P) \\ fm & = & atom \mid \neg fm \mid (fm_0 \vee fm_1) \mid \exists v fm\ (v \in V) \end{array}$$

A closed term involves no variable symbols. A sentence is a formula with no free variables.

Semantics A structure $S = (D, I)$ for $L(C, F, P)$ is a non-empty set D and an interprettation I, where $I = (I_c, I_f, I_p)$, I_c maps consant symbols in C to elements of D, I_f maps an n-ary function symbol $f \in F$ to an n-ary function over D, i.e. a map from D^n to D, and I_p maps an n-ary predicate symbol $p \in P$ to an n-ary relation over D, i.e. to a subset of D^n. If P includes the equality symbol $=$, then the symbol is always binary and is always interpretted as true equality, $I_p(=) = \{(d, d) : d \in D\}$.

Given a structure $S = (D, I)$, a variable assigment A is a map from V into D. Let $v \in V$ be a variable. Two variable assignments A, A^* are said to be v-equivalent if $A(x) = A^*(x)$ for all $x \in V \setminus \{v\}$. Given a structure S and a variable assignment A we may interpret any term by

$$\begin{array}{rcl} c^{S,A} & = & I_c(c) \\ v^{S,A} & = & A(v) \\ f^n(t_0, \ldots, t_{n-1})^{S,A} & = & I_f(f^n)(t_0^{S,A}, \ldots, t_{n-1}^{S,A}). \end{array}$$

Formulas may be evaluated by

$$\begin{array}{rcl} S, A \models p^n(t_0, \ldots, t_{n-1}) & \iff & (t_0^{S,A}, \ldots, t_{n-1}^{S,A}) \in I_p(p^n) \\ S, A \models \neg fm & \iff & S, A \not\models \phi \\ S, A \models (fm_0 \vee fm_1) & \iff & S, A \models fm_0 \text{ or } S, A \models fm_1 \\ S, A \models \exists v fm & \iff & S, A^* \models fm \text{ for some } A^* \equiv_v A \\ S, A \models \forall v fm & \iff & S, A^* \models fm \text{ for all } A^* \equiv_v A \end{array}$$

For predicate logic there are two distinct notions of validity and two notions of satisfiability. Given a structure S and a formula ϕ we say that ϕ is valid in S, and we write $S \models \phi$, if $S, A \models \phi$ for every variable assignment A. We say that ϕ is satisfiable in S if there is some variable assignment A such that $S, A \models \phi$. If ϕ is a sentence that ϕ is valid in S iff ϕ is satisfiable in S.

If ϕ is valid in all possible structures S then we say that ϕ is valid and we write $\models \phi$. If there is some structure S and ϕ is satisfiable in S then we say that ϕ is satisfiable.

For a fixed structure S, note that $\neg\phi$ is satisfiable in S iff $S \not\models \phi$. The problem of checking whether ϕ is satisfiable in S is called *model checking*. For arbitrary structures, $\neg\phi$ is satisfiable iff $\not\models \phi$. Checking whether ϕ is valid is the task of *theorem provers*.

4 Inference

We are looking for algorithms that automatically calculate whether a formula is valid or not. For propositional logic, a truth table is an easy way to check whether a formula is valid or not. Here is a sketch of the algorithm.

```
Input($\phi$).
    write out all the possible valuations on the propositions in $\phi$
    for each of these valuations $v$ check if $v(\phi) = \top$.
    if its true in each case return "yes" else "no".
```

EXERCISE 1

1. *These instructions leave out a lot of details. Can you make these instructions more precise?*
2. *If ϕ contains n distinct propositions, how many possible valuations are there? In order terms, what is the worst-case run-time of this algorithm?*
3. *Devise an algorithm to test the satisfiability of a propositional formula.*

For a formula with many propositions the truth table method might not be practical, it is an *intractable problem* (more on that next year). However, this algorithm is sure to terminate eventually and will always give the correct answer. Hence the validity problem for propositional logic is *decidable.*

For predicate logic we have two types of validity to check — validity of ϕ in a fixed structure S, or validity of ϕ over all possible structures. The focus here will be on the latter problem (though any method to test validity is likely to help us check validity in a single fixed model). We cannot use a truth table to test validity of predicate logic formulas. We do not have a "finite number of valuations to check", instead we are required to check an infinite number of possible structures, indeed the number of structures is so large that it is not even a set. Even if we fix on a single structure $S = (D, I)$, the domain D might be infinite so we would have to check infinitely many different variable assignments to be sure that a formula was valid in S.

Let us consider some valid formulas and see if we can automate the reasoning that leads us to conclude that the formula is indeed valid.

1. if $\theta(p_0, p_1, \ldots, p_{n-1})$ is a propositional tautology using only the propositions $p_0, \ldots, p_{n-1}$ and if $\phi_0, \phi_1, \ldots, \phi_{n-1}$ are arbitrary predicate formulas then $\theta(\phi_0, \phi_1, \ldots, \phi_{n-1})$ is valid, where this formula is obtained by substituting ϕ_i foreach occurence of p_i in θ. e.g. $(\forall x P^1(x) \to (\neg Q^2(x, c) \to \forall x P^1(x))$ is an instance of the tautology $p \to (q \to p)$.
2. $((\exists v \neg\phi) \leftrightarrow (\neg\forall v\phi))$ is valid, for any predicate formula ϕ and any variable v.
3. if the equality symbol $=$ is included in the language then $\forall x \exists y\ x = y$ is valid.

EXERCISE 2 1. *Prove all three parts, above.*

2. *Can you think of any other valid formulas?*

We will consider two of the main approaches to validity checking: axiomatic systems and tableaus. In each case, we will first see how they work for propositional logic, then we will extend them to predicate logic.

4.1 Hilbert Systems for propositional logic

Describing a Hilbert proof system is easy; implementing it as a workable theorem prover is more difficult. The proof system consists of a set A of *axioms* and a set R of *inference rules.* The set of axioms should be a set of valid formulas. For example $(p \to p)$ and $(p \leftrightarrow \neg\neg p)$ could be axioms. An inference rule is a rule of deduction: it says "if we know that these formulas are true we may deduce that this formula is also true". Modus Ponens is the main example of an inference rule

$$\frac{\phi \qquad (\phi \to \psi)}{\psi}$$

There are several other rules of inference we could use, for example *modus tollens*:

$$\frac{(\phi \to \psi) \qquad \neg\psi}{\neg\phi}$$

or *hypothetical syllogism*

$$\frac{(\theta \to \phi) \qquad (\phi \to \psi)}{(\theta \to \psi)}$$

however, Modus Ponens is all we need for propositional logic.

The idea now is that we can use the axioms and the inference rules to prove new formulas are valid. Formally a *proof* of a formula ϕ is a finite sequence of formulas

$$S_0, S_1, S_2, \ldots, S_n = \phi$$

such that each formula S_i is either an axiom or there is $j, k < i$ and S_i can be obtained from S_j, S_k using some inference rule in R. If there is a proof of ϕ then ϕ is called a *theorem* and we may write

$$\vdash \phi$$

We have seen that every propositional formula has an equivalent using only $\to$ and $\neg$ as connectives. It is also clear that we can eliminate any 'double negations'. Hilbert proof systems often adopt a language with only these connectives and with no double negations.

Example Let the set A of axioms consist of all instances of the following schemas:-

I. $(\theta \to (\phi \to \theta))$

II. $((\theta \to (\phi \to \psi)) \to ((\theta \to \phi) \to (\theta \to \psi)))$

III. $((\neg\theta \to \neg\phi) \to (\phi \to \theta))$

as θ, ϕ, ψ range over arbitrary propositional formulas. Note that these schemas actually stand for infinitely many axioms. However, a simple parser should be able to test whether a formula matches any of these axiom schemas.

EXERCISE 3 *Verify, using truth tables, that any instance of these schemas is valid.*

Let the set R of inference rules consist of modus ponens only. We show that $(p \to p)$ is a theorem; here a the proof.
1. $((p \to ((p \to p) \to p)) \to ((p \to (p \to p)) \to (p \to p)))$
This is an instance of axiom scheme II with $p, (p \to p), p$ for θ, ϕ, ψ.
2. $(p \to ((p \to p) \to p))$
Instance of axiom scheme I using formulas p and $(p \to p)$.
3. $((p \to (p \to p)) \to (p \to p))$
From 1 and 2 using modus ponens.
4. $(p \to (p \to p))$
Instance of axiom scheme I using formulas p and p.
5. $(p \to p)$
From 3 and 4 by modus ponens.

The example above is an example of a *sound* logical system — the axioms are all valid and each inference rule has the following properties: if both premises are valid then the conclusion is also valid. It follows from this that only valid formulas can be proved in a sound logical system.

EXERCISE 4 *Prove by induction over the length of a proof that if ϕ can be proved in a sound logical system then ϕ is valid.*

The thing that is less easy to see is that this proof system is also *complete* — if ϕ is any valid propositional formula using only the connectives $\neg, \rightarrow$, then ϕ can be proved using only the three axiom schemas and the inference rule. We won't prove the completeness theorem here. Its not really that difficult but it would take too much time and anyway we prove the corresponding result when we get on to the tableau method.

We can summarize the soundness and completeness theorems neatly using the notation we have already introduced. The soundness of a logical system is expressed as

$$\text{If } \vdash \phi \text{ then } \models \phi$$

and the completeness of the system is expressed as

$$\text{If } \models \phi \text{ then } \vdash \phi$$

Now suppose we want to use inference rules for arbitrary propositional formulas, not just those using $\neg$ and $\rightarrow$. We can handle this by adding additional axioms that force the extra connectives $\vee, \wedge$ to behave in the right way. So let A consist of all instances of the three axiom schemas given earlier, plus all instances of

IV. $(\neg\neg\phi \rightarrow \phi)$

V. $(\phi \rightarrow \neg\neg\phi)$

VI. $((\phi \vee \psi) \rightarrow (\neg\phi \rightarrow \psi))$

VII. $((\phi \rightarrow \psi) \rightarrow (\neg\phi \vee \psi))$

VIII. $((\phi \wedge \psi) \rightarrow \neg(\phi \rightarrow \neg\psi))$

IX. $((\phi \rightarrow \psi) \rightarrow \neg(\phi \wedge \neg\psi))$

EXERCISE 5

1. *Using modus ponens and hypothetical syllogism as rules of inference and the axiom schemas above, prove $(\neg p \vee p)$. You may find the proof of $(p \rightarrow p)$ useful for this.*

2. *Prove $(\neg\phi \rightarrow (\phi \rightarrow \psi))$.*

3. *Using only modus ponens as an inference rule and the axiom schemas above prove that $((\neg\phi \rightarrow \phi) \rightarrow \phi)$.*

4.2 Validity versus Satisfiability

So we can use the axioms and rules to tell whether a formula is valid or not. If it is valid then there is a proof of it (by completeness) and if there is a proof of it then the formula is valid (soundness). But what if we wish to know whether a formula ϕ is satisfiable or not? No problem. We know that ϕ is satisfiable if and only if $\neg\phi$ is not valid. So, to see if ϕ is satisfiable you try to prove $\neg\phi$ using the axioms and rules. If you can prove $\neg\phi$ then you know that ϕ is not satisfiable and conversely if there is no proof of $\neg\phi$ then ϕ is satisfiable.

4.3 Hilbert Systems for Predicate Logic

There are axiomatic systems that work quite well for predicate logic. Of course there will be more axioms than we had for propositional logic and we have extra inference rules too. It turns out that by adding axioms and inference rules we can obtain an axiomatic system that is sound and complete, for first-oder logic. Strangely, this is not enough to show that first-order logic is decidable.

We assume that our language only uses the quantifier $\forall$ — occurrences of $\exists$ must be replaced, as above. There are two inference rules:

$$\frac{\phi \qquad (\phi \to \psi)}{\psi}$$

and

$$\frac{\phi(x)}{\forall x \phi(x)}$$

for any formula ϕ and any variable x. The second rule, known as 'universal generalisation' says, informally, that if $\phi(x)$ can be proved valid then $\forall x \phi(x)$ must also be valid.

For axioms we start with an axiomatisation of propositional logic (axioms (I)–(III) above. Here we deal only with connectives $\neg, \to$. If connectives $\vee, \wedge$ are also included then axioms (IV)–(IX) are also inclued.

In addition we have some axiom schemas for the quantifiers.

X. $(\forall x \neg A \leftrightarrow \neg \exists x A)$

XI. $(\forall x A(x) \to A(t/x))$ if t is *substitutable* for x in A.

XII. $(\forall x(A \to B) \to (\forall x A \to \forall x B))$.

XIII. $(\phi \to \forall x \phi)$ if x does not occur free in ϕ.

Finally, if equality is included in our language we need special equality axioms.

XIV. $(x = x)$

XV. $((x = y) \to (t(x) = t(y/x)))$

XVI. $((x = y) \to (A(x) \to A(y/x)))$ if y is substitutable for x in A.

We write

$$\vdash \phi$$

if there is a proof of ϕ using these rules and only these axioms. More generally, if Γ is any set of sentences (assumptions), we write

$$\Gamma \vdash \phi$$

if there is a proof of ϕ using as axioms either the basic axioms or any of the sentences in Γ. Read the statement as 'ϕ can be proved using any sentence in Γ as an assumption'.

Certainly this system of axioms and rules is sound — all the axioms are valid in any structure under any assignment, and if the premises of any inference rule are valid then the consequent of that rule must also be valid. Hence any formula that can be proved with these rules (any theorem) must be valid. It also turns out that the system is complete — any valid formula can be proved in this system.

Example Here's an example of the proof of a valid statement: Socrates is a man, all men are mortal therefore Socrates is mortal. We use a constant symbol s to stand for Socrates, and we have two unary predicate symbols: $H^1(x)$ means 'x is human' and $M^1(x)$ means 'x is mortal'. We want to prove

$$\{H^1(s), \forall x(H^1(x) \to M^1(x))\} \vdash M^1(s)$$

Right, here is the proof.

1. $H^1(s)$ assumption.
2. $\forall x(H^1(x) \to M^1(x))$ assumption.
3. $(H^1(s) \to M^1(s))$, universal instantiation, from (2).
4. $M^1(s)$, by modus ponens from (1) and (3).

EXERCISE 6 *Since our system of axioms and rules is sound and complete, why can't we get a decidability result? That is, given any first-order formula ϕ, we want to know if it is valid or not. Soundness and completeness tells you that ϕ is valid if and only if it can be proved from the axioms. So to decide if ϕ is valid or not you just have to check whether there is a proof of ϕ using the axioms listed. Is this correct?*

4.4 Tableau for Propositional Logic

The main problem with the axiomatic system we have described is that although it is relatively easy to check that a proof of a formula is correct, it is much harder to construct a proof. You need intuition to know which axioms to pick and when to apply modus ponens. It is even worse if you do not know whether a formula ϕ can be proved or not. You could spend ages and ages trying different axioms and applying modus ponens to obtain new formulas. How long do you keep trying before you decide that ϕ cannot be proved? In fact there are algorithms which direct your choices of axioms and when to apply the inference rule, but it can be seen that the problem is not trivial.

In this course we will use the tableau method for two reasons: it is quite easy to use (though some may argue that natural deduction is easier) and it is very suitable for implementation on a computer.

4.5 Satisfiability or Validity

In brief, the tableau method works like this. You take a formula ϕ and place it at the root of a tree (called a tableau). There are expansion rules which make the tree grow until it is completed. If you end up with an open tableau then ϕ is satisfiable, if the tableau is closed then ϕ is not satisfiable.

What if you want to find out if ϕ is valid or not? You can still use a tableau, but instead you construct a tableau for $\neg\phi$. If the completed tableau for $\neg\phi$ is open then $\neg\phi$ is satisfiable and so ϕ is not valid. If the tableau for $\neg\phi$ is closed then $\neg\phi$ is not satisfiable and so ϕ is valid. Thus to find out if ϕ is valid or not it suffices to construct a tableau for $\neg\phi$. Next we show how to construct a tableau for a formula and how this tells us if the formula is satisfiable or not.

4.6 Tableau Construction Rules

To build a tableau for a formula ϕ we first place ϕ at the root of a tree. This tree has branches which start from the root and terminate at leaves. Each node of the tree is labelled by a formula. *Do not confuse these tableau trees with the parse trees for individual formulas, defined in COMP1002. The two types of trees are unrelated.* Then, step by step, we pick some formula from the tree and use it to expand the tree a little more, until none of the formulas in the tree can be expanded any more. When we expand a formula in the tree we place a tick beside the formula to indicate that the formula has bean dealt with and does not need expanding again. When we expand a formula ϕ we add new nodes to the tree at the leaves of each branch passing through ϕ.

Every formula is either a propositional letter p, a negated formula $\neg\phi$ for some ϕ or a disjunction $(\phi \vee \psi)$. A propositional letter cannot be expanded. Nor can a negated proposition $\neg p$. Propositions and negated propositions are called *literals* and cannot be expanded. Here are the expansion rules.

$$\alpha\text{-rule (i):} \quad \begin{tikzcd} \neg\neg\phi \arrow[d, dash] \\ \phi \end{tikzcd}$$

This means that when you expand a node n labelled by $\neg\neg\phi$ you place a tick next to it and add a new node labelled ϕ at the end of every branch passing through n. There are two other expansion rules.

$$\beta\text{-rule:} \quad \begin{tikzcd} & (\phi \vee \psi) \arrow[dl, dash] \arrow[dr, dash] & \\ \phi & & \psi \end{tikzcd}$$

This means that if you expand a node n labelled $(\phi \vee \psi)$ you place a tick next to it and two extra nodes on separate branches at the leaf of every branch through n, one labelled by ϕ and the other labelled by ψ. Finally,

$$\alpha\text{-rule (ii):} \quad \begin{tikzcd} \neg(\phi \vee \psi) \arrow[d, dash] \\ \neg\phi \\ \neg\psi \end{tikzcd}$$

This time you add two new nodes on the same branch at the end of each branch passing through our node n. The first node is labelled $\neg\phi$ and the second is labelled $\neg\psi$.

Now every formula is either a proposition, a disjunction or a negated formula. A proposition cannot be expanded and a disjunction is covered by the β-rule. A negated formula $\neg\phi$ is either a negated proposition which is a literal and cannot be expanded, or a negated negation (covered by α-rule (i)), or a negated disjunction (covered by α-rule (ii)) according to whether ϕ is a proposition, a negated formula or a disjunction. Thus every formula is either a literal or is covered by one of the three rules.

4.7 Closed and Open Branches

If you ever find a branch with two nodes, one labelled by some formula ϕ and the other node labelled by $\neg\phi$, then the branch is called a *closed branch*. You cannot satisfy the formula along this branch and you should draw a horizontal line under the branch. Do not expand any nodes in a closed branch (as it is a waste of time). If a branch is not closed it is *open*.

If every branch in a tableau is closed, we say that the tableau is closed. Otherwise, if there is at least one open branch, we say the tableau is open.

4.8 When do you stop?

A branch in a tableau is called *completed* if either the branch is closed or none of the nodes in the branch can be expanded. Thus in an open completed branch every node is either labelled by a literal (which cannot be expanded) or has a tick next to (so it has already been expanded). You stop expanding a tableau if either

- You find a completed, open branch, or
- The tableau is closed.

4.9 How can I tell if a formula is satisfiable or not?

Start a new tableau and place the formula ϕ at the root. Continue expanding until the tableau is closed, or there is a completed open branch. If the tableau is closed (i.e. every branch is closed) then ϕ is not satisfiable. If there is a completed open branch, say Θ, then ϕ is satisfiable. In this latter case we can construct a valuation making ϕ true, as follows. Look at all the literals that occur in the branch Θ. If the proposition p occurs in Θ then let $v(p) = \top$. If the negated proposition $\neg p$ occurs in Θ then let $v(p) = \bot$. For propositions p such that neither p nor $\neg p$ occur in Θ it doesn't matter how we define $v(p)$ so, just so that we have a definition, in this case we let $v(p) = \bot$.

Now, because Θ is an open branch, it is not possible that both p and $\neg p$ occur in Θ, so this definition of v is unambiguous. We want to show that $v(\phi) = \top$. This will show that ϕ is satisfiable. The way we do this is to prove, by structured induction, that for any formula $\psi \in \Theta$, $v(\psi) = \top$. It helps the induction if we also prove that if $\neg\psi \in \Theta$ then $v(\psi) = \bot$.

THEOREM 1 *Let T be a tableau for ϕ and let Θ be a completed, open branch of T. Let v be the valuation where for any proposition p we have $v(p) = \top$ if and only if $p \in \Theta$. Then for any formula $\psi \in \Theta$ we have $v(\psi) = \top$ and if $\neg\psi \in \Theta$ then $v(\psi) = \bot$.*

PROOF:

Base Case If ψ is a proposition p then if $p \in \Theta$ we have $v(p) = \top$ by definition of v. If $\neg p \in \Theta$ then, again by definition of v, $v(p) = \bot$. This proves the base case.

Inductive Hypothesis Assume, for some arbitrary formulas ψ and ρ that if $\psi \in \Theta$ then $v(\psi) = \top$ and if $\rho \in \Theta$ then $v(\rho) = \top$. Also if $\neg\psi \in \Theta$ then $v(\psi) = \bot$ and if $\neg\rho \in \Theta$ then $v(\rho) = \bot$.

Inductive Step Using our inductive hypothesis we must prove that the properties in the theorem also hold for $\neg\psi$ and for $(\psi \vee \rho)$. So we have to prove that (a) if $\neg\psi \in \Theta$ then $v(\neg\psi) = \top$ (b) if $\neg\neg\psi \in \Theta$ then $v(\neg\psi) = \bot$ (c) if $(\psi \vee \rho) \in \Theta$ then $v((\psi \vee \rho)) = \top$ and (d) if $\neg(\psi \vee \rho) \in \Theta$ then $v((\psi \vee \rho)) = \bot$. For no good reason, we'll do part (c) first.

If $(\psi \vee \rho) \in \Theta$ then, since Θ is completed, then $(\psi \vee \rho)$ must have been expanded by the β-rule. That means that either $\psi \in \Theta$ or $\rho \in \Theta$. In either case we can use our inductive hypothesis and conclude that either $v(\psi) = \top$ or $v(\rho) = \top$. In either case $v((\psi \vee \rho)) = \top$, as required.

Now we deal with part (a). If $\neg\psi \in \Theta$ then by the I.H. on ψ we know that $v(\psi) = \bot$ and so $v(\neg\psi) = \top$, as required.

For part (b), if $\neg\neg\psi \in \Theta$, then as Θ is completed, the formula must have been expanded by α-rule (i) and ψ must be on the branch Θ. By I.H. we conclude that $v(\psi) = \top$ and so $v(\neg\psi) = \bot$, as required.

Finally, for part (d), if $\neg(\psi \vee \rho) \in \Theta$, then as Θ is completed the formula must have been expanded using α-rule (ii), so both $\neg\psi$ and $\neg\rho$ are on the branch Θ. By I.H. $v(\psi) = v(\rho) = \bot$. Therefore $v(\psi \vee \rho) = \bot$, as required.

The theorem follows by structured induction on propositional formulas. □

COROLLARY 2 *If a completed tableau for a formula ϕ is open, then ϕ is satisfiable. If a completed tableau for ϕ is closed, then ϕ is unsatisfiable.*

PROOF:

If the completed tableau for ϕ is open, it contains a completed open branch. By the previous theorem this gives us a valuation satisfying ϕ, so ϕ is satisfiable.

Conversely, let ϕ be a satisfiable formula. We need to show that a tableau for ϕ cannot be closed. This is a reformulation of the second part of the corollary: if we show that "satisfiable implies open" then it follows that "closed implies unsatisfiable".

So assume ϕ is satisfiable and let v be a fixed valuation such that $v(\phi) = \top$. We prove by induction on the number of times the tableau has been expanded, that at each stage there is a branch Θ such that for every formula $\psi \in \Theta$ we have $v(\psi) = \top$.

Base Case For the base case we consider the tableau after zero expansions. So there is a single node, labelled ϕ, and just one branch in the tableau. By assumption $v(\phi) = \top$, so this one branch has the property that every node on it is labelled by a formula made true under the valuation v.

Induction Hypothesis Assume for some $n \geq 0$ that if T is a tableau obtained from the one-node tableau labelled by ϕ by the application of n expansions, then there is a branch Θ of T such that for all $\psi \in \Theta$ we have $v(\psi) = \top$.

Induction Step Now let T^+ be a tableau that is obtained from the one-node tableau labelled by ϕ by $n+1$ expansions. We must prove that T^+ contains a branch Θ^+, say, such that for all $\psi \in \Theta^+$ we have $v(\psi) = \top$.

Let T be the tableau obtained after n expansions, so T^+ is obtained from T by one additional expansion. By IH, T contains a branch Θ such that $v(\psi) = \top$ for all $\psi \in \Theta$. Let $t \in T$ be the node that gets expanded to produce T^+. Case 1. If $t \notin \Theta$ then Θ is still a branch of T^+ so we are done. Case 2. If $t \in \Theta$ then there are new nodes added at the end of Θ in T^+. We must consider subcases according to the formula expanded at t. If t is labelled by $\neg\neg\rho$ then ρ is added at the end of Θ. By assumption, $v(\neg\neg\rho) = \top$ hence $v(\rho) = \top$. Thus the branch obtained by adding the new node labelled ρ to Θ is the required solution. If t is labelled by $(\lambda \vee \rho)$ then by assumption we have $v(\lambda \vee \rho) = \top$ so either $v(\lambda) = \top$ or $v(\rho) = \top$ (or both). There will be two nodes added at the end of Θ in T^+: one labelled by λ and one labelled by ρ. Define the branch Θ^+ be extending Θ with whichever of these two is made true under v (or choose either if both are made true). Then Θ^+ satisfies the conditions. Finally if t is labelled by $\neg(\lambda \vee \rho)$ then as before $v(\neg(\lambda \vee \rho)) = \top$ so $v(\lambda) = v(\rho) = \bot$. Let Θ^+ be the branch of T obtained by extending Θ with two nodes, labelled $\neg\lambda$ and $\neg\rho$ respectively. Then Θ^+ satisfies the conditions.

By induction after any number of expansions, there will be a branch of the tableau such that every formula in the branch is made true by v.

Note that the branch in this induction must be open because a valuation cannot make both ψ and $\neg\psi$ true, so they cannot both occur in the branch. It follows that if ϕ is satisfiable then any tableau for ψ must contain an open branch. $\square$

4.10 Examples

Hopefully this will all get much clearer with some examples.

1. Is $\neg(p \vee \neg q)$ satisfiable?. Place the formula at the root of a new tableau. So far, there is only one node in the tableau and that node had not been ticked, nor is it a literal. So we must expand that node. The appropriate rule is the α-rule (ii), as the formula is a negated disjunction. This gives us

$$\begin{tikzcd}[row sep=small]
\neg(p \vee \neg q) \quad \surd \arrow[d, dash] \\
\neg p \\
\neg\neg q
\end{tikzcd}$$

Now there are three nodes in a single branch. The root node has been expanded and has a tick. The next node is labelled by $\neg p$ which is a literal. This cannot be expanded. That leaves the bottom node $\neg\neg q$ which gets expanded by α-rule (i).

$$\begin{array}{l}
\neg(p \vee \neg q) \quad \checkmark \\
\quad | \\
\neg p \\
\neg\neg q \quad \checkmark \\
\quad | \\
\quad q
\end{array}$$

Now, every node is either a literal or has been ticked so the single branch of the tableau is completed. There are no formulas ρ with both ρ and $\neg\rho$ in the branch so the branch is open. It follows that the formula $\neg(p \vee \neg q)$ is satisfiable.

A valuation v for it can be computed as in the theorem by letting $v(p) = \bot$ (as $\neg p$ is in the branch) and $v(q) = \top$ (as q is in the branch). Check that v makes every formula in the branch true.

2. Is $\neg((p \vee q) \vee \neg p)$ satisfiable? This formula expands by α-rule (ii) to

$$\begin{array}{l}
\neg((p \vee q) \vee \neg p) \quad \checkmark \\
\quad | \\
\neg(p \vee q) \\
\neg\neg p
\end{array}$$

We have a choice of two un-ticked, non-literal formulas to expand. Let us choose the top one first. This gets expanded by α-rule (ii), giving

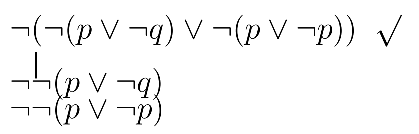
$$\begin{array}{l}
\neg((p \vee q) \vee \neg p) \quad \checkmark \\
\quad | \\
\neg(p \vee q) \quad \checkmark \\
\neg\neg p \\
\quad | \\
\neg p \\
\neg q
\end{array}$$

Now there is only one un-ticked, non-literal formula left. This gets expanded by α-rule (i), giving

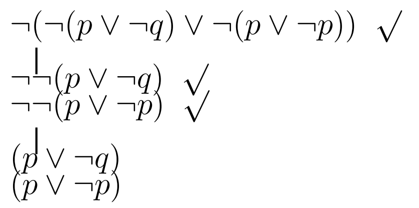
$$\begin{array}{l}
\neg((p \vee q) \vee \neg p) \quad \checkmark \\
\quad | \\
\neg(p \vee q) \quad \checkmark \\
\neg\neg p \quad \checkmark \\
\quad | \\
\neg p \\
\neg q \\
\quad | \\
p
\end{array}$$

Now there is only one branch and this contains both p and $\neg p$ and is therefore closed. Since the whole tableau is closed we conclude that the formula $\neg((p \vee q) \vee \neg p)$ is not satisfiable.

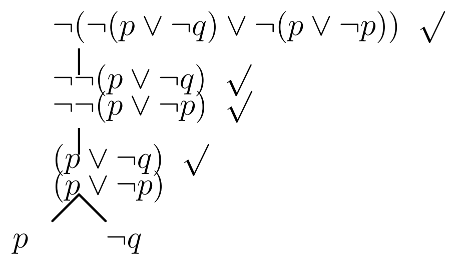
3. $\neg(\neg(p \vee \neg q) \vee \neg(p \vee \neg p))$. Using α-rule (ii) we get:


$\neg(\neg(p \vee \neg q) \vee \neg(p \vee \neg p))$ ✓
$\neg\neg(p \vee \neg q)$
$\neg\neg(p \vee \neg p)$

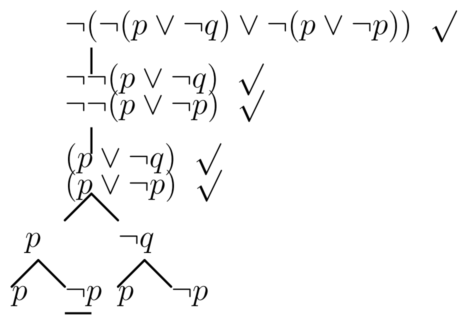
There are two un-ticked formulas, both double negations. Let us expand both (in one go) using α-rule (i) to get:


$\neg(\neg(p \vee \neg q) \vee \neg(p \vee \neg p))$ ✓
$\neg\neg(p \vee \neg q)$ ✓
$\neg\neg(p \vee \neg p)$ ✓
$(p \vee \neg q)$
$(p \vee \neg p)$

We still have two un-ticked formulas. Let us expand the top one first using β-rule.


$\neg(\neg(p \vee \neg q) \vee \neg(p \vee \neg p))$ ✓
$\neg\neg(p \vee \neg q)$ ✓
$\neg\neg(p \vee \neg p)$ ✓
$(p \vee \neg q)$ ✓
$(p \vee \neg p)$
p
$\neg q$

There is one un-ticked, non-literal formula left: $(p \vee \neg p)$. We expand this, using β-rule. Note that the two subformulas p and $\neg p$ must get added at the end of *every* branch that passes through the node we are expanding. In this case there are two such branches.


$\neg(\neg(p \vee \neg q) \vee \neg(p \vee \neg p))$ ✓
$\neg\neg(p \vee \neg q)$ ✓
$\neg\neg(p \vee \neg p)$ ✓
$(p \vee \neg q)$ ✓
$(p \vee \neg p)$ ✓
p
$\neg q$
p
$\neg p$
p
$\neg p$

There is a line under the second branch because it contains both p and $\neg p$, so this branch is closed. All the other branches are completed and open so the formula is satisfiable. The first branch gives the valuation v_1 where $v_1(p) = \top$ and all other propositions map to $\bot$. The third branch gives the valuation v_3 where $v_3(p) = \top$, $v_3(q) = \bot$ and all other propositions map to $\bot$ (so $v_1 = v_3$). The fourth branch gives the valuation v_4 where $v_4(p) = v_4(q) = \bot$ and all other propositions map to $\bot$.

EXERCISE 7 *Construct tableaus with the following formulas at the root. In each case either give a valuation showing that the formula is satisfiable (saying which branch of your tableau it comes from) or say if the formula is not satisfiable.*

1. $\neg(p \vee \neg q)$

2. $(p \vee \neg(p \vee \neg p))$

3. $\neg(\neg p \vee \neg\neg p)$

EXERCISE 8 *Let us revert to a fuller propositional language using* $\wedge$ *and* $\rightarrow$ *as well as* $\neg$ *and* $\vee$ *for connectives. Devise new expansion rules to deal with the extra connectives. All rules that produce only a single branch should be classified as* α*-rules and rules that produce two branches at each leaf are classified as* β*-rules.*

For example, what should you do with a formula $\neg(\psi \rightarrow \rho)$*?*

EXERCISE 9 *Use the tableau method to decide whether each of the formulas below are valid or not. NB. to test for validity you must first negate the formula — check the notes above if you are not clear.*

1. $(p \vee \neg p)$

2. $(p \rightarrow p)$

3. $(p \rightarrow (q \rightarrow p))$

4. $((p \rightarrow q) \rightarrow p)$

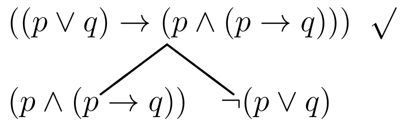
5. $((p \wedge q) \rightarrow (p \vee q))$

6. $\neg((p \wedge q) \rightarrow (p \vee q))$.

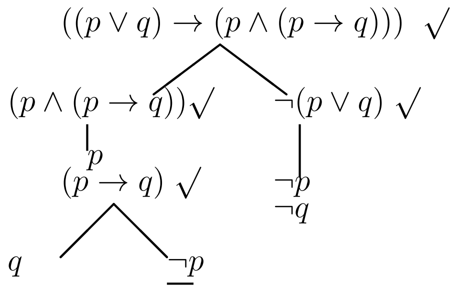
4.11 Disjunctive Normal Form Again

Tableau provides an effective method of finding an equivalent formula in DNF. Recall that closed branches cannot be satisfied but every completed open branch gives us a valuation that satisfies the formula at the root. So for each completed open branch we can form a conjunctive clause by taking the conjunction of all the literals that occur on that branch. Then the formula at the root is equivalent to the disjunction of all the clauses corresponding to the completed open branches.

Example Consider the formula
$((p \vee q) \rightarrow (p \wedge (p \rightarrow q)))$. To convert it to DNF make a tableau with this formula at the root. First expand the formula with a suitable β-rule for implication. You will need to do exercise 8 to do this.


$((p \vee q) \rightarrow (p \wedge (p \rightarrow q)))$ ✓
$(p \wedge (p \rightarrow q))$
$\neg(p \vee q)$

Further expansions yield


$((p \vee q) \rightarrow (p \wedge (p \rightarrow q)))$ ✓
$(p \wedge (p \rightarrow q))$ ✓
$\neg(p \vee q)$ ✓
p
$(p \rightarrow q)$ ✓
$\neg p$
$\neg q$
q
$\neg p$

There are three branches in the completed tableau, two open and one closed. The first branch contains the literals p, q so the conjunctive clause for this branch is $(p \wedge q)$. The second branch is closed because it contains both p and $\neg p$. We do not make a clause for this branch. The third branch contains $\neg p, \neg q$ so the clause is $(\neg p \wedge \neg q)$. Thus a DNF formula equivalent to the given one is

$$((p \wedge q) \vee (\neg p \wedge \neg q))$$

EXERCISE 10 *Use a truth table to confirm that* $((p \vee q) \rightarrow (p \wedge (p \rightarrow q)))$ *is equivalent to* $((p \wedge q) \vee (\neg p \wedge \neg q))$.

EXERCISE 11 *Construct tableaus to find DNF equivalents to each of the following.*

1. $\neg(p \rightarrow (q \vee p))$
2. $((p \vee q \vee r) \wedge (\neg p \vee \neg q))$
3. $(p \rightarrow (p \wedge \neg p))$.

4.12 Termination

If you start with a formula ϕ at the root of a tableau you keep picking un-ticked formulas and expanding them. Will this process ever end? In fact, for propositional logic the tableau method is guaranteed to terminate finitely. We can prove this in two stages, first we prove that every branch of a tableau must be finite, then we use a theorem from mathematics – König's tree lemma — to show that this means that the whole tree is finite.

THEOREM 3 *Any branch in a tableau has finite length.*

PROOF:

Let Θ be any branch in a tableau T with either the formula ϕ or the formula $\neg\phi$ at the root. We'll prove that Θ has finite length by structured induction on the formula ϕ.

Base Case ϕ is a proposition. At the root of the tableau we have either ϕ or $\neg\phi$. In either case we have a literal. This cannot be expanded so the tableau consists of a single node only. Hence the length of Θ must be one, certainly finite.

Inductive Hypothesis Assume that any branch in a tableau with ψ or $\neg\psi$ at the root is finite and any branch in a tableau with ρ or $\neg\rho$ at the root is finite.

Inductive Step We must prove that any branch in a tableau with $(\psi \vee \rho)$ or $\neg(\psi \vee \rho)$ at the root is finite and any branch with $\neg\psi$ or $\neg\neg\psi$ at the root is finite. If we have $(\psi \vee \rho)$ at the root, then there are two branches coming out. The first node on the first branch is ψ and the first node on the other branch is ρ. If you consider all the nodes from ψ down, you have a tableau with ψ at the root, so we can use our inductive hypothesis and conclude that all paths from ψ going down are of finite length. Similarly we know that all paths from ρ going down are of finite length. Now if Θ is any branch in the tableau T, Θ must start with the root $(\psi \vee \rho)$ and then either pass through ψ or through ρ. In either case the length of Θ is one plus some finite number and so it is finite.

Next suppose we have a tableau with $\neg(\psi \vee \rho)$ at the root. This gets expanded, by α-rule (ii), and $\neg\psi$ and $\neg\rho$ get placed immediately below the root of the tableau. Any branch starting from the root consists of $\neg(\psi \vee \rho)$ first, then some nodes which come from successive expansions of $\neg\psi$ and some nodes which come from successive expansions of $\neg\rho$. The ones which come from $\neg\psi$ occur in a branch in a tableau for $\neg\psi$ and by our inductive hypothesis they are finite in number. Similarly the ones which come from $\neg\rho$ are also finite in number. Hence the total number of nodes on the branch is 1 plus a finite number plus a finite number. This is finite.

If $\neg\psi$ is at the root of a tableau then our inductive hypothesis tells us immediately that any branch of the tableau is finite.

Finally, if $\neg\neg\psi$ is at the root of a tableau then this gets expanded by α-rule (i) and ψ gets placed immediately below the root. What we then get is $\neg\neg\psi$ at the root and a tableau for ψ below that. By our inductive hypothesis, any branch in a tableau for ψ is finite. So any branch in a tableau for $\neg\neg\psi$ consists of the root node followed by a finite path. Again, this is finite.

The theorem follows by structured induction.

□

THEOREM 4 (König's Tree Lemma) *Let T be a tree such that each node has only finitely many immediate successors (finite branching factor). If every branch is of finite length, then the number of nodes in the tree is finite.*

PROOF:

We work backwards by assuming that T has infinitely many nodes and showing that T must contain an infinite branch. For each node n in the infinite tree T, call n a *good node*[1] if there are infinitely many nodes below n in the tree. Because T is infinite it follows that the root is a good node. Let the root be the first node in a branch that we will construct.

Now the root has finitely many 'children' because we are assuming that there are finitely many immediate successors of each node. Now at least one of these children must be good, otherwise each child would have only finitely many nodes beneath it. But then the root would be succeeded by a finite set of bad nodes each with only finitely many beneath it. This would contradict the fact that the root has infinitely many nodes beneath it (the sum of a finite number of finite numbers is finite). So the root has at least one good child. Pick any good child and let this be the next node in the path under construction. This process, of picking a good child from the last node chosen, can be repeated infinitely. Thus we will construct an infinite branch.

We have shown, for finitely branching trees, that if the tree is infinite then there must be an infinite branch. It follows, conversely, that if the tree has no infinite branch the tree must be finite. □

COROLLARY 5 *The process of contructing a tableau for a propositional formula and repreatedly expanding nodes must terminate finitely.*

PROOF:

Suppose for contradiction (you should be getting used to this style of proof by now) that ϕ is a formula and it is possible to keep expanding nodes in a tableau for ϕ indefinitely. The 'limit' of this process is an infinite tableau T with ϕ at the root. The branching factor of T is 2 (each node has at most two immediate successors). By König's tree lemma (theorem 4) T has an infinite branch. This contradicts theorem 3.

□

So we have a second method of showing that propositional logic is decidable.

Method To prove that a formula ϕ is valid, make a tableau with $\neg\phi$ at the root. Keep expanding the tableau until it is completed. This process must terminate finitely. If the completed tableau for $\neg\phi$ is closed then ϕ is valid. If the completed tableau is open then ϕ is not valid.

[1]Words like "good" or "nice" are often used in formal reasoning as local variables standing for certain properties. Here we use the word "good" for the duration of this proof. Afterwards we might want to re-use the word for some other purpose.

4.13 Soundness and Completeness of Propositional Tableaus

Suppose we have a formula ϕ and we want to know if it is valid or not. Using the tableau method we construct a tableau with $\neg\phi$ at the root. If the tableau is closed then we know that $\neg\phi$ is not satisfiable, so ϕ is valid. If the completed tableau ends up open then we know that $\neg\phi$ is satisfiable so ϕ is not valid.

When we say that the tableau method is *sound* we mean that *if* you can construct a closed tableau for $\neg\phi$ then ϕ must certainly be a valid formula. This means that anything we prove by the tableau method is certainly valid. Soundness is usually the easy part to check for most inference systems.

When we say that the tableau method is *complete* we mean that if ϕ is any valid formula then you can construct a closed tableau for $\neg\phi$. This means that any valid formula can be proved using the tableau method. Showing completeness is usually the hard part to check for most inference systems.

By corollary 2 and the proof of termination we know that the tableau method is both sound and complete.

4.14 Tableau for Predicate Logic

We have seen that checking if a formula $\neg\phi$ is valid is equivalent to checking if the formula ϕ is satisfiable. Now ϕ is satisfiable if and only if there is some structure S and assignment A such that $S, A \models \phi$. If we want to check if such a structure exists there is a problem. We can find many other models for ϕ simply by using different names for the domain D. Certainly there is no finite limit to the number of different models we can construct this way and in fact the collection of distinct models we get this way is so big that it is not even a set.

Example Consider the sentence

$$\phi = \exists x \exists y (\neg =^2 (x, y) \wedge \forall z (=^2 (z, x) \vee =^2 (z, y))).$$

OK, we know each other well enough now do that we can drop the formality slightly and use the more familiar notation

$$\exists x \exists y ((x \neq y) \wedge \forall z ((z = x) \vee (z = y))).$$

This formula is satisfiable. Take a structure S with domain $D = \{a, b\}$. Then for any assignment A we have $S, A \models \phi$. To see why this holds,

$$\begin{aligned} S, A \models \exists x \exists y ((x \neq y) \wedge \forall z ((z = x) \vee (z = y))) \text{ iff} & \\ S, A^* \models \exists y ((x \neq y) \wedge \forall z ((z = x) \vee (z = y))) & \\ (\text{for some } x\text{-variant } A^* \text{ of } A) & \\ \text{iff} \quad S, A^{**} \models ((x \neq y) \wedge \forall z ((z = x) \vee (z = y))) & \\ (\text{for some } y\text{-variant } A^{**} \text{ of } A) & \end{aligned}$$

But for any assignment A there is an x variant A^* which agrees with A except $A^*(x) = a$ and there is a y-variant A^{**} which agrees with A^* (in particular $A^{**}(x) = a$) except $A^{**}(y) = b$. Yet with this assignment A^{**} we have

$$S, A^{**} \models (x \neq y)$$

because $(A^{**}(x), A^{**}(y)) = (a, b) \notin I_p(=)$. Also, for any z variant $A^\dagger$ of A^{**} either $A^\dagger(z) = a$ or $A^\dagger(z) = b$ (because there are only two points in the domain). Wlog (without loss of generality, pronounced 'wlog') let $A^\dagger(z) = a$. Then

$$S, A^\dagger \models (z = x)$$

because $(A^\dagger(z), A^\dagger(x)) = (a, a) \in I_p(=)$. Since this is true for any z-variant $A^\dagger$ of A^{**}, we conclude that

$$S, A^{**} \models \forall z((z = x) \vee (z = y))$$

Thus

$$S, A^{**} \models ((x \neq y) \wedge \forall z((z = x) \vee (z = y)))$$

Hence

$$S, A \models \phi$$

However, we could equally well have taken a structure T with domain $E = \{0, 1\}$. There really isn't any difference between S and T except for the names of the points in the domain. This makes it hard to check if a formula has a model or not. Luckily, if a formula has a model at all, there is always a standard form for the model and we don't have to check other models with different names for the domain points.

Definition of Herbrand models A structure $S = (D, I)$ is called a *Herbrand* structure for the language $L = L(C, F, R)$ if

- D is the set of closed terms of L
- for each closed term t we have $[t]^{S,A} = t$ (for any assignment A, though this isn't' really relevant as t contains no variables.)

THEOREM 6 *For any formula ϕ if there is a structure T and an assignment A such that $T, A \models \phi$ then there is a Herbrand structure S and an assignment B such that $S, B \models \phi$.*

We won't prove this here but it means that to check if a formula is satisfiable or not, it suffices to check the Herbrand models only. Still, Herbrand models can be infinite so this still isn't very easy.

EXERCISE 12 *Let $L(C, F, R)$ have constants $C = \{0, 1\}$, functions $F = \{s^1\}$ and predicates $P = \{=^2, R^1\}$. What are the elements of the Herbrand model?*

EXERCISE 13 *Under what conditions on the language L can the Herbrand structure be finite?*

For propositional logic it is possible to tell if a formula is valid (or satisfiable) by constructing a truth table. We can't do this here. To check if a sentence of predicate logic has a model we need to check if it is true in a Herbrand model, but these can be infinite. Of course, if the Herbrand structure is finite (see exercise 13) then we can check if our formula is true in the Herbrand structure and this gives us a decision procedure. This is more or less what the tableau method does — it actually tries to build a Herbrand structure for the formula and either closes if the formula is unsatisfiable (just as in the propositional case) or (step by step) it constructs a Herbrand structure where the formula is satisfied. The only problem with this is that the structure may be infinite.

4.15 First-order Tableaus

We use the same kind of tableau construction that we used for propositional logic, though we have some extra rules and it is a bit more complicated. We still have α and β rules to deal with $\neg\neg\phi$, $(\phi \vee \psi)$ and $\neg(\phi \vee \psi)$ (and $(\phi \wedge \psi), \neg(\phi \wedge \psi)$ if $\wedge$ is in the language, also $(\phi \rightarrow \psi)$ etc. if $\rightarrow$ is in the language). You might find it useful to look back to your notes on propositional tableau at this point. We also need rules to tell us what to do with formulas of the form $\exists x\phi$, $\neg\exists x\phi$, $\forall x\phi$ and $\neg\forall x\phi$.

Universal Formulas For $\forall x\phi$ and $\neg\exists x\phi$ we have a γ rule. Suppose $\forall x\phi$ labels a node n in a tableau. Expansion rule γ_1 says 'for some closed term t add $\phi(t/x)$ as a child to each leaf below n in the tableau, but *do not tick the node n unless it has been expanded with all possible closed terms.*' The formula $\phi(t/x)$ is the formula obtained from ϕ by replacing all free occurrences of the variable x by the closed term t. The rule for $\neg\exists x\phi$ is similar: γ_2 says to add $\neg\phi(t/x)$ as a child to each leaf below n but do not tick the node n, unless you have now expanded this node with all possible closed terms.

PROBLEM 1 *Why don't we have to insist that the term t is substitutable for $x \in \phi$?*

The reason we do not tick the node n is this: to show that the formula $\forall x\phi$ is satisfiable we have to show (more or less) that ϕ holds under all possible assignments to x. Because of the theorem about Herbrand models, we need only check assignments to closed terms, but we still have to check all of them. Thus, after expanding the node n using a particular closed term t you can expand it again using a different closed term s. Clearly this can go on infinitely (if the set of closed terms is infinite). We also need to think of a 'fair policy' whereby each node in the tableau gets expanded eventually (rather than expanding the same node by the γ rule again and again and missing all the other nodes). Also for a node labelled by a universal formula we want to ensure that for any closed term t, this node will be expanded by the term t eventually. If you've done some work (in MATH6301) on countable sets then you should know that it is possible to do this.

Existential Formulas For $\exists x\phi$ and $\neg\forall x\phi$ we have δ rules. If $\exists x\phi$ labels the node n, rule δ_1 says 'add $\phi(p/x)$ as the child of every leaf below n where p is a *parameter*, or a constant new to the tableau. *Do tick the node n.*' Similarly, if $\neg\forall x\phi$ labels the node n then δ_2 says to add $\neg\phi(p/x)$ as the child to every leaf below n and tick the node n.

The intuitive content of the δ rules is perhaps less obvious. Suppose $\exists x\phi$ is satisfiable in S. That means, for some assignment A, that $S, A \models \exists x\phi$, so for some x-variant A^* of A we have $S, A^* \models \phi$. We would like to name the element $A^*(x)$ but we do not know its name, or perhaps it does not have a name. So we pick a name (p) which we know has not been used for anything else. To do otherwise would be to introduce the possibility of confusion (and we wouldn't want that, would we?)

Parameters (or Skólem constants as they are sometimes known) serve the intended role. They constitute a stock of constants, which are available to name entities for the δ rules. When we choose a parameter in applying the rule, we know that it cannot be the name of anything else; p is not only new to the language L, it is also new to the tableau.

These are the rules then. To determine whether a formula ϕ is satisfiable place ϕ at the root of a new tableau. Continue to expand nodes of the tableau, preferably in a 'fair' order. If, at any stage, we produce a closed tableau (i.e. a tableau in which every branch contains both θ and $\neg\theta$ for some formula θ) then ϕ is definitely not satisfiable. Also, if we produce an open, completed branch (i.e. a branch where every node is ticked) then the formula is satisfiable. As with propositional tableaus we can look at the literals — the atoms and the negated atoms — that occur on an open, completed branch and construct a structure from them. The only problem is that we might go on forever building a bigger and bigger tableau and it never closes, nor do we ever get an open completed branch.

PROBLEM 2 *Suppose we are building a tableau for a formula ϕ and the tableau construction never terminates. Does this mean that ϕ is satisfiable or unsatisfiable?*

Let's summarize what we do know. If a tableau for ϕ closes then ϕ is unsatisfiable. If a tableau for ϕ contains a completed open branch then ϕ is satisfiable. Finally, if the tableau construction goes on forever without closing or producing a completed open branch then ϕ is satisfiable. This is because the *limit* of the tableau construction is an infinite tableau containing an (infinite) open branch. A model for ϕ can then be read off from this open branch, in rather the same sort of way as we did for propositional tableaus. The fact that the branch and the constructed model may be infinite does not matter.

Conversely, if ϕ is not satisfiable then, provided we pick nodes fairly, eventually the tableau will definitely become closed. We don't know in advance how long this will take, but we know it will happen after a finite amount of time. This is the completeness theorem.

So, if we are building a tableau for ϕ and we don't know if ϕ is satisfiable or not then we just have to keep expanding the tableau. After the tableau has, say, 10,000 nodes if it hasn't become closed we might think to ourselves:

> Hmm. This is taking ages. I wonder if this is ever going to finish. I bet it isn't. ϕ is probably satisfiable, and we're going to get an infinite tableau.

The problem is that if we assume that ϕ is satisfiable we might be wrong. Perhaps the tableau would have become closed after just one more expansion.

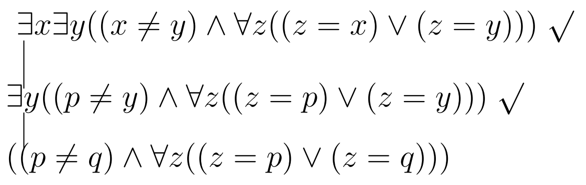
Predicate logic is called *semi-decidable.* You can always prove that an unsatisfiable formula is unsatisfiable: keep building a tableau till, eventually, it will become closed. But in general this procedure won't prove that a satisfiable formula is satisfiable. In fact it can be shown that predicate logic is not decidable: there is no algorithm, that is guaranteed to terminate finitely, that takes a formula ϕ and outputs "yes" if ϕ is valid and "no" otherwise. We won't prove that here but you will probably see a proof in COMP3004 in your third year.

Example Let $L(C, F, R)$ have no constant or function symbols and just one predicate symbol $=$. Is the formula

$$\exists x \exists y((x \neq y) \wedge \forall z((z = x) \vee (z = y)))$$

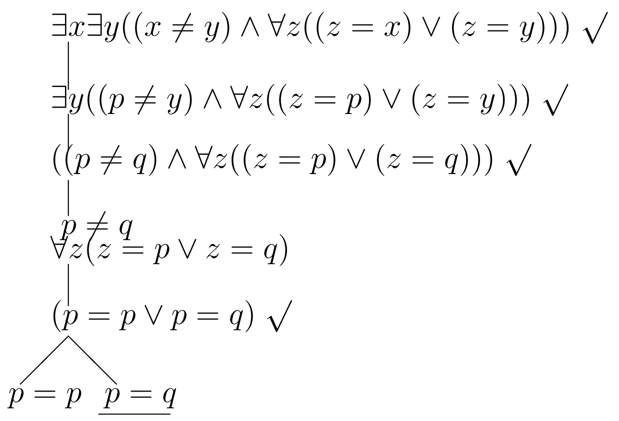
satisfiable?

Put the formula at the root of a new tableau. This is an existential formula so we expand it with rule δ using a new parameter p, and we tick the root node. Again we use rule δ to expand the new node with a new parameter q to get



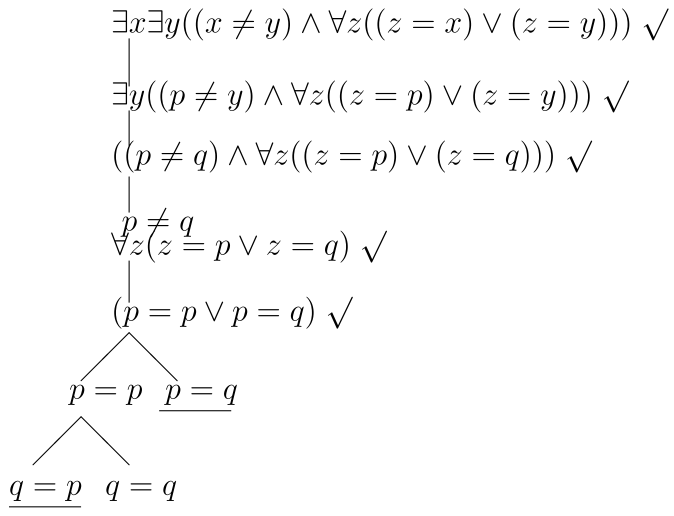
$\exists x \exists y((x \neq y) \wedge \forall z((z = x) \vee (z = y)))$ ✓
$\exists y((p \neq y) \wedge \forall z((z = p) \vee (z = y)))$ ✓
$((p \neq q) \wedge \forall z((z = p) \vee (z = q)))$

The un-ticked formula is a conjunction which gets expanded by an α rule. Then only the bottom formula $\forall z(z = p \vee z = q)$ is un-ticked. This gets expanded, by rule γ using the closed term p to give



$\exists x \exists y((x \neq y) \wedge \forall z((z = x) \vee (z = y)))$ ✓
$\exists y((p \neq y) \wedge \forall z((z = p) \vee (z = y)))$ ✓
$((p \neq q) \wedge \forall z((z = p) \vee (z = q)))$ ✓
$p \neq q$
$\forall z(z = p \vee z = q)$
$(p = p \vee p = q)$ ✓
$p = p$
$p = q$

The right branch is closed because it contains both $p \neq q$ and $p = q$. The universal formula has not been ticked and can be expanded again, this time using the closed term q. As there are no other closed terms in this language, this time we can tick this node.


$\exists x \exists y((x \neq y) \wedge \forall z((z = x) \vee (z = y)))$ ✓
$\exists y((p \neq y) \wedge \forall z((z = p) \vee (z = y)))$ ✓
$((p \neq q) \wedge \forall z((z = p) \vee (z = q)))$ ✓
$p \neq q$
$\forall z(z = p \vee z = q)$ ✓
$(p = p \vee p = q)$ ✓
$p = p$
$\underline{p = q}$
$\underline{q = p}$
$q = q$

Now the universal formula has been ticked because the only closed terms of our language are p and q. We have a completed tableau with an open branch. A model for the formula can be read off from this branch. It is the Herbrand structure. The domain consists of two points p and q. The interpretation of the predicates can be told by looking at the literals that hold on the open branch. Here we have $p \neq q$, so our structure is forced to have two distinct points.

EXERCISE 14 *Let L have no constants $C = \{\}$, functions $F = \{s^1\}$ and predicates $P = \{R^1, P^2\}$. For each formula construct a tableau and say if the formula is satisfiable or not.*

1. $\exists x R^1(x) \wedge \forall x \neg R^1(x)$
2. $\exists x(R^1(x) \wedge \forall y(R^1(y) \to (P^2(x, y) \leftrightarrow \neg P^2(y, y))))$ *(you will need to define a rule for $(\phi \leftrightarrow \psi)$ here. One way of doing this is to replace $(\phi \leftrightarrow \psi)$ by $(\phi \to \psi) \wedge (\psi \to \phi)$.)*
3. $\exists x R^1(x) \wedge \forall x(R^1(x) \to R^1(s^1(x)))$

Prove that every asymmetric binary relation P^2 is irreflexive. That is, prove that

$$(\forall x \forall y(P^2(x, y) \to \neg P^2(y, x)) \to \forall x \neg P^2(x, x))$$

is valid.

As with Hilbert systems, we write $\vdash \phi$ is there is a closed tableau for $\neg\phi$. Let Γ be any set of sentences. We write $\Gamma \vdash \phi$ is there is a closed tableau for $\neg\phi$ in which assumptions from Γ may be introduced at any node of the tableau.

4.16 Big theorems

Let S be an L-structure and let Σ be a set of L-sentences. We write

$$S \models \Sigma$$

if $S \models \sigma$ for each $\sigma \in \Sigma$. In this case we say that S is a model of Σ. In general, Σ could be an infinite set of sentences. Also, write

$$\Sigma \models \phi$$

if every model of Σ is also a model of ϕ, in other words $S \models \Sigma \Rightarrow S \models \phi$.

Let Γ be any set of sentences and let ϕ be any formula.

THEOREM 7 (Soundness) *If $\Gamma \vdash \phi$ then $\Gamma \models \phi$. In other words, if there is a closed tableau for $\neg\phi$ using assumptions from Γ then every model of Γ is also a model of ϕ.*

THEOREM 8 (Completeness) *If $\Gamma \models \phi$ then $\Gamma \vdash \phi$. So, if every model of Γ is a model of ϕ then there is a closed tableau for $\neg\phi$ using assumptions from Γ.*

THEOREM 9 (Compactness) *Let Γ be a possibly infinite theory (a theory is just a set of sentences). If $\Gamma \models \phi$ then there is a finite subset $\Gamma_0 \subseteq \Gamma$ such that $\Gamma_0 \models \phi$.*

PROOF:

> Suppose $\Gamma \models \phi$. By completeness, $\Gamma \vdash \phi$, so there is a closed tableau for $\neg\phi$ using assumptions from Γ. But a closed tableau is finite, so the tableau can only use finitely many assumptions from Γ. Let Γ_0 be the set of assumptions used in this closed tableau. Then $\Gamma_0 \vdash \phi$. By soundness, $\Gamma_0 \models \phi$. □

This is a powerful, but in a way obvious, theorem. It works equally well with a complete Hilbert system. If you can prove a formula from a possibly infinite set of assumptions, then your proof can only use finitely many of the assumptions, as proofs are only finitely long. Thus you can prove it from only finitely many assumptions.

COROLLARY 10 *Let Σ be a (possibly infinite) theory. If every finite subset of Σ is satisfiable (has a model) then Σ also is satisfiable.*

PROOF:

> Work backwards. Suppose that Σ is not satisfiable. Then $\Sigma \models \bot$ because every model of Σ (there are none) is a model of $\bot$. Hence, by the compactness theorem, there is a finite subset $\Sigma_0 \subseteq \Sigma$ such that $\Sigma_0 \models \bot$. So some finite subset of Σ is unsatisfiable. □

An example of the kind of thing you can prove using compactness is that there is no set of first-order formulas that define 'connectedness' in a graph. We'll prove this by contradiction.

So let L be the first-order language with a binary predicate E^2 (to represent the edge relation) and equality, two constants c and d, and no other non-logical symbols. Suppose Γ is a theory such that for any graph G, we have $\Gamma \models G$ if and only if G is a connected graph. Let $\sigma_n(x, y)$ be a formula that says there is no path from node x to node y whose length is less than or equal to n. Formally,

$$\begin{aligned} \sigma_1(x,y) &= \neg E^2(x,y) \\ \sigma_{n+1}(x,y) &= \neg\exists z(E^2(x,z) \wedge \sigma_n(z,y)) \end{aligned}$$

Consider the theory $\Sigma = \Gamma \cup \{\sigma_n(c,d) : \text{ all finite } n\}$. The sentences $\{\sigma_n(c,d) : \text{ finite } n\}$ say that there is no path of length less that n from c to d, for all finite n. This means there is no path at all from c to d, which means that any model of this set of sentences is a disconnected graph. Therefor $\Sigma = \Gamma \cup \{\sigma_n(c,d) : \text{ finite } n\}$ has no models at all (a structure cannot be both connected and disconnected). In symbols, $\Sigma \models \bot$.

Now consider a finite subset Σ_0 of $\Gamma \cup \{\sigma_n(c,d) : \text{ finite } n\}$. We have $\Sigma_0 = \Gamma_0 \cup \{\sigma_n(c,d) : n \in S\}$ for some finite subset Γ_0 of Γ and some *finite* set of natural numbers S. Pick a natural number N such that $s \in S \Rightarrow s < N$. This finite theory Σ_0 is actually satisfiable. For a model, take a connected graph G containing two nodes c, d, but the shortest path from c to d has length $N+1$. E.g. G could be a graph with nodes $\{0, 1, \ldots, N+1\}$ with edges between consecutive numbers only and $c = 0,\ d = N+1$. This graph is connected and the shortest path from 0 to $N+1$ has length $N+1$. So this is a model of Σ_0.

But this gives us a contradiction because we have an unsatisfiable theory Σ where every finite subset of Σ is satisfiable. This contradicts the corollary. We conclude that the theory Γ expressing connectedness does not exist.

EXERCISE 15 *Let L be a language including the equality symbol $=$.*

1. *Define a theory Σ such that every infinite structure is a model of Σ but no finite structure is a model of Σ. [Hint: write down a formula ϕ_n that says that there are at least n distinct elements.]*
2. *Prove by compactness that there is no theory Γ defining exactly the finite structures. I.e. there is no Γ such that all finite structures are models of Γ but no infinite structures are models of Γ.*

THEOREM 11 (Gödel 1931) *Let Λ be a formal, recursively enumerable logic, sufficient for arithmetic (i.e. there should be constants $0, 1$, binary function symbols $+, \times$ and appropriate axioms for these). Then if Λ is sound (it cannot prove any false statements about arithmetic, like $1 = 2$) then it is not complete — there are true statements of arithmetic for which there exists no proof.*

See [Göd] PROOF:

Here is a sketch of a proof.

- Every formula in the language of Λ is a string and every tableau can be written as a string. Hence every formula and every tableau can be identified by a number (its *Gödel number*). E.g.

$$+(x, y) = -x$$

could be written as a string of octal ASCII codes as

$$053\ 050\ 170\ 053\ 171\ 051\ 075\ 055\ 170$$

(I used the UNIX command "od -b" for this). So the octal number

$$053050170053171051075055170$$

represents the formula $+(x, y) = -x$. And it is possible to work out the formula from the Gödel number. Similarly, we can identify each tableau by a unique Gödel number.

- We can write a formula, with two free variables, $\theta(n, m)$ which says "the tableau with Gödel number n proves the formula with Gödel number m". Also, if formula number m has one free variable, say it is $A_m(x)$, then we can write

$$\phi(n, m, q) = \text{"tableau } n \text{ proves } A_m(q)\text{"}$$

- Consider the formula

$$\neg\exists n\phi(n, q, q)$$

This is true in $\mathbb{N}$ if there is no tableau that proves $A_q(q)$. The formula $\neg\exists n\phi(n, x, x)$ has one free variable (x) and it must have a certain Gödel number, say n_0. So

$$A_{n_0}(x) = \neg\exists n\phi(n, x, x)$$

This is true in $\mathbb{N}$ iff there is no tableau proof of $A_x(x)$.

- Finally, consider

$$A_{n_0}(n_0)$$

This is true iff there is no tableau proof of $A_{n_0}(n_0)$ (!)

So, if there is a proof of $A_{n_0}(n_0)$ then $A_{n_0}(n_0)$ is false, in which case our proof system is not sound. On the other hand, if there is no proof of $A_{n_0}(n_0)$ then it is true, in which case our proof system is not complete. □

5 Modal logic

Modal logic goes back to Aristotle and was studied intensively in the medieval period. Medieval scholars such as Avicenna (Ibn Sina), Anselm, Abelard, Aquinas and others with name beginning with A were interested in truth modalities — what *must* be true, what *could* be true, what is *impossible*. See the lecture slides for an outline of Anselm's *ontological argument* for the necessity of the existence of God. Modern (twentieth century) versions of modal logic apply very widely, to reasoning about *possibility* but also *belief* and *knowledge* (epistemic logic), *time* (temporal logic) and program transitions (program logics). Basic propositional modal logic, which we study here, can be thought of as intermediate between propositional logic and first-order logic. Although there are no explicit quantifiers $\forall, \exists$ in basic modal logic, one can express more than propositional logic using additional operators $\Box$ and $\Diamond$ (think of $\Box\phi$ as meaning 'ϕ must be true', and $\Diamond\phi$ as meaning 'ϕ could be true'). Yet in many cases (unlike propositional logic, there are many propositional modal logics) the set of valid modal formulas turns out to be decidable, giving it a strong computational advantage over first-order logic.

5.1 Syntax

Here is the syntactic definition of a formula.

$$form ::= prop|\neg form|(form \vee form)|\Box form|\Diamond form$$

we may also write $\phi\wedge\psi$ as an abbreviation of $\neg(\neg\phi\vee\neg\psi)$ and $\phi \to \psi$ as an abbreviation of $\neg\phi\vee\psi$. so, e.g., $\neg\Box(p \to \Diamond(p \wedge \neg q))$ is a modal formula.

EXERCISE 16 *The operator $\Box$ is intended to mean 'it must be that ...' and the operator $\Diamond$ is intended to mean 'it could be that ...'. We have no operator for 'it is impossible that'. How would you express 'p is impossible'? Is there more than one way to express this?*

5.2 Semantics

The idea with propositional modal logic is that propositions are not simply true or false; their truth value depends on where they are evaluated. Now we give the formal semantics and give meaning to the new modal operators $\Box, \Diamond$.

A *modal frame* or *Kripke frame* $\mathcal{F} = (W, R)$ consists of a set W (the set of 'possible worlds') and a binary relation $R \subseteq W \times W$ (the 'accessibility relation'). The idea here is that a world w' is accessible from a world w iff $(w, w') \in R$. Then a *valuation* is a mapping $v : props \to \wp(W)$, i.e. v assigns a set of worlds $v(p)$ for each proposition p. The idea is that p is true at each of the worlds in $v(p)$ and false at all other worlds. A frame $\mathcal{F}$ together with a valuation v forms a *model* or *Kripke model* $(\mathcal{F}, v)$.

Now all formulas can be evaluated at a world $w \in W$ in a model as follows.

$$\begin{aligned}
\mathcal{F}, v, w \models p &\iff w \in v(p)\\
\mathcal{F}, v, w \models \neg\phi &\iff \mathcal{F}, v, w \not\models \phi\\
\mathcal{F}, v, w \models (\phi\vee\psi) &\iff \mathcal{F}, v, w \models \phi \text{ or } \mathcal{F}, v, w \models \psi\\
\mathcal{F}, v, w \models \Box\phi &\iff \mathcal{F}, v, w' \models \phi \text{ for all } w' \text{ with } (w, w') \in R\\
\mathcal{F}, v, w \models \Diamond\phi &\iff \mathcal{F}, v, w' \models \phi \text{ for some } w' \text{ with } (w, w') \in R
\end{aligned}$$

EXERCISE 17 *Let $\mathsf{N} = (\mathbb{N}, R)$ be the frame whose set of worlds is the natural numbers $\mathbb{N}$, and the accessibility relation R is $\{(m, n) : m < n\}$. Let $v(p)$ be the set of all even numbers. Then $\mathbb{N}, v, n \models p$ iff n is even. Which of these formulas are true when evaluated at 0 in this model?*

1. $\Box p$

2. $\Diamond p$

3. $\Box\Diamond p$

4. $\Diamond\Box p$

5.3 Validity and Satisfiability

With first order logic we had two separate definitions for 'valid in a given structure' and 'valid over all structures'. It is similar with modal logic. Given a particular frame $\mathcal{F} = (W, R)$ and a modal formula ϕ, we may say that ϕ is valid over $\mathcal{F}$ if for all valuations $v : \mathit{props} \to \wp(W)$ and all $w \in W$ we have $\mathcal{F}, v, w \models \phi$. This means, regardless of the valuation and regardless of the point of evaluation, the formula is always true in this particular frame. We write

$$\mathcal{F} \models \phi$$

in this case. Also, we say that ϕ is satisfiable in $\mathcal{F}$ if there is some valuation v and some world $w \in W$ such that $\mathcal{F}, v, w \models \phi$.

EXERCISE 18 *Prove that ϕ is valid in $\mathcal{F}$ if and only if $\neg\phi$ is not satisfiable in $\mathcal{F}$.*

EXERCISE 19 *Let $\mathsf{N} = (\mathbb{N}, R)$ be the frame defined in the previous exercise. Which of the following modal formulas are valid in N?*

1. $\Diamond p$

2. $\Box(p \to \Diamond\neg p)$

3. $\Box(p \vee \neg p)$

4. $\Box p$.

If a modal formula is valid in all possible frames $\mathcal{F}$ then we may say that ϕ is valid (over all frames) and write

$$\models \phi$$

Also, a formula is satisfiable if it is true at some point, under some evaluation, in some frame. A formula is valid if and only if its negation is not satisfiable.

EXERCISE 20 *Which of the following formulas are valid over all frames?*

1. $\Diamond p$

2. $\Box(p \vee \neg p)$

3. $\Box(p \vee q) \to (\Box p \vee \Box q)$

4. $(\Box p \vee \Box q) \to \Box(p \vee q)$

How can you tell whether a given modal formula is valid or not? In fact, is it decidable?

Quite often, we want to restrict to a certain class $\mathcal{K}$ of frames, e.g. the class of symmetric frames, where $(w, v) \in R \iff (v, w) \in$. We say that ϕ is valid over $\mathcal{K}$ if for all frames $\mathcal{F} \in \mathcal{K}$ we have $\mathcal{F} \models \phi$, and in this case we write

$$\mathcal{K} \models \phi$$

A formula ϕ *defines* a class of frames $\mathcal{K}$ if $\mathcal{K} = \{\mathcal{F} : \mathcal{F} \models \phi\}$, i.e. ϕ is valid over $\mathcal{K}$ but not valid on any frame outside $\mathcal{K}$. The formula $p \to \Box\Diamond p$ defines the class of all symmetric frames. To prove this, first we show that the formula is valid over symmetric trames. In an arbitrary symmetric frame $\mathcal{F} = (W, R)$ under an arbitrary valuation $v : \mathit{props} \to \wp(W)$ and an arbitrary $w \in W$ suppose $\mathcal{F}, v, w \models p$, i.e. $w \in v(p)$. We must prove that $\mathcal{F}, v, w \models \Box\Diamond p$. In order to prove that, let $w' \in W$ be any world accessible from w, i.e where $(w, w') \in R$. We must prove that $\mathcal{F}, v, w' \models \Diamond p$.

But, since R is symmetric, we know that $(w', w) \in R$ and we are assuming that $\mathcal{F}, v, w \models p$ so $\mathcal{F}, v, w' \models \Diamond p$ is necessarily true, as required. Now consider a not symmetric frame (W, R), so there is $(u, w) \in R$ but $(w, u) \notin R$ for some $u, w \in W$. Let v be the valuation $v : p \to \{u\}$ making p true at u but nowhere else. Then $(W, R), v, u \models p$. Since there is no arrow $(w, u) \in R$ we have $\mathcal{F}, v, w \not\models \Diamond p$. Since $(u, w) \in R$ we have $\mathcal{F}, v, u \not\models \Box\Diamond p$, and so $\mathcal{F}, v, u \not\models (p \to \Box\Diamond p)$. It follows that $p \to \Box\Diamond p$ is not valid over (W, R).

EXERCISE 21 *Let $\mathcal{R}$ be the class of all* reflexive *frames $\mathcal{F} = (W, R)$ where for all $w \in W$ we have $(w, w) \in R$. Write down a modal formula ϕ such that $\mathcal{R} \models \phi$, but where ϕ is not valid over any frame containing an irreflexive point.*

Also consider the class $\mathcal{T}$ of transitive *frames $\mathcal{F} = (W, R)$ where for all $u, v, w \in W$ if $(u, v) \in R$ and $(v, w) \in R$ then $(u, w) \in R$. Write down a formula ϕ that defines $\mathcal{T}$, i.e. for an arbitrary frame $\mathcal{F}$ we have $\mathcal{F} \in \mathcal{T} \iff \mathcal{F} \models \phi$.*

5.4 p-morphisms

A p-morphism f from a frame (W, R) to a frame (W', R') is a map $f : W \to W'$ such that

- for all $v, w \in W$ if $(v, w) \in R$ then $(f(v), f(w)) \in R'$ and
- for all $v \in W$ and all $u \in W'$ if $(f(v), u) \in R'$ then there is $w \in W$ where $f(w) = u$ and $(v, w) \in R$.

Given two Kripke models $((W, R), v)$ and $((W', R'), v')$ and a p-morphsim from (W, R) to (W', R'), if for every proposition p and $w \in W$ we have $w \in v(p) \iff f(w) \in v'(p)$ then f is called a model-p-morphism

LEMMA 12 *If f is a model-p-morphism from model $((W, R), v)$ to model $((W', R'), v')$ then for any formula ϕ and any $w \in W$,*

$$(W, R), v, w \models \phi \iff (W', R'), v', f(w) \models \phi$$

This lemma is proved by structured formula induction over ϕ. In the base case, ϕ is a proposition p and the lemma holds because f is a model-p-morphism. The induction steps use the fact that f is a frame p-morphsim from (W, R) to (W', R').

LEMMA 13 *Let f be a p-morphism from frame (W, R) to frame (W', R'). Suppose f is surjective, i.e. the range of f covers the whole of W'. Then for all formulas ϕ,*

$$(W, R) \models \phi \Rightarrow (W', R') \models \phi$$

PROOF:

Suppose $(W, R) \models \phi$. We have to prove that $(W', R') \models \phi$, i.e. for all valuations $v' : \mathit{props} \to \wp(W')$ and all worlds $w' \in W'$ we have $(W', R'), v', w' \models \phi$. Given a valuation $v' : \mathit{props} \to \wp(W')$ define a valuation $v : \mathit{props} \to \wp(W)$ by letting $v(p) = \{w \in W : f(w) \in v'(p)\}$. Then f is a model-p-morphism from $((W, R), v)$ to $((W', R'), v')$. Since f is surjective, there is some $w \in W$ such that $f(w) = w'$. We are supposing $(W, R) \models \phi$ so $(W, R), v, w \models \phi$. Since f is a model-p-morphism it follows from the previous lemma that $(W', R'), v', w' \models \phi$, which is what we set out to prove. $\Box$

A frame (W, R) is *irreflexive* if for all $w \in W$ we have $(w, w) \notin R$.

EXERCISE 22 *Define a frame (as small as possible) which is neither reflexive nor irreflexive.*

p-morphisms can be used to prove that there is no formula that defines the class of irreflexive frames. Consider two frames $\mathcal{F} = (\{u, w\}, \{(u, w), (w, u)\})$ and $\mathcal{F}' = (\{u'\}, \{(u', u')\})$, as shown here.

$$\begin{tikzcd}
w \arrow[d, bend left] & & u' \arrow[loop above] \\
u \arrow[u, bend left] & &
\end{tikzcd}$$

The first frame is irreflexive with two worlds, the second is reflexive with just a single world. The map $f : \{u, w\} \to \{u'\}$ is a surjective p-morphism from $\mathcal{F}$ to $\mathcal{F}'$. Suppose for contradiction that ϕ defines the class of irreflexive frames. Since $\mathcal{F}$ is irreflexive, $\mathcal{F} \models \phi$. Since f is a surjective p-morphism it follows that $\mathcal{F}' \models \phi$, but $\mathcal{F}'$ is not irreflexive, contradicting our assumption that ϕ defined the class of all irreflexive frames.

5.5 Modal tableau

The set of all modal formulas that are valid over all frames is called K (after Kripke) and a basic computational problem is to work out whether $\phi \in K$ or not, for a given modal formula ϕ. Note that $\phi \in K$ if and only if $\neg\phi$ is not satisfiable in any frame. We can test whether $\neg\phi$ is satisfiable in some frame by constructing a tableau. You can do this with a tree-structure as we did for propositional and first order tableaus, but here instead it is convenient to consider a tableau as a list of alternative labelled Kripke frames (each of these alternatives would correspond to a branch in a tree-like version of the tableau). In the intial tableau we have a list of just a single alternative, and this is a frame $\mathcal{F}_0 = (\{w_0\}, \emptyset)$ with a single world w_0 and empty accessiblity relation. Each world in each of the alternative frames is labelled by a set of formulas. Formally, a labelled frame $(\mathcal{F}, \lambda)$ consists of a frame $\mathcal{F} = (W, R)$ and a function $\lambda : W \to \mathsf{Form}$ where Form is the set of all formulas. A tableau is a finite list of labelled frames $\mathcal{T} = [(\mathcal{F}_0, \lambda_0), (\mathcal{F}_1, \lambda_1), \ldots, (\mathcal{F}_k, \lambda_k)]$. The intial tableau is

$$\mathcal{T}_0 = [((\{w_0\}, \emptyset), \lambda_0)]$$

where $\lambda_0(w_0) = \{\neg\phi\}$. So the intial world w_0 in the initial frame is labelled by the singleton set $\{\neg\phi\}$ in a tableau for $\neg\phi$.

A labelled frame $(\mathcal{F}, \lambda)$ is *closed* if there is a world w in the frame and a proposition p such that $\{p, \neg p\} \subseteq \lambda(w)$. At any stage as we expand the tableau, if a labelled frame is closed we may delete it from the list of alternatives. A labelled frame is *completed* if the only formulas labelling worlds are literals p or $\neg p$. If a tableau is not completed we may expand it by picking one of the alternative labelled frames $(\mathcal{F}, \lambda)$ and picking a non-literal formula $\psi \in \lambda(w)$ for some world w in the frame. How we exand the formula ψ depends on whether it is an α, β, δ or γ.

α If ψ is an α formula with expansion formulas α_1, α_2 then replace α by α_1 and α_2, i. e. $\lambda(w) := \lambda(w) \cup \{\alpha_1, \alpha_2\} \setminus \{\alpha\}$, but $\lambda(w')$ is unchanged for $w' \neq w$.

β If ψ is a β formula with expansion formulas β_1, β_2 then replace the labelled frame $(\mathcal{F}, \lambda)$ by two alternative labelled frames, the first with β_1 in place of β in $\lambda(w)$ and the second with β_2 in place of β. In this case (and only this case) the list of alternative labelled frames grows by one.

δ A formula is a δ formula if it has the form $\Diamond\theta$ or $\neg\Box\theta$. The expansion formula δ' is θ in the first case and $\neg\theta$ in the second. To expand the formula you change the frame $\mathcal{F} = (W, R)$ by adding a single new world $w \notin W$ (so $W := W \cup \{w\}$) and letting $\lambda(w) = \{\delta'\}$, i.e. put the expansion formula into the label of the single new world. Also, add an R-arrow from w to w', i.e. $R := R \cup \{(w, w')\}$.

γ A formula is a γ formula if it has the form $\Box\theta$ or $\neg\Diamond\theta$, the expansion formula γ' is θ in the first case, $\neg\theta$ in the second. To expand a γ formula, the frame $\mathcal{F} = (W, R)$ does not change only the labelling λ. If $\psi \in \lambda(w)$ is a γ formula with expansion formula γ' then for all worlds $w' \in W$ if $(w, w') \in R$ then add the expansion formula to $\lambda(w')$, i. e. $\lambda(w') := \lambda(w') \cup \{\gamma'\}$.

Since these expansion rules always replace a formula by a shorter formula this must terminate eventually (how long in the worst case?) Termination occurs if either the list of alternative labelled frames becomes empty, in which case $\neg\psi$ is not satisfiable, or there are remaining open completed labelled frames. If $((W, R), \lambda)$ is an open completed frame and w_0 is the initial world then you can use λ to define a valuation v where $v(p) = \{w \in W : p \in \lambda(w)\}$. You can check that

$$(W, R), w_0 \models \neg\phi$$

so $\neg\phi$ is satisfiable if the tableau does not close. Therefore, this modal algorithm is sound and complete for K.

Reflexive frames Instead of working with the class of all frames, you may restrict to the reflexive frames. The formula $\Box p \to p$ is not generally valid, but it is valid over reflexive frames. In fact, a frame is reflexive if and only if $\Box p \to p$ is valid in that frame. The set of formulas valid over all reflexive frames is called T (so $\Box p \to p$ belongs to T). You can use a tableau to test whether a formula ϕ is in T by starting a new tableau with a single world labelled $\{\neg\phi\}$ and expanding it as before, but this time the initial frame is $(\{w\}, \{(w, w)\})$ (this time reflexive) and whenever a new world w' is added to a frame we also include (w', w') in the accessibility relation R. For γ expansions, if $\Box\theta \in \lambda(w')$ is picked, you must also include θ in $\lambda(w')$, otherwise the procedure is unchanged. Here are some other classes of frames and the formulas that define them

Class of Frames	Defining Formulas	Name of Logic
All frames	$\top$	K
Reflexive frames	$\Box p \to p$	T
Symmetric frames	$p \to \Box\Diamond p$	?
Transitive frames	$\Box\Box p \to \Box p$	K4
Equivalence relation frames	all above	S5

5.6 Temporal Logic

This is rather like modal logic except formulas are evaluated at points of time instead of at different worlds. Instead of a modality to take you to other possible worlds we have two inverse modalities: one for accessing the future and one for the past. Of course there is a connection between these two.

$$form ::= prop | \neg form | (form \vee form) | \mathbf{F} form | \mathbf{P} form$$

The temporal operator $\mathbf{F}$ is a kind of diamond operator and means 'at some time in the future'. There is a dual operator $\mathbf{G}$ meaning 'at all times in the future' but $\mathbf{G}\phi$ can be expressed as $\neg\mathbf{F}\neg\phi$. Similarly, $\mathbf{P}$ means 'at some time in the past' and it has a dual $\mathbf{H}$ meaning ' at all times in the past'.

For the semantcs, a temporal frame $(T, \prec)$ consists of a set T of time points and a transitive binary relation $\prec$ over T (this binary relation replaces R in Kripke frames). A valuation v is a map $v : props \to \wp(T)$. A temporal formula ϕ may be evaluated at a time point $t \in T$ with propositional connectives evaluated as normal, and with the temporal operators evaluated by,

$$\begin{aligned}
(T, \prec), v, t \models \mathbf{F}\phi &\iff \exists s\, (t \prec s \wedge (T, \prec), v, s \models \phi) \\
(T, \prec), v, t \models \mathbf{G}\phi &\iff \forall s\, (t \prec s \to (T, \prec), v, s \models \phi) \\
(T, \prec), v, t \models \mathbf{P}\phi &\iff \exists s\, (s \prec t \wedge (T, \prec), v, s \models \phi) \\
(T, \prec), v, t \models \mathbf{H}\phi &\iff \forall s\, (s \prec t \to (T, \prec), v, s \models \phi)
\end{aligned}$$

Example Let the flow of time be $(\mathbb{Z}, <)$, i.e. the time points are integers. Let v be a valuation with $v(p)$ equal to the set of all even integers and let $v(q)$ be all integers greater than 10. Calculate the truth of the following formulas when evaluated at 0.

1. p
2. $p \vee q$
3. $\mathbf{F}(p \wedge q)$
4. $\mathbf{F}(p \wedge \mathbf{G}q)$
5. $\mathbf{GF}p$
6. $\mathbf{FG}p$.

With modal and temporal logics you can write down axioms and rules, or define tableau expansion rules to prove validities. In many cases the logics turn out to be decidable.

Observe that $p \to \mathbf{GP}p$ is valid over all temporal frames, since starting from any time point t if you go anywhere into the future you will always see t in your past. Similarly $p \to \mathbf{HF}p$ is valid.

EXERCISE 23 *Consider the temporal frame* $(\mathbb{N}, <)$ *where* $<$ *is the usual strict ordering of natural numbers. Let* v *be a valuation where* $v(p) = \{0, 2, 4, \ldots\}$*, the even natural numbers. Which of the following are true at* 0*?*

1. $\mathbf{G}p$

2. $\mathbf{GF}p$

3. $\mathbf{FH}(\neg p)$.

EXERCISE 24 *A temporal frame* $(T, \prec)$ *is reflexive if* $\prec$ *is reflexive. Write down a temporal formula that is valid in any reflexive frame, but not valid in any other frames.*

5.7 Propositional Dynamic Logic (PDL)

The modal logics above are called *uni-modal* because there is a single modality, $\Box$ and its dual $\Diamond$. In a *multi-modal* logic there may be several modalities $\Box_i$ and their duals $\Diamond_i$ with distinct accessibility relations R_i, for $i < k$. So a Kripke frame becomes $(W, \overline{R})$ where $\overline{R} = (R_0, R_1, \ldots, R_{k-1})$ is a sequence of k binary relations over W. Given a valuation $v : \mathit{props} \to \wp[(W)$ and a world $w \in W$ we may evaluate multimodal formulas using

$$(W, \overline{R}), v, w \models \Diamond_i \phi \iff \text{there is some } w' \in W \text{ wiith } (w, w') \in R_i \text{ and } (W, \overline{R}), v, w' \models \phi$$
$$(W, \overline{R}), v, w \models \Box_i \phi \iff \text{whenever } (w, w') \in R_i \text{ we have } (W, \overline{R}), v, w' \models \phi$$

In PDL you have a set of actions and a separate modality for each action. So if α is an action then you get a diamond $<\alpha>$ and a box $[\alpha]$. You have a set of atomic actions called Π. Formulas and actions are defined by mutual recursion.

$$\begin{aligned} \mathit{fmla} &::= \mathit{prop} \mid \neg \mathit{fmla} \mid (\mathit{fmla}_1 \vee \mathit{fmla}_2) \mid <\mathit{act}>\mathit{fmla} \mid [\mathit{act}]\mathit{fmla} \\ \mathit{act} &::= \Pi \mid (\mathit{act}_1; \mathit{act}_2) \mid (\mathit{act}_1 + \mathit{act}_2) \mid \mathit{act}^* \mid \mathit{fmla}? \end{aligned}$$

So for formulas you have all of propositional logic but you can also use modal operators $<\alpha>$, $[\alpha]$ for any action α. You can also use symbols $\wedge, \to$ as propositional abbreviations, as before. The set of actions is built from atomic actions in the way of regular expressions. The action $\mathit{act}_1; \mathit{act}_2$ means first do act_1 and then do act_2. The action $\mathit{act}_1 + \mathit{act}_2$ means non-deterministically do either act_1 or act_2. The action act^* means repeat the action act zero or more times. Finally, given a formula $fmla$ the action $fmla?$ is the action that skips and proceeds if $fmla$ is true, else it fails (this is called a *test*).

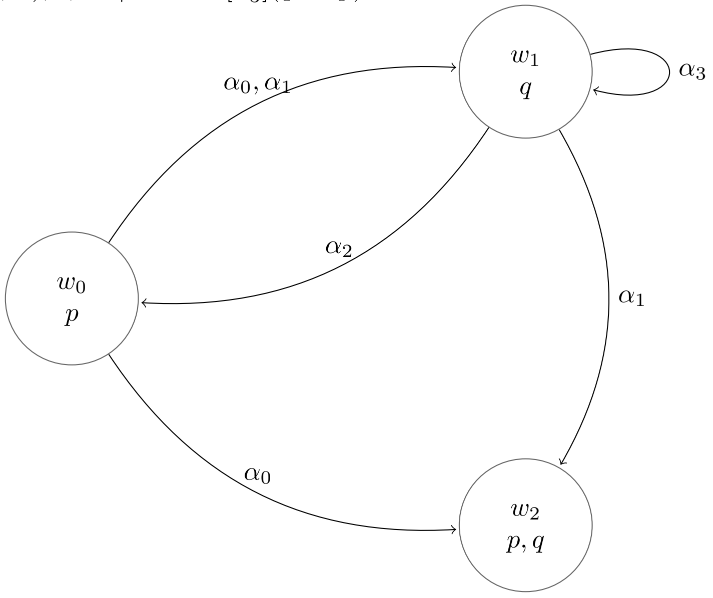
For the semantics of PDL we use *labelled transition systems* (LTS). This is like a Kripke frame for a multimodal logic, but only the accessibility relations for atomic actions are specified. Formally, an LTS (W, R, v) consists of a set of worlds W, a function $R : \Pi \to \wp(W \times W)$ (so each atomic action is interpretted by R as a binary relation over W) and a valuation $v : props \to \wp(W)$. The action interpretation R can be extended from atomic actions to all actions by recursion, as follows.

$$\begin{aligned}
R(\alpha;\beta) &= \{(w_1, w_2) \in W \times W : \exists w_3((w_1, w_3) \in R(\alpha) \wedge (w_3, w_2) \in R(\beta))\} \\
R(\alpha + \beta) &= R(\alpha) \cup R(\beta) \\
R(\alpha^*) &= \bigcup_{n\in\mathbb{N}} R(\alpha^n) \\
R(\phi?) &= \{(w, w) \in W \times W : (W, R), v, w \models \phi\}
\end{aligned}$$

and formulas may be evaluated by

$$\begin{aligned}
(W, R), v, w \models p &\iff w \in v(p) \\
(W, R), v, w \models \neg\phi &\iff (W, R, v), w \not\models \phi \\
(W, R), v, w \models (\phi_1 \vee \phi_2) &\iff (W, R, v), w \models \phi_1 \text{ or } (W, R, v), w \models \phi_2 \\
(W, R), v, w \models <\alpha>\phi &\iff \exists w' \in W\ ((w, w') \in R(\alpha) \text{ and } (W, R, v), w' \models \phi) \\
(W, R), v, w, \models [\alpha]\phi &\iff \forall w' \in W\ (\text{if } ((w, w') \in R(\alpha) \text{ then } (W, R, v), w \models \phi)
\end{aligned}$$

Example and Exercise Consider the LTS shown below. There are three worlds $W = \{w_0, w_1, w_2\}$, the valuation v is given by $v(p) = \{w_0, w_2\}$, $v(q) = \{w_1, w_2\}$, $v(r) = \emptyset$. The atomic actions $\alpha_0, \ldots, \alpha_3$ are interpretted by R as shown in the diagram. Observe that every α_3 arrow from w_1 leads to w_1, where $q \wedge \neg p$ is true. So $(W, R), v, w_1 \models [\alpha_3](q \wedge \neg p)$. Hence at w_0, since there is an α_0 arrow (w_0, w_1), we have $(W, R), v, w_0 \models <\alpha_0>[\alpha_3](q \wedge \neg p)$, indeed $(W, R), v, w_0 \models <\alpha_0>[\alpha_3^*](q \wedge \neg p)$.



EXERCISE 25 *Which of the following hold are true at* $(W, R), v, w_0$*?*

1. $[\alpha_0]p$
2. $[\alpha_0]q$

3. $[(\alpha_1;\alpha_2)^*]p$

4. $[\alpha_0]{<}\alpha_3{>}(p \vee q)$.

If you think of the worlds as program states, the atomic actions as primitive program transitions, then you can write program specifications using PDL formulas. A tableau for PDL can be used to prove that a program meets its specifications.

6 Other Logics

Although predicate logic is a very fine logic, it is not perfect. Firstly, although it is very expressive, its expressivity is limited. We saw that a corollary of the compactness theorem is that first-order logic cannot express connectedness in a graph. Nor can it pick out the finite structures from a class of structures. Secondly, it is an undecidable logic. Dealing with these two difficulties pushes you in opposite directions.

6.1 Higher-order logics

To add expressive power, there are a number of alternatives. In *second-order logic* you are allowed to quantify over predicates as well as elements. And there can be predicates of predicates too. So to define connectedness in a graph, take a language with a binary relation E (edge) and a binary predicate variable X.

$$\forall X[\forall x\forall y[((E(x,y)\vee(x=y))\to X(x,y))\wedge\exists z((E(x,z)\wedge X(z,y))\to X(x,y))]\to\forall x\forall y X(x,y)]$$

The formula says that for any binary relation X if X is reflexive and contains all the edges and if X is closed under 'adding an edge' then X holds on all pairs of nodes. This is supposed to hold for all possible X, in particular it holds when X is the transitive closure of the edge relation. This means that the transitive closure of the edge relation (the set of pairs connected by a path) holds between any pair of nodes. Hence the formula implies that the graph is connected. And conversely, this formula will hold in any connected graph.

In third-order logic you can quantify over predicates of predicates, etc. etc.

The problem with second order logic is that there is no complete axiomatisation possible. So the logic is not even semi-decidable. This means there is no systematic way of telling whether second-order formulas are valid or not. However, there is a lot of research into well-behaved fragments of second-order logic with better computational behaviour. An example of such a fragment is called *fixpoint* logic. This logic does not have the full second-order power but it includes a fixpoint operator that, roughly, calculates the transitive closure of any relation. The transitive closure of the edge relation in a graph is the binary relation which says whether two points are connected to each other or not. Thus fixpoint logic can express connectedness in a graph. The fixpoint operator is the appropriate algebraic operator to model recursive programming constructs, like `while` loops.

Another logic that increases the expressive power is called *infinitary* logic. In this kind of logic you can form infinite conjunctions and disjunctions.

There are many other logics defined for different purposes. *Epistemic* logic is a kind of modal logic for expressing knowledge and belief. So you can write down formulas stating that 'A believes that B knows the answer', etc. Epistemic logic is often used to model the behaviour of agents.

In non-monotonic logic you can add more and more assertions, holding them all to be true only tentatively. When you discover a contradiction you do not reject the entire theory but instead you might withdraw just a few of the statements you have collected — enough to remove the contradiction, but not more than that. Non-monotonic logic is useful for handling large databases which are likely to contain inconsistencies. If you have come across *truth maintenance schemes* then you have come across a type of non-monotonic logic. *Para-consistent* logic is another kind of logic that is able to cope with inconsistencies.

References

[Fre72] G Frege. *Conceptual notation (Begriffschrift), and related articles.* Oxford : Clarendon Press, 1972. English translation.

[Göd] K Gödel. Gödel's incompleteness theorems. `http://en.wikidepdia.org/wiki/Godels_Incompleteness_Theore`