Logic

Roland Meyer

TU Kaiserslautern

Summer Term 2014

Logic

Lecture:
Mi 11.45 - 13.15 Uhr 52-207

- Informations
  http://concurrency.informatik.uni-kl.de/teaching.html
- The lecture is based on the German script *Einführung in die Logik*
- Links to English lecture notes can be found on the lecture website

Organizational Matters

Exercise sheets:

- One sheet every two weeks
- Sheets are handed out on Wednesday, and are to be handed in on Friday (one week later) in the box near room 401 and the SoftTech workgroup
- Groups of three people, first sheet is handed out today

Exercises:

- Alternating sheet exercises and presence exercises
- First exercise: next week (presence exercise)
- Registration starting 14:00 today, via the STATS system
- Tutors: Martin Köhler, Jonathan Kolberg, Elisabeth Neumann, Albert Schimpf

Requirements for admission to the final exam:

- Participation in the exercises (mandatory)
- At least 60% of the exercises on the sheets solved with +
- Presentation of one solution at the blackboard
- Passed the midterm exam (in the 1st or 2nd try)

Contents

4. Foundations of Predicate Logic
   - Syntax
   - Semantics
   - Substitution
   - Normal Forms
   - Herbrand Theory
   - Semi-Decidability of Validity
   - Lower Bound for Validity
   - Compactness Theorem of First-Order Logic
5. Deductive Perspective on Predicate Logic
   - Logical Consequence
   - The Deductive System $\mathcal{F}$
   - First Order Theories
   - Axiomatization
6. Algorithmic Perspective on Predicate Logic
   - Semantic Tableaux
   - Unification
   - Resolution

Introduction

Methods for solving problems with the help of computers
Formalization

- **Logic:** Science of sound reasoning or Science of the formal relations between thought contents.

Central questions: Truth and provability of statements $\rightsquigarrow$ **mathemathical logic**.

- **Logic in computer science**:
  - **Propositional logic**: Boolean algebra. Logical circuits (control systems), optimization. SAT can be found everywhere.
  - **Predicate logic**: Reasoning about data (AI, IS, SE).
  - **Modal and temporal logic**: Specification and verification (hardware, since 2000 software).

Logic in Computer Science

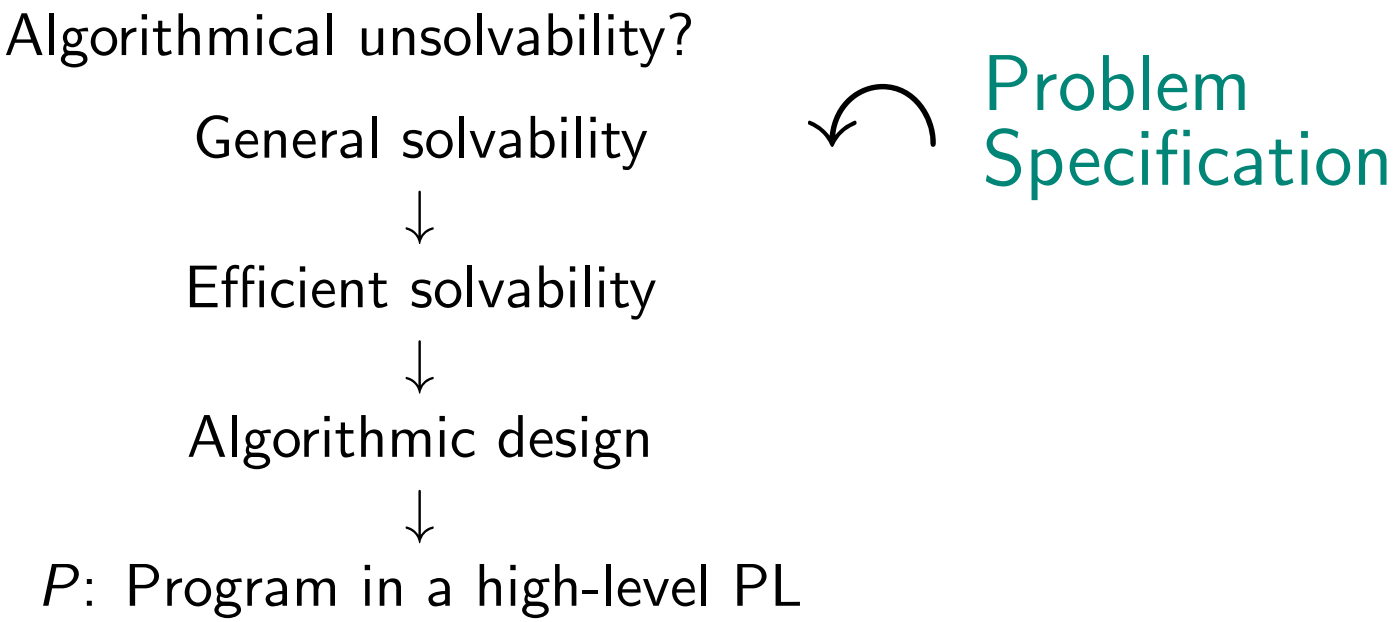
1. Semantics of programming languages (Hoare logic).
2. Specification of functional properties.
3. Verification process in software development.
   Proofs of program properties.
4. Representation of data (Predicate Abstraction).
5. Dedicated programming languages (PROLOG)

- **Automation of logical reasoning**
  1. Mechanized proofs (Methods,...)
  2. Foundations of information systems (Processing of knowledge, reasoning,...)

Requirements

1. **Mathematical foundations**. Sets, relations, functions. Formalizations: Mathematical proofs, mathematical language, i.e. usage and meaning of the common operators of naïve logic. Also the meaning of **not, and, or, if . . . then, if and only if, exists, for all**
2. **Foundations for the description of formal languages**. Grammars or more generally **calculi** (set of objects and rules for the generation of new objects from existing objects), generation of sets, relations and functions, closure operators (closure of sets with respect to relations).
3. **Concepts of calculability**, i.e. decidable, and recursively enumerable sets, existence of undecidable sets and uncalculable functions.

Computation Models / Programming Languages



Syntactic and Semantic Verification

- **Syntactic analysis**
  Chomsky hierarchy of languages
  Context-free languages
  Grammars / production process

- **Program verification**
  Does P work as expected?
  (Requirements) specification and (program) verification.

Typical Expressions

- $(x+1)(y-2)/5$ Terms as identifiers of objects.
- $3+2=5$ Equations as special formulae
- "'29 is (not) a prime"' Statement.
- "'$3+2=5$ and 29 is not a prime"' Statement.
- "'if 29 is not a prime, then $0=1$ is true"' Statement.
- "'every even number larger than two is the sum of two primes"' Statement.
- $2 \leq x$ and $(\forall y \in \mathbb{N})$
  $((2 \leq y \text{ and } y+1 \leq x) \rightarrow \text{not}(\exists z \in \mathbb{N}) y * z = x)$ Statement.

Typical Expressions (Cont.)

- $(\forall X \subseteq \mathbb{N})(0 \in X \wedge (\forall x \in \mathbb{N})(x \in X \to x + 1 \in X) \to X = \mathbb{N})$
  Principle of mathemathical induction.
- $(\forall X \subseteq \mathbb{N})(X \neq \emptyset \to X$ has a minimal element)
  Every nonempty set of natural numbers contains a minimal element.

Binary logic Every statement is either **true** or **false**.

- There are also other options (many-valued logic).
- First-order predicate logic (PL1): Only properties of elements and quantification of element variables are allowed.

Chapter I

Propositional Logic

Propositional Logic

- Structure of propositions $\rightsquigarrow$ **Syntax**

- Meaning of propositions $\rightsquigarrow$ **Semantics** **true** (1), **false** (0)

Syntax of Propositional Logic

Definition 1.1 (Syntax)

Consider the alphabet $\Sigma = V \cup O \cup K$ with

$V = \{p_1, p_2, ...\}$ a countable set of **propositional variables**,

$O = \{\neg/_1, \wedge/_2, \vee/_2, \rightarrow/_2, \leftrightarrow/_2\}$ **operators** with arities (connectives),

$K = \{(,)\}$ **brackets** (auxiliary symbols).

The set of **statement forms** (formulae of propositional logic) $F \subseteq \Sigma^*$ is inductively defined by:

1. $V \subseteq F$ set of **atomic propositions**
2. If $A, B \in F$ then $(\neg A), (A \wedge B), (A \vee B), (A \rightarrow B), (A \leftrightarrow B) \in F$.

Inductive definitions implicitly use the hull operator: F is the smallest set that contains V and satisfies 2. This addendum is often omitted.

Structural Induction

Properties of elements in F are proven by **structural induction**, i.e. induction over the structure of formulae.

Let for example $f : F \times \mathbb{N} \to \mathbb{N}$ be defined by

$$f(A, i) := \text{Number of opening (minus number of closing brackets) in the first } i \text{ letters of } A.$$

The following statement can be proven via structural induction:

Lemma 1.2

For every formula $A \in F$ and for all $1 \leq i < |A|$ it is $f(A, i) > 0$. Moreover, $f(A, |A|) = 0$.

Corollary 1.3

Let $A \in F$ and $B \in \Sigma^*$ a true prefix of A. Then $B \notin F$.

Theorem 1.4 (Uniqueness Theorem)

Every formula $A \in F$ is either atomic or can be uniquely represented as $A \equiv (\neg A_1)$ or $A \equiv (A_1 * A_2)$ with $* \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$ and $A_1, A_2 \in F$.

Here $\equiv \subseteq \Sigma^* \times \Sigma^*$ is the syntactical equality of words, so the formulae are identical letter for letter.

Conventions: Abbreviations and Priorities

- Examples for formulae in proposition logic are

$$p_1, p_{101}, (((p_1 \to p_2) \wedge (\neg p_2)) \to (\neg p_1)), (p_1 \vee (\neg p_1))$$

- Omit outer brackets
- For improved readability: Priorities: $\neg, \wedge, \vee, \to, \leftrightarrow$

$$\begin{array}{rll} A \wedge B \to C & \text{stands for} & ((A \wedge B) \to C) \\ A \vee B \wedge C & \text{stands for} & (A \vee (B \wedge C)) \\ \neg A \vee B \wedge C & \text{stands for} & ((\neg A) \vee (B \wedge C)) \\ A \vee B \vee C & \text{stands for} & ((A \vee B) \vee C) \quad \text{(left-first bracketing).} \end{array}$$

Semantics

Definition 1.5 (Valuation)

A **valuation** of propositional formulae is a function $\varphi : F \to \mathbb{B} := \{0, 1\}$, so that:

$$\varphi(\neg A) = 1 - \varphi(A)$$
$$\varphi(A \vee B) = max(\varphi(A), \varphi(B))$$
$$\varphi(A \wedge B) = min(\varphi(A), \varphi(B))$$
$$\varphi(A \to B) = \begin{cases} 0 & \text{if } \varphi(A) = 1 \text{ and } \varphi(B) = 0 \\ 1 & \text{else} \end{cases}$$
$$\varphi(A \leftrightarrow B) = \begin{cases} 0 & \text{if } \varphi(A) \neq \varphi(B) \\ 1 & \text{if } \varphi(A) = \varphi(B) \end{cases}$$

Assignments and Valuations (Cont.)

- We say: A is **false** under φ, if $\varphi(A) = 0$
  A is **true** under φ or φ **satisfies** A, if $\varphi(A) = 1$.
- Notation of valuations using **truth tables**:

A	$\neg A$
1	0
0	1

A	B	$A \vee B$	$A \wedge B$	$A \rightarrow B$	$A \leftrightarrow B$
0	0	0	0	1	1
0	1	1	0	1	0
1	0	1	0	0	0
1	1	1	1	1	1

Assignments and Valuations (Cont.)

- An **assignment** of the variables V is a function $\psi : V \to \mathbb{B}$.
- Every valuation induces a unique assignment: $\psi(p_i) := \varphi(p_i)$.

Lemma 1.6

Every assignment $\psi : V \to \mathbb{B}$ can be extended to a valuation $\varphi : F \to \mathbb{B}$ in exactly one way. In particular, every valuation is uniquely determined by the values of V.

Assignments and Valuations (Cont.)

Consequence 1.7

The valuation of a statement form $A \in F$ depends only on the values of the propositional variables from V that occur in it. I.e. if one wants to compute $\varphi(A)$, it suffices to know the values $\varphi(p)$ for all propositional variables p occurring in A.

- Example: Let $\varphi(p) = 1, \varphi(q) = 1, \varphi(r) = 0$. $\varphi(A)$ can be computed iteratively:

$$A \equiv \underbrace{\underbrace{(\underbrace{p}_{1} \rightarrow \underbrace{(\underbrace{q}_{1} \rightarrow \underbrace{r}_{0})}_{0})}_{0} \rightarrow \underbrace{(\underbrace{(\underbrace{p}_{1} \wedge \underbrace{q}_{1})}_{1} \rightarrow \underbrace{r}_{0})}_{0}}_{1}$$

So $\varphi(A) = 1$.

Assignments and Valuations (Cont.)

Which values does $\varphi(A)$ attain, when φ runs through **all** assignments?

A defines a boolean function $f_A : \mathbb{B}^n \to \mathbb{B}$.

- Is $\varphi(A) = 1$ for all assignments φ?
- It suffices to check the **finitely** many assignments of the variables occurring in A.
- If n variables occur in A, there are 2^n different assignments.
- Example: For the three variables p, q and r from the example above, there are 8 assignments that must be considered.

Assignments and Valuations (Cont.)

p	q	r	$q \to r$	$p \wedge q$	$p \to (q \to r)$	$(p \wedge q) \to r$	A
0	0	0	1	0	1	1	1
0	0	1	1	0	1	1	1
0	1	0	0	0	1	1	1
0	1	1	1	0	1	1	1
1	0	0	1	0	1	1	1
1	0	1	1	0	1	1	1
1	1	0	0	1	0	0	1
1	1	1	1	1	1	1	1

A is true independently of the values of p, q, r, i.e. for every valuation φ.
Other such formulae are:
$(A \to (B \to A))$, $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$ or
$((\neg A \to \neg B) \to (B \to A))$.

Important Terms

Definition 1.8

Let $A \in F$, $\Sigma \subseteq F$.

1.(a) A is called **tautology** (**valid**), if $\varphi(A) = 1$ for every valuation φ. (Notation $\models A$)

(b) A is **satisfiable**, if there is a valuation φ so that $\varphi(A) = 1$.

(c) A is **contradictory**, if $\varphi(A) = 0$ for every valuation φ.

(d) $\mathrm{TAUT} := \{A \in F \mid A \text{ is tautology}\}$ the **set of tautologies**.

(e) $\mathrm{SAT} := \{A \in F \mid A \text{ is satisfiable}\}$ the **set of satisfiable formulae**. Note that $\mathrm{TAUT} \subseteq \mathrm{SAT}$.

Definition (Cont.)

2.(a) Σ is **satisfiable**, if there is a valuation φ with $\varphi(A) = 1$ for all $A \in \Sigma$. (φ **satisfies** Σ)

(b) **Semantic notion of inference:** A is **logical consequence** of Σ, if $\varphi(A) = 1$ for every valuation φ that satisfies Σ.

We write $\Sigma \models A$. Also $A_1, \ldots, A_n \models A$, if $\Sigma = \{A_1, \ldots, A_n\}$.

(c) The set $\mathrm{Cons}(\Sigma)$ of consequences of Σ is defined by:

$$\mathrm{Cons}(\Sigma) := \{A \in F \mid \Sigma \models A\}.$$

Examples

Example 1.9

1. $(p \vee (\neg p))$, $((p \to q) \vee (q \to r))$, $p \to (q \to p)$, $(p \to p)$, $(p \to \neg\neg p)$ and A from consequence 1.7 are tautologies.
2. $(p \wedge (\neg p))$ is contradictory.
3. $(p \wedge q)$ is satisfiable, but neither a tautology nor a contradiction.
4. Let $\Sigma = \{p\}$ and $A = p \vee q$. Then $\Sigma \models A$, because if $\varphi(p) = 1$, then also $\varphi(p \vee q) = 1$. Every valuation that satisfies Σ, also satisfies A.

Consequences

Lemma 1.10

(a) *A valid iff $\neg A$ contradictory.*

(b) $\emptyset \models A$ *iff* A *is a tautology:* $Cons(\emptyset) = TAUT$.

(c) *If* Σ *is not satisfiable, then* $\Sigma \models A$ *for all* $A \in F$*:* $Cons(\Sigma) = F$*. In particular,* $\Sigma \models A$ *and* $\Sigma \models \neg A$ *for* $A \in F$*.*

(d) *Let* $\Sigma \subseteq \Sigma'$*. If* Σ' *is satisfiable, then* Σ *is also satisfiable.*

(e) $\Sigma \subseteq Cons(\Sigma)$ *and* $Cons(Cons(\Sigma)) = Cons(\Sigma)$.

(f) *If* $\Sigma \subseteq \Sigma'$*, then* $Cons(\Sigma) \subseteq Cons(\Sigma')$.

(g) $\Sigma \models A$ *iff* $\Sigma \cup \{\neg A\}$ *unsatisfiable.*

(h) *If* Σ *is finite, then it is decidable whether* Σ *is satisfiable, and the set* $Cons(\Sigma)$ *ist decidable.*

(i) *The sets* $TAUT, SAT$ *are decidable.*

Deduction Theorem and Modus Ponens

Lemma 1.11

a) **Deduction theorem** *(semantic version):*

$$\Sigma, A \models B \qquad \text{iff} \qquad \Sigma \models (A \to B).$$

(Σ, A is shorthand for $\Sigma \cup \{A\}$)

b) **Modus ponens**:

$$\{A, A \to B\} \models B.$$

In particular, B is a tautology if A and $(A \to B)$ are tautologies.

Compactness Theorem of Propositional Logic

Theorem 1.12 (Compactness Theorem)

*$\Sigma \subseteq F$ is satisfiable iff every **finite** subset of Σ is satisfiable.*
*$\Sigma \subseteq F$ is unsatisfiable iff there is an unsatisfiable **finite** subset of Σ.*

Corollary 1.13

*$\Sigma \models A$ iff there is a **finite** subset $\Sigma_0 \subseteq \Sigma$ with $\Sigma_0 \models A$.*

- The second part of the theorem is the basis for proof methods for $\Sigma \models A$. This is the case, if $\Sigma \cup \{\neg A\}$ is unsatisfiable.
- Proofs by contradiction try to systematically find a **finite** set $\Sigma_0 \subseteq \Sigma$ so that $\Sigma_0 \cup \{\neg A\}$ is unsatisfiable.

Applications for the Compactness Theorem

Example 1.14

Let $\Sigma \subseteq F$. If for every valuation φ there is a $A \in \Sigma$ with $\varphi(A) = 1$, then there are $A_1, ..., A_n \in \Sigma$ $(n > 0)$ with $\models A_1 \vee ... \vee A_n$.

- Consider the set $\Sigma' = \{\neg A \in F \mid A \in \Sigma\}$, which is unsatisfiable.
  Hence there is a finite nonempty subset $\{\neg A_1, ..., \neg A_n\}$ of Σ' that is unsatisfiable.
  Thus, for every valuation φ there is an i with $\varphi(\neg A_i) = 0$.
  So we have $\varphi(A_i) = 1$ and thus $\varphi(A_1 \vee ... \vee A_n) = 1$.

Logical Equivalence

Definition 1.15 (Logical Equivalence)

Formulae $A, B \in F$ are called **logically equivalent**, $A \models\!\models\! B$, if for every valuation φ we have $\varphi(A) = \varphi(B)$.

Examples of logically equivalent formulae:

(Involution)	$A \models\!\models\! \neg(\neg A)$	
(Idempotence)	$A \models\!\models\! A \wedge A$	$A \models\!\models\! A \vee A$
(Commutativity)	$A \wedge B \models\!\models\! B \wedge A$	$A \vee B \models\!\models\! B \vee A$
(Associativity)	$A \wedge (B \wedge C) \models\!\models\! (A \wedge B) \wedge C$	
	$A \vee (B \vee C) \models\!\models\! (A \vee B) \vee C$	
(Distributivity)	$A \wedge (B \vee C) \models\!\models\! (A \wedge B) \vee (A \wedge C)$	
	$A \vee (B \wedge C) \models\!\models\! (A \vee B) \wedge (A \vee C)$	

Logical Equivalence (Cont.)

(De Morgan) $\neg(A \wedge B) \models\!\!\models \neg A \vee \neg B$ $\quad$ $\neg(A \vee B) \models\!\!\models \neg A \wedge \neg B$

$A \to B \models\!\!\models \neg A \vee B$ $\quad$ $A \leftrightarrow B \models\!\!\models (A \to B) \wedge (B \to A)$

$A \wedge B \models\!\!\models \neg(A \to \neg B)$ $\quad$ $A \vee B \models\!\!\models \neg A \to B$

Lemma 1.16

Logical equivalence $\models\!\!\models \subseteq F \times F$ is an equivalence relation, that means it is reflexive, symmetric and transitive.

It is even a congruence: if one replaces a subformula B in formula A by $C \models\!\!\models B$, the result is $A' \models\!\!\models A$.

Logical Equivalence (Cont.)

Lemma 1.17

The following statements are equivalent:

$$\models A \leftrightarrow B \qquad\qquad A \models\!\models B$$

$$A \models B \text{ and } B \models A \qquad\qquad Cons(A) = Cons(B)$$

Lemma 1.18

For every formula $A \in F$ there are $B, C, D \in F$ with

1. $A \models\!\models B$, *B contains only $\rightarrow$ and $\neg$ as connectives*
2. $A \models\!\models C$, *C contains only $\wedge$ and $\neg$ as connectives*
3. $A \models\!\models D$, *D contains only $\vee$ and $\neg$ as connectives*

Results from the equivalences above.

Logical Equivalence (Cont.)

Definition 1.19 (Complete Sets of Operators)

A set $OP \subseteq \{\neg, \vee, \wedge, \rightarrow, \leftrightarrow\}$ is called **complete**, if for every $A \in F$ there is a logically equivalent formula $B \in F(OP)$. Here $F(OP)$ is the set of formulae with connectives from OP.

- Complete sets of operators for propositional logic are f.ex.: $\{\neg, \rightarrow\}, \{\neg, \vee\}, \{\neg, \wedge\}, \{\neg, \vee, \wedge\}, \{\text{false}, \rightarrow\}$.

  Here false is a constant with $\varphi(\text{false}) = 0$ for every valuation φ. Obviously $\neg A \models\!\!\models (A \rightarrow \text{false})$.
- **Normal forms**: DNF (Disjunctive normal form), CNF (Conjunctive normal form), CDNF, CCNF (Canonical forms).

Boolean Functions

Every formula $A(p_1, \ldots, p_n)$ can be considered a boolean function $f_A : \mathbb{B}^n \to \mathbb{B}$, defined as $f_A(b_1, ..., b_n) := \varphi_{\overline{b}}(A)$ where $\varphi_{\overline{b}}(p_i) := b_i$.

- It can be shown that every boolean function $f : \mathbb{B}^n \to \mathbb{B}$ $(n > 0)$ in the above form can be represented by a formula $A(p_1, \ldots, p_n)$, provided that the set of operators is complete.
- Boolean algebra has as its usual set of operators **true, false, not, or, and**.
- For other sets of operators, containing f.ex. **nand, nor**, see the digital logic. It favors nand and nor gates, as they require only two transistors.

Boolean Functions: Example

A patient monitoring system gets certain data about a patient's condition: temperature, blood pressure, pulse rate. The threshold values for the data are as follows:

Conditions In/Outputs	Meaning
A	Temperature outside 36-39 degrees C.
B	Blood pressure outside 80-160 mm.
C	Pulse rate outside 60-120 beats per minute.
O	Alarm activation is necessary

Boolean Functions: Example (Cont.)

The requirements, i.e. which combinations of values necessitate alarm activation, are determined by the medical expert. They are given in the following table:

I/O table

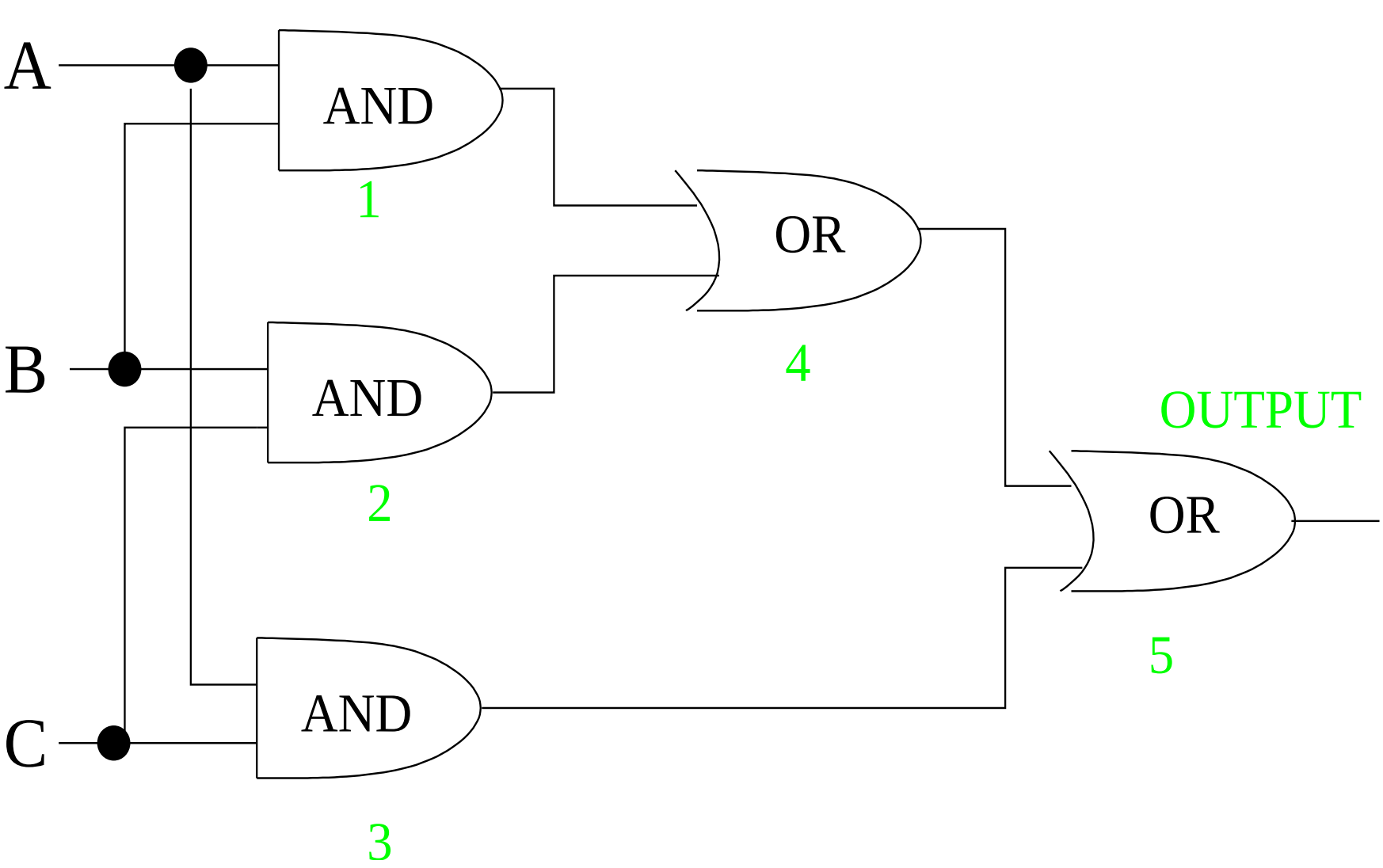
A	B	C	O
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Logical Design: Consider the columns in which O has the value 1 and construct the CDNF:

$$(\neg A \wedge B \wedge C) \vee (A \wedge \neg B \wedge C) \vee (A \wedge B \wedge \neg C) \vee (A \wedge B \wedge C)$$

Boolean Functions: Example (Cont.)

As a realization one could take the following combinatorial circuit:



Deductive Perspective on Propositional Logic

This section deals with an axiomatic construction of propositional logic by means of a **deductive system** or **proof calculus**.

A syntactically correct formula in a deductive system is called theorem, if it can be derived by purely mechanical applications of the system's rules to its axioms.

There are deductive systems in which propositional formulae are theorems if and only if they are also tautologies.

Deductive Systems

Definition 2.1 (Deductive System)

A **deductive system** $\mathcal{F}(\mathrm{Ax}, R)$ consists of

- an alphabet Δ,
- a set of formulae $F \subseteq \Delta^\star$,
- a set of axioms $\mathrm{Ax} \subseteq F$ and
- a set R of rules of the form $\frac{A_1, \ldots, A_n}{A}$ with $n > 0$ and $A_1, \ldots, A_n, A \in F$.

The sets F, Ax and R are typically **decidable**.

Deductive Systems (Cont.)

Definition 2.2

The set $T = T(\mathcal{F})$ of **theorems** is inductively defined by:

1. $Ax \subseteq T$ all axioms are theorems
2. If $A_1, \ldots, A_n \in T$ and $\frac{A_1, \ldots, A_n}{A}$ in R, then $A \in T$.

Write $A \in T(\mathcal{F})$ as $\vdash_{\mathcal{F}} A$ or $\vdash A$ and say A is derivable in $\mathcal{F}$.

Deductive notion of inference: Let $\Sigma \subseteq F$, $A \in F$. Then A in $\mathcal{F}$ is derivable from Σ, short $\Sigma \vdash_{\mathcal{F}(Ax,R)} A$, if $\vdash_{\mathcal{F}(Ax \cup \Sigma,R)} A$. Also:

$$Cons_{\mathcal{F}}(\Sigma) := \{A \in F \mid \Sigma \vdash_{\mathcal{F}(Ax,R)} A\}.$$

Σ is called **consistent**, if there is no formula $A \in F$ so that $\Sigma \vdash A$ and $\Sigma \vdash \neg A$.
If such a formula exists, then Σ is called **inconsistent**.

Proofs

Note 2.3

Formula A is derivable in $\mathcal{F}$ if there is a finite sequence of formulae $B_1, \ldots, B_n$ with $A \equiv B_n$ and for $1 \leq i \leq n$ we have:

$$B_i \in Ax \text{ or there are } i_1, \ldots, i_l < i \text{ and } \frac{B_{i_1} \ldots B_{i_l}}{B_i} \in R.$$

The sequence $B_1, \ldots, B_n$ is also called **proof** *for A in $\mathcal{F}$.*

A finite sequence $B_1, \ldots, B_n$ is called **abbreviated proof** *for $\Sigma \vdash B_n$, if for $1 \leq j \leq n$ we have:*

$$\Sigma \vdash B_j \text{ or there are } j_1, \ldots, j_r < j \text{ with } B_{j_1}, \ldots, B_{j_r} \vdash B_j.$$

Lemma 2.4

1. $\vdash A$ *iff there is a proof for* A.
2. *There is a proof for* $\Sigma \vdash A$ *iff there is an abbreviated proof for* $\Sigma \vdash A$.

Note

Note 2.5

- *Properties of elements of T are proven by structural induction.*
- *The set T of theorems is recursively enumerable, since Ax and R are decidable and therefore enumerable.*
- *The set of proofs*

$$\text{Proof} := \{B_1, \ldots, B_n \in F^+ \mid B_1, \ldots, B_n \text{ is a proof}\}$$

  is decidable.
- *If Σ is decidable, the statements are correspondingly true. In particular, $Cons_{\mathcal{F}}(\Sigma)$ is enumerable.*

Note (Cont.)

Lemma 2.6

- *If* $\Sigma \vdash A$, *the definition of derivation implies that there is a finite subset* $\Sigma_0 \subseteq \Sigma$ *with* $\Sigma_0 \vdash A$.
  (This corresponds to the compactness theorem for $\models$ *.)*
- *If* Σ *is inconsistent, there is a finite subset* $\Sigma_0 \subseteq \Sigma$ *which is inconsistent.*
- *If* $\Sigma \subseteq \Gamma$, *then* $Cons_{\mathcal{F}}(\Sigma) \subseteq Cons_{\mathcal{F}}(\Gamma)$.
- $\Sigma \vdash A$ *and* $\Gamma \vdash B$ *for all* $B \in \Sigma$ *implies that* $\Gamma \vdash A$.
  If $\Sigma \subseteq Cons_{\mathcal{F}}(\Gamma)$, *then* $Cons_{\mathcal{F}}(\Sigma) \subseteq Cons_{\mathcal{F}}(\Gamma)$.
  (Proofs can be composed.)
- *If* $\Sigma \vdash A$, *then* $\{\Sigma, \neg A\}$ *is inconsistent.*
  (Is the inverse true as well?)
- $T(\mathcal{F}) \subseteq Cons_{\mathcal{F}}(\Sigma)$ *for every set* Σ.

Schemata

Is there a deductive system $\mathcal{F}_0$ so that

$$\vdash_{\mathcal{F}_0} A \quad \text{iff} \quad \models A?$$

For this purpose, Ax and R are often finitely described by means of schemata.

For example, the schema $A \to (B \to A)$ describes the set

$$\{A_0 \to (B_0 \to A_0) \mid A_0, B_0 \in F\}$$

The schema $\dfrac{A, A \to B}{B}$ describes the set of rules

$$\{\frac{A_0, A_0 \to B_0}{B_0} \mid A_0, B_0 \in F\}.$$

The Deductive System $\mathcal{F}_0$

Introduced by Stephen Cole Kleene (1909 — 1994).

Definition 2.7 (The deductive system $\mathcal{F}_0$)

The deductive system $\mathcal{F}_0$ for propositional logic consists of the set F_0 of formulae in $V, \neg, \rightarrow, ($ and $)$. The set of axioms Ax is described by the following axiom schemata:

Ax1: $A \rightarrow (B \rightarrow A)$

Ax2: $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$

Ax3: $(\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A)$

The set of rules R is described by the rule schema

$$\text{MP:} \quad \frac{A, (A \rightarrow B)}{B} \quad \text{(modus ponens).}$$

Notes on the Deductive System $\mathcal{F}_0$

- Ax1, Ax2 and Ax3 describe disjoint sets of formulae.
- Ax and R are decidable.
- All axioms are tautologies. Since those are closed under modus ponens, all theorems are tautologies: $T(\mathcal{F}_0) \subseteq Taut(F_0)$.
- The modus ponens rule is not unambiguous: $\frac{A, A \to B}{B}$ and $\frac{A', A' \to B}{B}$ derive the same formula. Makes it harder to find proofs.
- It suffices to consider only axioms for formulae in $\to$ and $\neg$. Other formulae are logically equivalent to these.

  For proofs in the entire F, additional axioms are required, such as:

$$\text{Ax1}\wedge : (A \wedge B) \to \neg(A \to \neg B) \quad \text{Ax2}\wedge : \neg(A \to \neg B) \to (A \wedge B)$$

Example

Example 2.8

For all $A \in F_0$ we have $\vdash (A \to A)$, so $(A \to A) \in T(\mathcal{F}_0)$

Proof:

$$
\begin{array}{lll}
B_0 \equiv (A \to ((A \to A) \to A)) \to & & \\
\qquad ((A \to (A \to A)) \to (A \to A)) & & \text{Ax2} \\
B_1 \equiv A \to ((A \to A) \to A) & & \text{Ax1} \\
B_2 \equiv (A \to (A \to A)) \to (A \to A) & & \text{MP}(B_0, B_1) \\
B_3 \equiv A \to (A \to A) & & \text{Ax1} \\
B_4 \equiv A \to A & & \text{MP}(B_2, B_3)
\end{array}
$$

■

Deduction Theorem

How to find proofs in the system $\mathcal{F}_0$?

Only clue: if target formula B is not an axiom, it must occur in the form $(A_1 \to \ldots (A_n \to B) \ldots)$. Choose fitting A.

Helpful:

Theorem 2.9 (Deduction Theorem (syntactic version))

Let $\Sigma \subseteq F_0$ and $A, B \in F_0$. Then

$$\Sigma, A \vdash B \quad \textit{iff} \quad \Sigma \vdash (A \to B).$$

Applications of the Deduction Theorem

Example 2.10

To show $\vdash \neg\neg A \to A$, it suffices to show $\neg\neg A \vdash A$.

Proof:

$B_1 \equiv \neg\neg A$	
$B_2 \equiv \neg\neg A \to (\neg\neg\neg\neg A \to \neg\neg A)$	Ax1
$B_3 \equiv \neg\neg\neg\neg A \to \neg\neg A$	MP
$B_4 \equiv (\neg\neg\neg\neg A \to \neg\neg A) \to (\neg A \to \neg\neg\neg A)$	Ax3 ■
$B_5 \equiv \neg A \to \neg\neg\neg A$	MP
$B_6 \equiv (\neg A \to \neg\neg\neg A) \to (\neg\neg A \to A)$	Ax3
$B_7 \equiv \neg\neg A \to A$	MP
$B_8 \equiv A$	MP

Applications of the Deduction Theorem (Cont.)

Lemma 2.11

- *The following formulae are theorems in* $\mathcal{F}_0$*:*

(Transitivity of implication)	$\vdash (A \to B) \to ((B \to C) \to (A \to C))$	(1)
(Implication from inconsistency)	$\vdash \neg B \to (B \to A)$	(2)
(Double negation)	$\vdash B \to \neg\neg B$	(3)
(Contraposition)	$\vdash (A \to B) \to (\neg B \to \neg A)$	(4)
(Implication)	$\vdash B \to (\neg C \to \neg(B \to C))$	(5)
(Auxiliary lemma 1)	$\vdash (A \to B) \to ((A \to \neg B) \to (A \to \neg Ax))$	(E1)
(Auxiliary lemma 2)	$\vdash (A \to \neg Ax) \to \neg A$	(E2)
(Negation from inconsistency)	$\vdash (A \to B) \to ((A \to \neg B) \to \neg A)$	(6)
(Elimination of assumptions)	$\vdash (B \to A) \to ((\neg B \to A) \to A)$	(7)

- $\Sigma \vdash A$ *iff* $\Sigma \cup \{\neg A\}$ *is inconsistent.*

Soundness and Completeness of $\mathcal{F}_0$

Question: Can all tautologies be derived as theorems in the system $\mathcal{F}_0$?

Theorem 2.12 (Soundness and Completeness of $\mathcal{F}_0$)

Let $A \in F_0$ a formula of propositional logic.

a) Soundness: $\vdash_{\mathcal{F}_0} A$ *implies* $\models A$, *only tautologies can be derived as theorems in* $\mathcal{F}_0$.

b) Completeness: $\models A$ *implies* $\vdash_{\mathcal{F}_0} A$, *all tautologies can be derived in* $\mathcal{F}_0$.

Soundness and Completeness of $\mathcal{F}_0$ (Cont.)

As a tool we use:

Lemma 2.13

Consider $A(p_1, \ldots, p_n) \in F_0$ *with* $n > 0$. *Let* φ *be a valuation. With*

$$P_i := \begin{cases} p_i, & \text{if } \varphi(p_i) = 1 \\ \neg p_i, & \text{if } \varphi(p_i) = 0 \end{cases} \qquad A' := \begin{cases} A, & \text{if } \varphi(A) = 1 \\ \neg A, & \text{if } \varphi(A) = 0 \end{cases}$$

we have $P_1, \ldots, P_n \vdash A'$.

Consequence

Consequence 2.14

Let $\Sigma \subseteq F_0, A \in F_0$.

- $\Sigma \vdash_{\mathcal{F}_0} A$ *iff* $\Sigma \models A$.
- Σ *is consistent iff* Σ *is satisfiable.*
- *If* Σ *is finite and* $A \in F_0$*, then* $\Sigma \vdash_{\mathcal{F}_0} A$ *is decidable.*

Proof

Proof:

$$
\begin{aligned}
& \Sigma \vdash_{\mathcal{F}_0} A \\
\overset{2.6}{\Longleftrightarrow}\ & \text{There are } A_1, \ldots, A_n \in \Sigma \text{ with } A_1, \ldots, A_n \vdash_{\mathcal{F}_0} A \\
\overset{\text{D.T.}}{\Longleftrightarrow}\ & \text{There are } A_1, \ldots, A_n \in \Sigma \text{ with} \\
& \quad \vdash_{\mathcal{F}_0} (A_1 \to (A_2 \to \ldots (A_n \to A) \ldots)) \\
\overset{2.12}{\Longleftrightarrow}\ & \text{There are } A_1, \ldots, A_n \in \Sigma \text{ with} \\
& \quad \models (A_1 \to (A_2 \to \ldots (A_n \to A) \ldots)) \\
\overset{\text{D.T.}}{\Longleftrightarrow}\ & \text{There are } A_1, \ldots, A_n \in \Sigma \text{ with } A_1, \ldots, A_n \models A \\
\overset{\text{C.T.}}{\Longleftrightarrow}\ & \Sigma \models A
\end{aligned}
$$

■

Proof (Cont.)

Proof:

$$
\begin{aligned}
& \Sigma \text{ is consistent} \\
\iff & \text{There is no } A \text{ with } \Sigma \vdash A \text{ and } \Sigma \vdash \neg A \\
\iff & \text{There is no } A \text{ with } \Sigma \models A \text{ and } \Sigma \models \neg A \\
\iff & \Sigma \text{ is satisfiable. (Lemma 1.10(c)).}
\end{aligned}
$$

■

Sequent Calculus

There are other sound and complete deductive systems.
The following system is due to Gerhard Gentzen (1909 — 1945).
It is especially suited for automating proofs.

Definition 2.15 (Gentzen Sequent Calculus)

Let $\Gamma, \Delta \subseteq F$ finite sets of formulae. A sequent is a character sequence of the form $\Gamma \vdash_G \Delta$.

Semantic interpretation of sequents: For every valuation φ there is a formula $A \in \Gamma$ with $\varphi(A) = 0$ or there are $B \in \Delta$ with $\varphi(B) = 1$.

If $\Gamma = \{A_1, \ldots, A_n\}$ and $\Delta = \{B_1, \ldots, B_m\}$, then the sequent $\Gamma \vdash_G \Delta$ corresponds to the formula

$$(A_1 \wedge \cdots \wedge A_n) \rightarrow (B_1 \vee \cdots \vee B_m).$$

Sequent Calculus (Cont.)

Definition 2.15 (Gentzen Sequent Calculus (Cont.))

The calculus for objects of the form $\Gamma \vdash_G \Delta$ is defined by the axioms:

$$\text{(Ax1) } \Gamma, A \vdash_G A, \Delta \qquad \text{(Ax2) } \Gamma, A, \neg A \vdash_G \Delta \qquad \text{(Ax3) } \Gamma \vdash_G A, \neg A, \Delta$$

The rules of the sequent calculus are as follows:

$$\mathbf{R}_{\wedge,\vee}: \quad \frac{\Gamma, A, B \vdash_G \Delta}{\Gamma, A \wedge B \vdash_G \Delta} \qquad \frac{\Gamma \vdash_G A, B, \Delta}{\Gamma \vdash_G A \vee B, \Delta}$$

$$\mathbf{R}_{\rightarrow}: \quad \frac{\Gamma, A \vdash_G \Delta, B}{\Gamma \vdash_G A \rightarrow B, \Delta} \qquad \frac{\Gamma \vdash_G A, \Delta \,; \quad \Gamma, B \vdash_G \Delta}{\Gamma, A \rightarrow B \vdash_G \Delta}$$

$$\mathbf{R}_{\neg}: \quad \frac{\Gamma, A \vdash_G \Delta}{\Gamma \vdash_G \neg A, \Delta} \qquad \frac{\Gamma \vdash_G A, \Delta}{\Gamma, \neg A \vdash_G \Delta}$$

$$\mathbf{R}_{\wedge'}: \quad \frac{\Gamma \vdash_G A, \Delta \,; \quad \Gamma \vdash_G B, \Delta}{\Gamma \vdash_G A \wedge B, \Delta}$$

$$\mathbf{R}_{\vee'}: \quad \frac{\Gamma, A \vdash_G \Delta \,; \quad \Gamma, B \vdash_G \Delta}{\Gamma, A \vee B \vdash_G \Delta}$$

Sequent Calculus (Fort.)

A sequent $\Gamma \vdash_G \Delta$ is called derivable, if there is a finite sequence of sequents $\Gamma_1 \vdash_G \Delta_1, \ldots, \Gamma_r \vdash_G \Delta_r$ with $\Gamma_r \equiv \Gamma$, $\Delta_r \equiv \Delta$ and

> Every $\Gamma_j \vdash_G \Delta_j$ with $1 \leq j \leq r$ is an axiom or follows from previous sequence elements due to a rule.

Theorem 2.16

The sequent calculus is

sound: $\Gamma \vdash_G \Delta$ *implies* $\Gamma \models \Delta$

complete: $\Gamma \models \Delta$ *implies* $\Gamma \vdash_G \Delta$.

Here $\Gamma \models \Delta$ *with* $\Delta \subseteq F$ *is finitely defined as* $\Gamma \models \bigvee_{B \in \Delta} B$.

Example

Proofs in sequent calculus are constructed bottom-up and tree-like:

Example 2.17

$p \vee q, \neg p \vee r \vdash_G q \vee r$

Proof:

$$
\begin{array}{ll}
q, \neg p \vee r \vdash q, r & \text{Ax1} \\
p, r \vdash q, r & \text{Ax1} \\
p, \neg p \vdash q, r & \text{Ax2} \\
p, \neg p \vee r \vdash q, r & R_{\vee'} \\
p \vee q, \neg p \vee r \vdash q, r & R_{\vee'} \\
p \vee q, \neg p \vee r \vdash q \vee r & R_{\vee}
\end{array}
$$

■

We consider methods to decide whether $\Sigma \models A$ is true for a given finite set $\Sigma \subseteq F$ and $A \in F$.

The previously considered methods check all assignments of the variables occurring in the formulae, or enumerate the theorems of a suitable deductive system. This is very expensive.

Use satisfiability checker:

$$\Sigma \models A \quad \text{iff} \quad \Sigma \cup \{\neg A\} \text{ unsatisfiable.}$$

The complexity of satisfiability stays large: SAT is NP-complete.

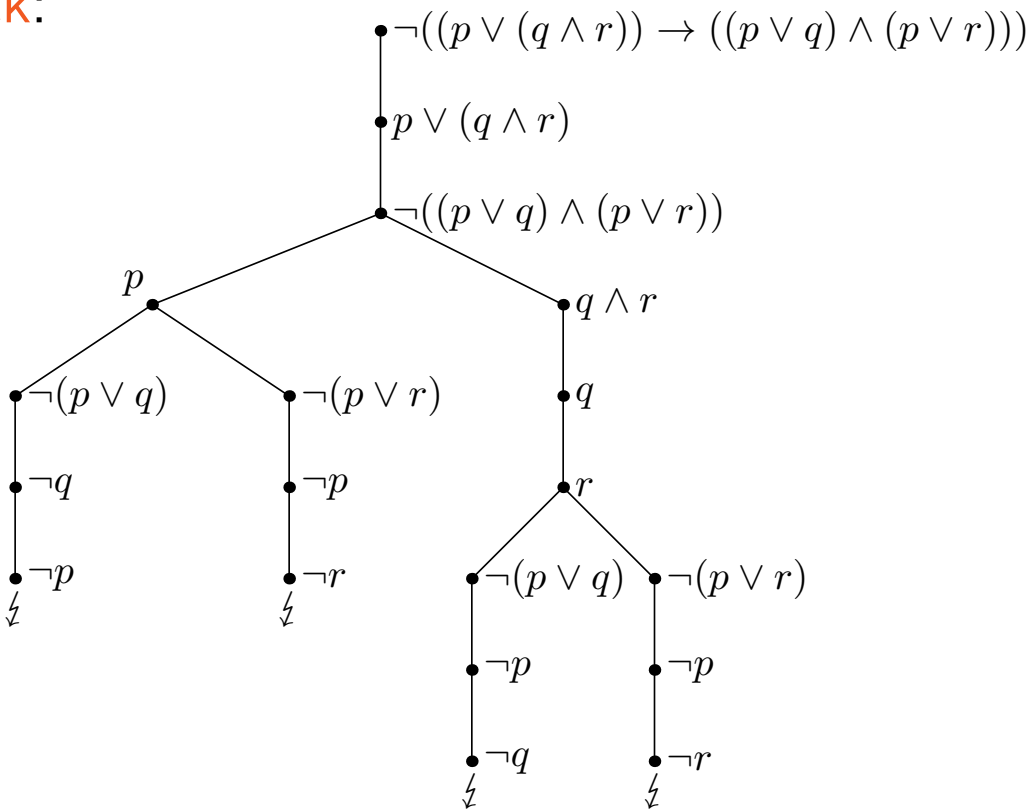
Look for methods, which are faster than the brute force method when provided with usual input.

Semantic Tableaux Davis-Putnam Resolution.

Semantic Tableaux: Example

Show that $\neg((p \vee (q \wedge r)) \rightarrow ((p \vee q) \wedge (p \vee r)))$ is unsatisfiable.

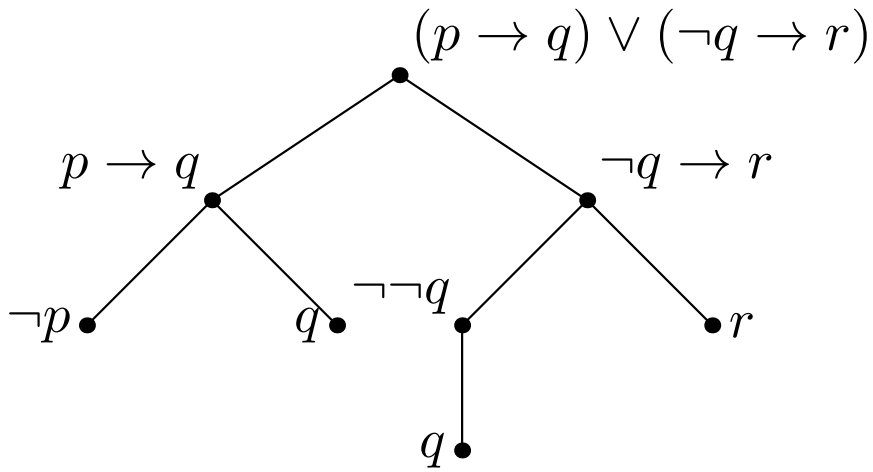
Satisfiability check:



Since all branches lead to contradictions, the formula is unsatisfiable.

Semantic Tableaux: Example (Cont.)

Find all valuations that fulfill $A \equiv (p \to q) \vee (\neg q \to r)$:



Thus, $\{\varphi : F \to \mathbb{B} \mid \varphi(p) = 0 \text{ oder } \varphi(q) = 1 \text{ oder } \varphi(r) = 1\}$ is the set of all valuations that satisfy A

At the leaves, a logically equivalent DNF can be read: $\neg p \vee q \vee r$

Intuition for Tableaux

The satisfying validations of the root formula are the union of the satisfying valuations of all branches.

- For every satisfying valuation of the root formula, there is a branch in the tableau so that all its formulae are satisfied by that valuation.
- Conversely, every satisfiable branch defines satisfying validations of the root formula.

Trick: If the formulae are maximally unfolded (the tableau is complete), satisfying valuations or contradictions are immediately visible.

Definition of Tableaux

Two kinds of formulae: β-formulae cause branching, α-formulae do not cause branching:

- α-formulae with components α_1 and α_2 lead to child nodes with markings α_1 and α_2:

α	$\neg\neg A$	$A_1 \wedge A_2$	$\neg(A_1 \vee A_2)$	$\neg(A_1 \rightarrow A_2)$
α_1	A	A_1	$\neg A_1$	A_1
α_2	(A)	A_2	$\neg A_2$	$\neg A_2$

- β-formulae with components β_1 and β_2 lead to branches with node markings β_1 and β_2:

β		$\neg(A_1 \wedge A_2)$		$A_1 \vee A_2$		$A_1 \rightarrow A_2$	
β_1	β_2	$\neg A_1$	$\neg A_2$	A_1	A_2	$\neg A_1$	A_2

Note: Every formula is a literal (p or $\neg p$ with $p \in V$), an α- or a β-formula, and exactly one of these three types.

Definition of Tableaux (Cont.)

Definition 3.1 (Tableau)

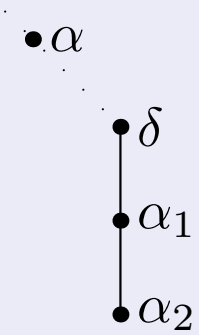
Tableaux are binary trees whose nodes are marked with formulae in F.
The set of tableaux T_A for $A \in F$ is inductively defined by:

(a) $\tau_A \in T_A$, where τ_A has one node labeled with A:

$$\bullet A$$

(b) If $\tau \in T_A$ and δ the marking of a leaf of τ, then τ can be extended to a tableau $\tau' \in T_A$ as follows:

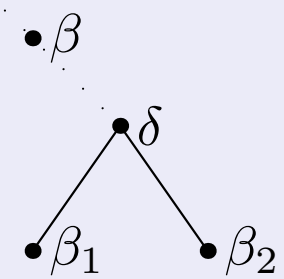
(α) Add two subsequent nodes to δ that are marked with α_1 and α_2 if the α-formula α occurs on the branch to δ:



Definition of Tableaux (Cont.)

Definition 3.1 (Tableau (Cont.))

(β) Add in τ' as successors of δ two nodes that are marked with the components β_1 and β_2 of a β-formula β if β occurs on the branch to δ:



In the following, branches in $\tau \in T_A$ are identified with their formula set $\Theta \subseteq F$.

Properties of Tableaux I: Semantics

Lemma 3.2

Let $A \in F$ a formula and $\tau \in T_A$ a tableau for A. Then

$$A \text{ is satisfiable} \quad \text{iff} \quad \exists \text{ branch } \Theta \in \tau : \Theta \text{ is satisfiable.}$$

The lemma follows from a stronger statement. For every valuation φ:

$$\varphi \text{ satisfies } A \quad \text{iff} \quad \exists \text{ branch } \Theta \in \tau : \varphi \text{ satisfies } \Theta.$$

The satisfying valuations of the branches are exactly the satisfying valuations of the root formula.

Tableaux are not unique, but Lemma 6.2 has the following consequence: Either every tableau $\tau \in T_A$ has a satisfiable branch or none of them.

Complete and Closed Sets and Tableaux

The notion of branch satisfiability is of semantical nature.
The goal of tableaux is to
ccemphautomatically check the satisfiability of formulae.
For this purpose, satisfiability must be characterized synctactically.

Definition 3.3

- A set of formulae $\Theta \subseteq F$ is called **complete** if for $\alpha \in \Theta$ we also have $\{\alpha_1, \alpha_2\} \subseteq \Theta$ and for $\beta \in \Theta$ also $\beta_1 \in \Theta$ or $\beta_2 \in \Theta$.

  A tableau τ is called **complete** if every branch $\Theta \in \tau$ is complete.
- A set of formulae Θ is called **closed** if there is a formula $B \in F$ so that $\{B, \neg B\} \subseteq \Theta$. Otherwise the set is called **open**.

  A tableau τ is called **closed** if every branch $\Theta \in \tau$ is closed.

Every tableau can be extended to a complete tableau.

Properties of Tableaux II: Syntax

Lemma 3.4 (Hintikka)

Let $\Theta \subseteq F$ complete. Then Θ is satisfiable iff Θ is open.

Closed sets are per definition unsatisfiable.

For the converse direction let Θ be a complete and open set. Define

$$\varphi(p) := \begin{cases} 0 & \neg p \in \Theta \\ 1 & \text{otherwise.} \end{cases}$$

Valuation φ is well-defined.

Show via induction over the size of Θ that $\varphi(A) = 1$ for all $A \in \Theta$.

Soundness and Completeness of Tableaux

Theorem 3.5

A formula $A \in F$ is unsatisfiable iff there is a closed tableau $\tau \in T_A$.

There is a closed tableau for A iff all complete tableaux for A are closed.

The tableau method is due to Evert Willem Beth (1908 — 1964).

Complete and open formula sets are Hintikka sets, due to Jaakko Hintikka (*1929). Hintikka's lemma shows that they are satisfiable.

Soundness and Completeness of Tableaux (Cont.)

Proof (of Theorem 3.5)

Let A be unsatisfiable.
Every tableau can be extended to a complete tableau. Hence, for A there is a complete tableau $\tau \in T_A$.
By lemma 6.2, all branches $\Theta \in \tau$ are unsatisfiable.
By lemma 6.4, all branches $\Theta \in \tau$ are closed.
Hence, there is a closed tableau $\tau \in T_A$.

For the converse direction let $\tau \in T_A$ be closed.
Closed branches are unsatisfiable.
With lemma 6.2, formula A is unsatisfiable. ■

Tableaux for Sets of Formulae

Let $\Sigma \subseteq F$ a possibly infinite set of formulae.
The set T_Σ of the **tableaux for Σ** is defined as before, with the difference that

- the construction begins with a formula $A \in \Sigma$ and
- in every step, $\sigma \in \Sigma$ may be appended to a leaf δ.

The tableau $\tau \in T_\Sigma$ is called complete if in addition to the previous requirements every branch $\Theta \in \tau$ contains the set Σ, so $\Sigma \subseteq \Theta$.

Tableaux for Sets of Formulae (Cont.)

Lemma 3.6

Let $\Sigma \subseteq F$ and $\tau \in T_\Sigma$ with $\Sigma \subseteq \Theta$ for every branch $\Theta \in \tau$. Then:

$$\Sigma \text{ is satisfiable} \quad \text{iff} \quad \exists \text{ branch } \Theta \in \tau : \Theta \text{ is satisfiable.}$$

Theorem 3.7

A set of formulae $\Sigma \subseteq F$ is unsatisfiable iff T_Σ contains a closed tableau.

The old proof still works, minding the following changes:

Lemma 6.2 is to be replaced by lemma 3.6.

For the completeness, the following lemma is required.

Lemma 3.8

For every set of formulae $\Sigma \subseteq F$ there is a complete tableau $\tau \in T_\Sigma$.

Systematic Construction of Tableaux

Proof of lemma 3.8 Let Σ be infinite. Given here is a non-terminating method that constructs a sequence of tableaux

$$\tau_0 \subseteq \tau_1 \subseteq \ldots \quad \text{with} \quad \tau := \bigcup_{i \in \mathbb{N}} \tau_i \text{ complete.}$$

Since $\Sigma \subseteq F$, Σ is countable, hence $\Sigma = \{A_0, A_1, \ldots\}$.

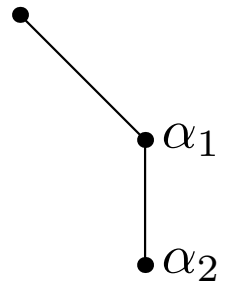
Use a FIFO worklist $WL := \emptyset$ to store nodes.
Use a counter $j := 0$, to iterate through Σ.

- $\tau_0 := \tau_{A_0}$. If A_0 is not a literal, push the node of A_0 into WL.
- τ_{n+1} is generated from τ_n as follows.

  If $WL \neq \emptyset$, pop WL. Let the node be labeled with $Y \in F$.

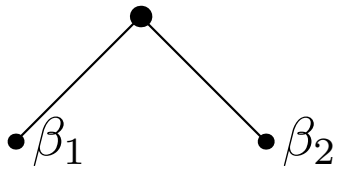
Systematic Construction of Tableaux (Cont.)

- If Y is an α-formula, extend every branch that passes the node of Y by the subformulae α_1 and α_2:



  If α_1 or α_2 aren't literals, add all new nodes labeled with α_1 bzw. α_2 to the worklist.

- If Y is a β-formula, extend every branch that passes Y by



  If the subformulae β_1, β_2 aren't literals, add the corresponding nodes to the worklist.

Systematic Construction of Tableaux (Cont.)

If $WL = \emptyset$, increment j and select $Y := A_j$.

- Add nodes labeled with Y to all branches.
  If Y isn't a literal, add the nodes to the worklist.

Systematic Construction of Tableaux (Cont.)

Claim: τ is complete.
To be more precise: Every branch $\Theta \in \tau$ is complete and contains Σ.

Proof (Sketch):
Let $\alpha \in \Theta$ be an α-formula.
Then it was added to the worklist when generated.
Due to the FIFO order it was processed at some point.
Hence, $\{\alpha_1, \alpha_2\} \subseteq \Theta$.

Consider $A_i \in \Sigma$.
At some point $j = i$.
Assume that this is not the case.
Then there is an index for which the worklist was never emptied.
This has to be false.
With the removal of a formula $A \in F$ finitely many formulae were added to the worklist, but they were all smaller.

Exercise: Why does termination follow?

Decidability and Semi-Decidability

To derive the semi-decidability of unsatisfiability from the systematic construction of tableaux, adapt the method as follows:

Do not add nodes to closed branches.

Lemma 3.9

(1) The systematic construction of tableaux terminates for finite $\Sigma \subseteq F$.

(2) Let $\Sigma \subseteq F$ be infinite and unsatisfiable. The the modified tableau construction algorithm terminates with a closed tableau.

Note: The compactness theorem follows from the second statement.
If Σ is unsatisfiable, then T_Σ contains a finite closed tableau.
Hence, a finite subset of Σ is unsatisfiable.

Decidability and Semi-Decidability (Cont.)

Lemma 3.10 (König)

Let T be an infinite tree with finite outdegree.
Then there is an infinite path in T.

Show lemma 3.9(2):

In the case of termination, the resulting tableau is closed,
since closedness is the only termination condition.

It remains to show termination.
Assume the modified method doesn't terminate.
Then it constructs an infinite tableau τ. Since the tableau has a finite outdegree, with König's lemma there is an infinite path $\Theta \in \tau$.
As in lemma 3.8 the path contains Σ, is complete and open.
With Hintikka's lemma, Θ is satisfiable.
So Σ is also satisfiable. Contradiction.

Decidability and Semi-Decidability (Cont.)

Note 3.11

- *The tableau method is a semi-decision procedure for the unsatisfiability of enumerable sets of formulae $\Sigma \subseteq F$.*
- *The tableau method is a decision procedure for the satisfiability of finite sets of formulae $\Sigma \subseteq F$.*

For the decidability, it should be noted that for an enumerable set the addition of a formula $A_i \in \Sigma$ to a tableau is effective.
Moreover, the test for closedness is decidable.

Normal Forms

Advantages:
The simpler structure of a normal form allows the use of special algorithms for the solution of certain problems.

The transformation should not be too expensive, otherwise the effort would not be worthwhile.

Examples:

- From a DNF, all satisfying assignments can be read immediately.
- From a minimal DNF one can easily derive combinatorial circuits (with AND, OR, NOT gates).
- The systematic construction of tableaux allows reading these normal forms immediately from a complete tableau.

Normal Forms (Cont.)

One can transform a formula into a

- logically equivalent formula: $A \models\!\!\models T(A)$
- equisatisfiable formula: A satisfiable iff. $T(A)$ satisfiable

We cover three of these normal forms:

- Negation normal form (**NNF**) Form in $\neg, \vee, \wedge$
- Conjunctive normal form (**CNF**) Form in $\neg, \vee, \wedge$
- Disjunctive normal form (**DNF**) Form in $\neg, \vee, \wedge$

Negation Normal Form

A formula $A \in F$ is in **Negation normal form (NNF)** if every negation is placed directly before a variable and there are no two directly subsequent negations.

Definition 3.12 (NNF)

The set of formulae in **NNF** is inductively defined by

- For $p \in V$, p and $\neg p$ are in NNF.
- If A, B are in NNF, then $(A \vee B)$ and $(A \wedge B)$ are also in NNF.

Lemma 3.13

For every formula $A \in F(\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\})$ there is a $B \in F(\neg, \vee, \wedge)$ in NNF with $A \models\!\!\models B$ and $|B| \in O(|A|)$.

Conjunctive Normal Form

Definition 3.14 (Clause)

A formula $A \equiv (L_1 \vee ... \vee L_n)$ with literals $L_1, \ldots, L_n$ is called **clause**.

- If all literals are negative, it is a **negative** clause.

  If all literals are positive, it is a **positive** clause.

  Clauses containing at most one positive literal are called **Horn clauses**.

- A is called k-clause if A contains at most $k \in \mathbb{N}$ literals.

  1-clauses are also called unit clauses.

A formula $A \equiv (A_1 \wedge ... \wedge A_m)$ is in **CNF** if A is a conjunction of clauses $A_1, \ldots, A_m$.

- If all of them are k-clauses, A is in k-**CNF**.

Conjunctive Normal Form (Cont.)

Example 3.15

$A \equiv (p \vee q) \wedge (p \vee \neg q) \wedge (\neg p \vee q) \wedge (\neg p \vee \neg q)$ is in 2-CNF.

If one considers clauses as sets of literals, then formulae in CNF can be represented as sets of sets of literals. For example A:

$$\{\{p, q\}, \{p, \neg q\}, \{\neg p, q\}, \{\neg p, \neg q\}\}.$$

Lemma 3.16

For every formula $A \in F$ there is a formula B in CNF with $A \models\!\!\models B$ and $|B| \in O(2^{|A|})$.

The bound is **strict**:
There is a sequence of formulae $(A_n)_{n \in \mathbb{N}}$ with $|A_n| = 2n$, so that every logically equivalent formula B_n in CNF is at least of length 2^n.

Disjunctive Normal Form

Definition 3.17 (DNF)

A formula $A \in F$ is in **DNF** if A is a disjunction of conjunctions of literals:

$$A \equiv (A_1 \vee \ldots \vee A_m) \quad \text{mit} \quad A_i \equiv (L_1^i \wedge \ldots \wedge L_{n_i}^i).$$

Definition 3.18 (Dual Formula)

The **dual formula** $d(A)$ of a formula $A \in F$ is defined as:

$$\begin{aligned} d(p) &\equiv p \quad \text{für } p \in V \\ d(\neg A) &\equiv \neg d(A) \\ d(B \vee C) &\equiv d(B) \wedge d(C) \\ d(B \wedge C) &\equiv d(B) \vee d(C). \end{aligned}$$

Relationships between the Normal Forms

Lemma 3.19

For every formula $A \in F$:

(1) *If A is in CNF, then NNF($\neg A$) is in DNF.*

(2) *If A is in CNF, then $d(A)$ is in DNF and vice versa.*

Lemma 3.20

For every formula $A \in F$:

(1) *If one sets $\psi(p) := 1 - \varphi(p)$, then $\psi(d(A)) = 1 - \varphi(A)$.*

(2) *A is a tautology iff $d(A)$ is a contradiction.*

(3) *A is satisfiable iff $d(A)$ is not a tautology.*

Davis-Putnam Algorithms

Idea: Reduce satisfiability for a formula with $n \in \mathbb{N}$ variables to the satisfiability problem for formulae with **at most** $n - 1$ variables.

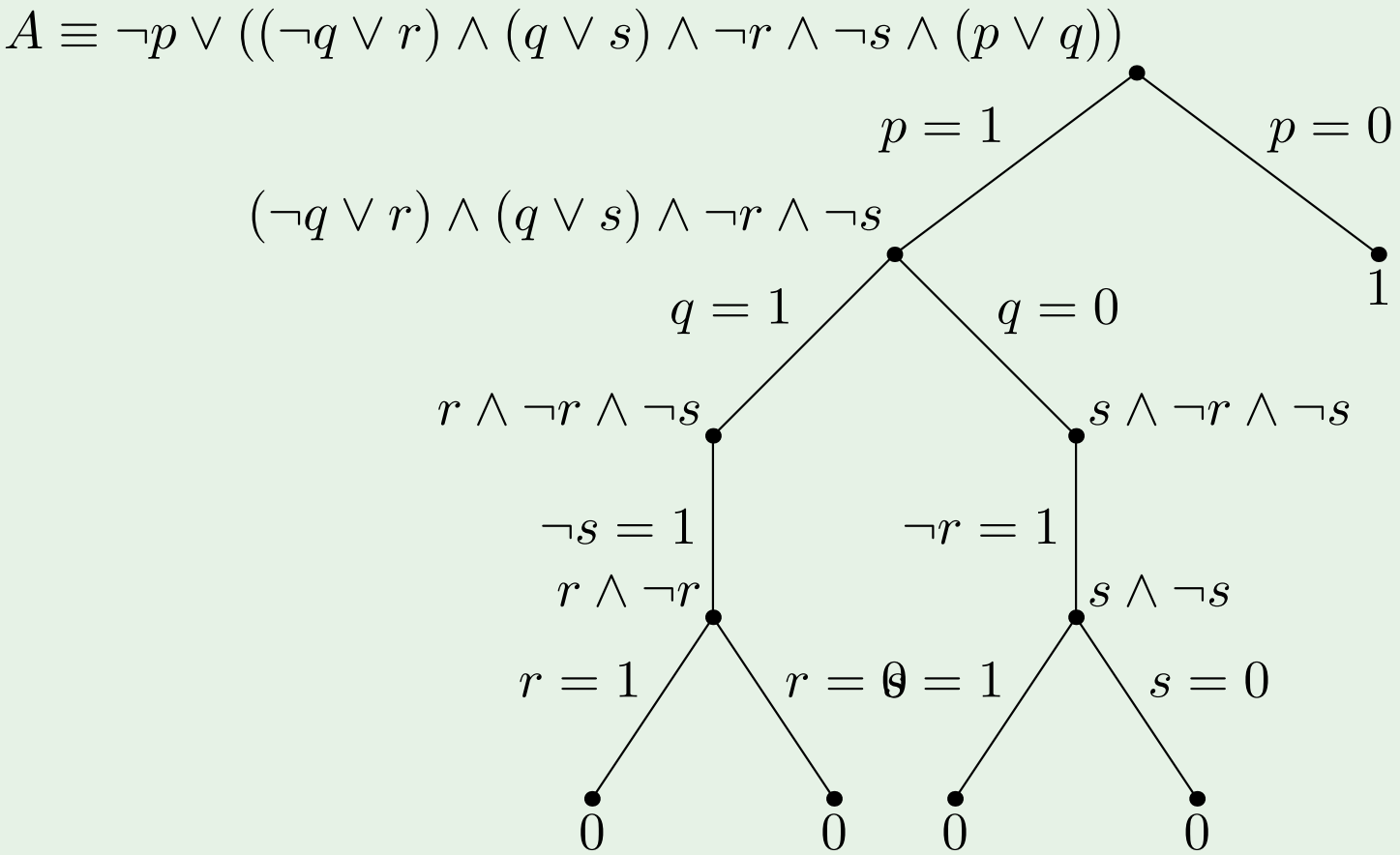
Approach: Search for a satisfying valuation by iterative choice of the values of single variables — **Bottom-Up Method**.

Algorithms using this idea, heuristics and other refinements are referred to as Davis-Putnam algorithms, due to Martin Davis (*1928) and Hilary Putnam (*1926).

Requirement: Formula in **NNF** over $\neg, \wedge, \vee$.

Davis-Putnam Algorithms (Cont.)

Example 3.21 (Visualization of the processing as tree)



Substitution

Definition 3.22 (Substitution)

Let Formula $A \in F$ in NNF and $p \in V$.
Define $A[p/1]$ as the result of the following substitution process:

(1) Replace every occurrence of p in A by 1.

(2) Execute the following rules as long as possible:

- If a subformula $\neg 1$ occurs, replace it by 0.
- Replace $\neg 0$ by 1.
- Subformulae $B \wedge 1$ and $B \vee 0$ are replaced by B.
- Subformulae $B \vee 1$ are replaced by 1.
- Subformulae $B \wedge 0$ are replaced by 0.

$A[p/0]$ is defined analogously, with p being replaced by 0.

In general, use $A[l/1]$ or $A[l/0]$ for literals l.

Substitution (Cont.)

Lemma 3.23

$A[p/1]$ and $A[p/0]$ are well-defined.

The formula $A[p/i]$ with $i \in \mathbb{B}$ is:

- *a formula in NNF or CNF if A had this form,*
- *the empty formula, which is interpreted as true, notation $A[p/i] = 1$,*
- *the empty clause, which is interpreted as false, notation $A[p/i] = 0$.*

The variable $p \in V$ does not occur in $A[p/i]$ any more.

Example 3.24

For A in CNF and literal l:
$A[l/1]$ is created by removing all clauses in A that contain literal l, and removing all occurrences of $\neg l$ in the other clauses.

Soundness of Davis-Putnam Algorithms

Lemma 3.25

A formula A in NNF is satisfiable iff $A[p/1] = 1$ or $A[p/0] = 1$ or one of the formulae $A[p/1], A[p/0]$ is satisfiable.

The lemma follows from the fact that for every valuation φ

$$\varphi(A) = \varphi(A[p/i]), \quad \text{where } i = \varphi(p).$$

By testing the formulae $A[p/1]$ and $A[p/0]$, which no longer contain $p \in V$, the satisfiability of A can be decided recursively.

Rule-based Definition of Davis-Putnam

Definition 3.26 (Rules for Formulae in NNF)

Pure-Literal rule If a variable $p \in V$ occurs only positive or only negative in a formula A, assign 1 to p or 0 to p, respectively, and reduce the formula.

A is equisatisfiable to $A[p/1]$ or $A[p/0]$, respectively.

Splitting rule If a variable $p \in V$ occurs positive as well as negative in A, create the two reduced formulae $A[p/1]$ und $A[p/0]$.

A is satisfiable iff one of the reduced formulae is 1 or satisfiable.

Rule-based Definition of Davis-Putnam (Cont.)

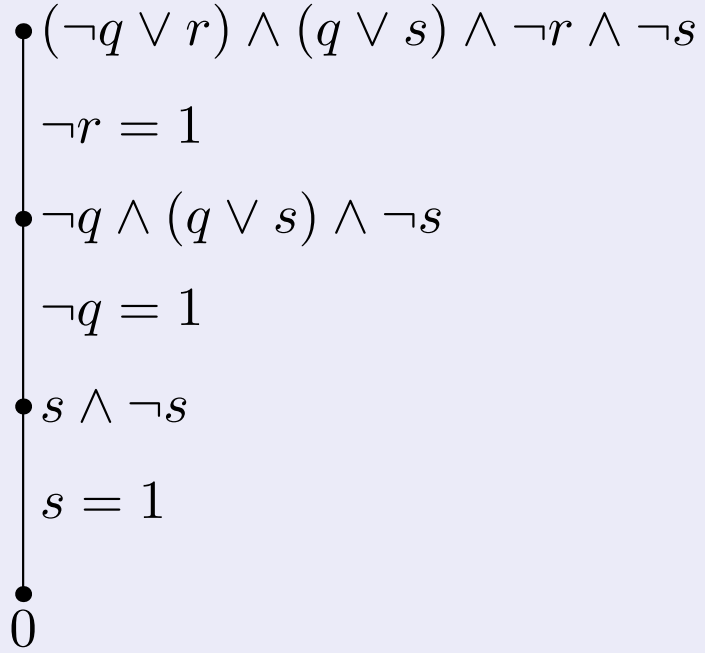
Definition 3.27 (Rules for Formulae in **CNF**)

Unit rule

Let A be in CNF and it contains a unit clause $A_i \equiv l$.

Create $A[l/1]$:

A satisfiable iff $A[l/1]$ satisfiable.



Rule-based Definition of Davis-Putnam (Cont.)

Clause A_1 **subsumes** clause A_2, notation $A_1 \subseteq A_2$ if every literal in A_1 also occurs in A_2.

From the satisfiability of a clause A_1, the satisfiability of all the clauses A_2 it subsumes follows directly: $A_1 \subseteq A_2$

Definition 3.27 (Rules for formulae in **CNF** (Cont.))

Subsumption rule Let A in CNF. Remove all clauses from A that are subsumed by others: function Subsumption_Reduce(A).

Also remove tautological clauses that contain p and $\neg p$ for a $p \in V$.

> *Since clauses are connected conjunctively, only those have to be considered that are not subsumed by others.*

procedure DPA — Davis-Putnam Algorithm

Input: A in CNF
Output: Boolean Value for satisfiability {0,1}

begin
if $A \in \{0,1\}$ **then** **return**(A);
p:=**pure**(A,s);
  //*returns variable and assignment if only positive*
  //*or negative occurrences, otherwise* **null**
if $p \neq$ **null** **then** **return**(DPA(A[p/s]));
p:=**unit**(A,s);
  //*Unit clause with assignment, otherwise* **null**
if $p \neq$ **null** **then** **return**(DPA(A[p/s]));
A:=Subsumption_Reduce(A); //*removes subsumed clauses*
p:=**split**(A); //*returns variable in A*
if DPA($A[p/1]$) $= 1$ **then** **return**(1);
return(DPA(A[p/0]));
end

Selection Criteria for the Splitting Rule

- Choose the first variable that occurs in the formula.
- Choose the variable that occurs the most often in the formula.
- Choose the variable with $\sum_{p \text{ in } A_i} |A_i|$ minimal.
- Choose the variable that occurs the most often in the shortest clauses.
- Comoute the number of positive and negative occurrences in the shortest clauses and select the variable with the largest difference.
- Other heuristics can be found in implementations.

Resolution

Idea: From clauses $(A \vee l)$ and $(B \vee \neg l)$ the new clause $(A \vee B)$ is generated, since

$$(A \vee l) \wedge (B \vee \neg l) \models\!\models (A \vee l) \wedge (B \vee \neg l) \wedge (A \vee B).$$

Here let $\neg l \equiv \neg p$ if $l \equiv p$ with $p \in V$. If $l \equiv \neg p$ then $\neg l \equiv p$.

Goal: Generating the empty clause $\sqcup$ to show **unsatisfiability**.

Resolution works on formulae **in CNF**. Here it is practical to consider clauses as sets:

$$(p \vee \neg q \vee p) \quad \text{considered as} \quad \{p, \neg q\}.$$

Resolution is due to John Alan Robinson (*1928).

Resolution (Cont.)

Definition 3.28 (Resolvent)

Let K_1, K_2 be clauses and l a literal with $l \in K_1$ and $\neg l \in K_2$. Then

$$R \equiv (K_1 \setminus \{l\}) \cup (K_2 \setminus \{\neg l\})$$

is the **resolvent of K_1 and K_2 on l**.

Note: The resolvent can be the empty clause $\sqcup$.

Adding resolvents leads to equivalent formulae.

Lemma 3.29

Let A be in CNF and R a resolvent of two clauses from A. Then

$$A \models\!\!\models A \cup \{R\}.$$

Resolution (Cont.)

Definition 3.30 (Derivations)

Let A be in CNF and K a clause. A sequence $K_1, \ldots, K_n$ of clauses with $K_n \equiv K$ is a **derivation of K from A**, $A \underset{Res}{\vdash} K$ if for $1 \leq k \leq n$:

$K_k \in A$ *or* K_k *is a resolvent of two* K_i, K_j *with* $i, j < k$.

Lemma 3.31

As a calculus, resolution is sound but **not complete***:*

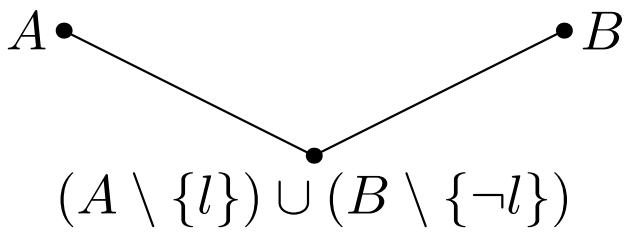
$A \underset{Res}{\vdash} K$ *implies* $A \models K$. *The converse is not true.*

Theorem 3.32 (Soundness and Refutation Completeness, Robinson)

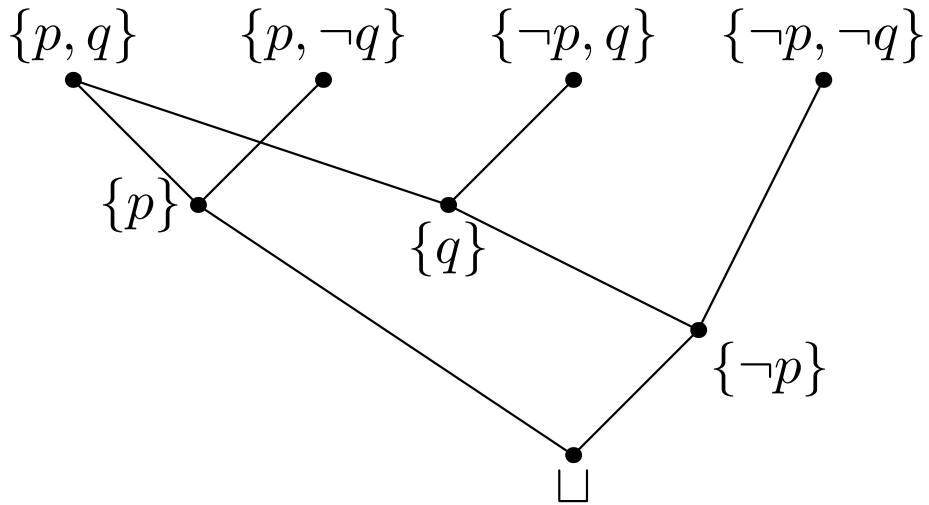
A formula A in CNF is unsatisfiable *iff* $A \underset{Res}{\vdash} \sqcup$.

Visualization

Visualization of the resolvent of two clauses A, B on l:



Visualization of derivations as directed acyclic graph (DAG):



Resolution Method: Heuristics

Strong derivations: Let A be in CNF and unsatisfiable.
Then there is a derivation $K_1, \ldots K_n \equiv \sqcup$ so that

1. no clause occurs more than once in the derivation,
2. no tautology occurs in the derivation,
3. no already subsumed clause occurs in the derivation:
   There are no K_i, K_j with $i < j$ and $K_i \subseteq K_j$.

Resolution Method: Heuristics (Cont.)

- Stepwise strategy (resolution closure)
  (all satisfying valuations)
- Set-of-support restriction
  (prefer unit clauses)
- P-(N-)resolution
- Linear resolution (SL resolution, PROLOG inference machine).

Resolution Method: Heuristics (Cont.)

Example: $A \equiv \{\{\neg p, \neg q, \neg r\}, \{p, \neg s\}, \{q, \neg r\}, \{r, \neg t\}, \{t\}\}$
Steps:

0	1	2	3
1. $\{\neg p, \neg q, \neg r\}$	6. $\{\neg q, \neg r, \neg s\}$ (1,2)	11. $\{\neg r, \neg s\}$ (6,3)	21. $\{\neg s, \neg t\}$ (11,4)
2. $\{p, \neg s\}$	7. $\{\neg p, \neg r\}$ (1,3)	12. $\{\neg q, \neg s, \neg t\}$ (6,4)	22. $\{\neg s\}$ (11,10)
3. $\{q, \neg r\}$	8. $\{\neg p, \neg q, \neg t\}$ (1,4)	13. $\{\neg p, \neg t\}$ (7,4)	⋮
4. $\{r, \neg t\}$	9. $\{q, \neg t\}$ (3,4)	14. $\{\neg p, \neg r, \neg t\}$ (8,3)	
5. $\{t\}$	10. $\{r\}$ (4,5)	15. $\{\neg p, \neg q\}$ (8,5)	
		16. $\{q\}$ (10,3)	
		17. $\{\neg r, \neg s, \neg t\}$ (6,9)	
		18. $\{\neg q, \neg s\}$ (6,10)	
		19. $\{\neg p\}$ (7,10)	
		20. $\{\neg p, \neg t\}$ (8,9)	

Get the satisfying valuation

$$\varphi(q) = 1, \varphi(p) = 0, \varphi(s) = 0, \varphi(r) = 1, \varphi(t) = 1.$$

Chapter II

First-Order Predicate Logic (with Equality)

Foundations of Predicate Logic

Goal: Formulation and inference of relations between elements of a data domain.

Applications:

- Solution of queries on datasets in AI or information systems.
- Formulation of integrity constraints on data: loop invariants of a program, constraints on XML files or data base entries.
- Solution of constraint systems in testing or planning.
- Logical programming.

Syntax of the predicate logic 1879 in the article "Begriffsschrift" (concept notation) by Gottlob Frege (1848 — 1925).

Semantics 1934 by Alfred Tarski (1901 — 1983).

Foundations of Predicate Logic (Cont.)

Semantically: Elements of a data domain, functions on these elements and relations between these elements.

Syntactically:

- Terms describe elements of the data domain.
  For describing elements: constants and variables.
  For calculating further elements: function symbols.
- Formulae make statements about the elements: true or false.
  For describing relations between elements: predicate symbols.
  Operations on the resulting truth values via logical connectives and quantifiers.

Function and predicate symbols **depend on the application**, so they are parameters of the syntax definition.

Logical symbols are **fixed**.

Foundations of Predicate Logic (Cont.)

Example 4.1 (Description of Mathematical Relations)

Syntax: Constants 1, 2, 3, function symbols $+$, $/$, variables x, y, z, predicate $<$, connectives $\rightarrow$, $\wedge$, quantifiers $\forall$, $\exists$

$$\text{Terms:} \quad 1,\ 1+\frac{2}{3},\ x+\frac{3}{2}$$

$$\text{Formulae:} \quad x<3,\ \forall x \forall y(x<y \rightarrow \exists z(x<z \wedge z<y))$$

Semantics: Data domain $\mathbb{Q}$, constants 1 to predicate $<$ have the usual meaning.

Foundations of Predicate Logic (Cont.)

Example 4.2 (Description of Relations between Data)

Syntax: Variables x, y, function $\mathit{travelDistance}(-)$, predicates $\mathit{isDog}(-)$, $\mathit{isFish}(-)$, $<$, quantifier $\forall$.

Formula:

$$\forall x \forall y \big((\mathit{isDog}(x) \wedge \mathit{isFish}(y)) \rightarrow \mathit{travelDistance}(x) < \mathit{travelDistance}(y) \big)$$

Semantics: Data domain $\{\mathit{Lassie}, \mathit{Nemo}\} \cup \mathbb{N} \cup \{\bot\}$.

The function $\mathit{travelDistance}(x)$ returns the travel distance of an animal from the data domain and $\bot$ if no animal is entered.

The predicate $\mathit{isFish}(x)$ returns true if x is a fish.

Syntax of First-Order Logic

Definition 4.3 (Signature)

A signature is a pair $S = (Func, Pred)$ with

- $Func$ a set of function symbols $f, g, \ldots \in Func$ and
- $Pred$ a set of predicate symbols $p, q, \ldots \in Pred$.

Every function and predicate symbol has an arity $k \in \mathbb{N}$.
Notation as $f/_k \in Func$ or $p/_k \in Pred$ if f resp. p has an arity of k.
Functions and predicates of arity 0 are called constants.

Assumptions:

- $Func$ and $Pred$ are decidable, not necessarily finite.
- Apart from the signature there is a countable set V of variables.
  V, $Func$, $Pred$ are pairwise disjoint and do not contain

$$\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \exists, \forall, \ , \ , = , (,).$$

Syntax of First-Order Logic (Cont.)

Definition 4.4 (Syntax of First-Order Logic)

Let $S = (Func, Pred)$ be a signature.
The set $Term(S)$ of all terms over S is inductively defined as

$$t ::= x \mid f(t_1, \ldots, t_k), \quad \text{where } x \in V \text{ and } f/_k \in Func.$$

The set $FO(S)$ of the first-order formulae over S is inductively defined as

$$\begin{aligned} A ::= {} & t_1 = t_2 \mid p(t_1, \ldots, t_k) \mid \\ & (\neg A) \mid (A_1 \wedge A_2) \mid (A_1 \vee A_2) \mid (A_1 \rightarrow A_2) \mid (A_1 \leftrightarrow A_2) \mid \\ & (\exists x A) \mid (\forall x A) \end{aligned}$$

with $t_1, t_2, \ldots, t_k \in Term(S)$, $p/_k \in Pred$ and $x \in V$.

$t_1 = t_2$ and $p(t_1, \ldots, t_k)$ are also referred to as atomic formulae.

Syntax of First-Order Logic (Cont.)

For increased **readability**:

- Omit outer braces.
- **Priorities**: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
- $\forall x_1, \ldots, x_n A$ stands for $\forall x_1(\ldots(\forall x_n A)\ldots)$
  $\exists x_1, \ldots, x_n A$ stands for $\exists x_1(\ldots(\exists x_n A)\ldots)$
- For binary predicate and function symbols, one can also use infix notation. For example, write $t_1 < t_2$ instead of $<(t_1, t_2)$.

Syntax of First-Order Logic (Cont.)

Definition 4.5 (Free and Bound Variables)

In a formula (QxA) with $Q \in \{\exists, \forall\}$, A is referred to as the scope of Qx.

An occurrence of a variable $x \in V$ in a formula is called bound if it occurs in the scope of a quantifier Qx.

Other occurrences of a variable are called free.

Formulae without free occurrences are called closed.

The set $V(A)$ contains the variables in $A \in FO(S)$. Similarly, $FV(A)$ and $GV(A)$ contain the variables that occur bound and freely, respectively, in A.

Lemma 4.6

(a) *If S is decidable, then $Term(S)$ and $FO(S)$ are also decidable.*

(b) *Compound terms and formulae can be uniquely decomposed.*

(c) *Free and bound occurrences can be effectively determined.*

Semantics of First-Order Logic

Terms and formulae are syntactic objects without meaning.

What does a term mean? What does a formula mean?

Definition 4.7 (Structure)

Let $S = (\mathit{Func}, \mathit{Pred})$ be a signature. A structure of the signature S, also referred to as S-structure, is a pair $\mathcal{M} = (D, I)$ consisting of

a non-empty set D, the domain,

and an interpretation I of the function and predicate symbols in S. Here I maps every $f/_k \in \mathit{Func}$ to a k-ary function

$$I(f) : D^k \to D \qquad \text{(Notation also } f^{\mathcal{M}} \text{ instead of } I(f))$$

and every $p/_k \in \mathit{Pred}$ to a k-ary predicate:

$$I(p) : D^k \to \mathbb{B} \qquad \text{(Notation also } p^{\mathcal{M}} \text{ instead of } I(p)).$$

Semantics of First-Order Logic (Cont.)

Assumption: Structures are chosen to fit the signatures.

Note: Equality is a logical symbol, not part of the signature.
Is not interpreted by the structure.

Definition 4.8 (Assignment)

An assignment of the variables in $\mathcal{M} = (D, I)$ is a mapping

$$\sigma : V \to D.$$

The modification $\sigma\{x/d\}$ of σ is the assignment with

$$\sigma\{x/d\}(y) := \begin{cases} d, & \text{if } y = x \\ \sigma(y), & \text{otherwise.} \end{cases}$$

The set of all assignments is denoted by D^V.

Semantics of First-Order Logic (Cont.)

Definition 4.9 (Semantics of Terms)

The semantics of a term $t \in \mathit{Term}(S)$ in $\mathcal{M} = (D, I)$ is a function

$$\mathcal{M}[\![t]\!] : D^V \to D,$$

that is defined inductively as follows:

$$\mathcal{M}[\![x]\!](\sigma) := \sigma(x)$$
$$\mathcal{M}[\![f(t_1, \ldots, t_k)]\!](\sigma) := f^{\mathcal{M}}(\mathcal{M}[\![t_1]\!](\sigma), \ldots, \mathcal{M}[\![t_k]\!](\sigma)).$$

Here $\mathcal{M}[\![t]\!](\sigma)$ is the value of t in $\mathcal{M}$ under the assignment σ.

Semantics of First-Order Logic (Cont.)

Definition 4.10 (Semantics of Formulae)

The semantics of a formula $A \in FO(S)$ in $\mathcal{M} = (D, I)$ is a function

$$\mathcal{M}[\![A]\!] : D^V \to \mathbb{B},$$

that is defined inductively as follows:

$$\mathcal{M}[\![t_1 = t_2]\!](\sigma) := 1 \quad \text{iff} \quad \mathcal{M}[\![t_1]\!](\sigma) = \mathcal{M}[\![t_2]\!](\sigma)$$
$$\mathcal{M}[\![p(t_1, \ldots, t_k)]\!](\sigma) := p^{\mathcal{M}}(\mathcal{M}[\![t_1]\!](\sigma), \ldots, \mathcal{M}[\![t_k]\!](\sigma))$$

$\neg, \wedge, \vee, \to, \leftrightarrow$ as in propositional logic: $\mathcal{M}[\![\neg A]\!](\sigma) := 1 - \mathcal{M}[\![A]\!](\sigma)$ etc.

$$\mathcal{M}[\![\exists x A]\!](\sigma) := 1 \quad \text{iff exists } d \in D \text{ with } \mathcal{M}[\![A]\!](\sigma\{x/d\}) = 1$$
$$\mathcal{M}[\![\forall x A]\!](\sigma) := 1 \quad \text{iff for all } d \in D \text{ we have } \mathcal{M}[\![A]\!](\sigma\{x/d\}) = 1.$$

Here $\mathcal{M}[\![A]\!](\sigma)$ is the truth value of A in $\mathcal{M}$ under assignment σ.

Semantics of First-Order Logic (Cont.)

Lemma 4.11 (Coincidence Theorem)

Consider $A \in FO(S)$, $\mathcal{M} = (D, I)$ and $\sigma_1, \sigma_2 \in D^V$. If $\sigma_1(x) = \sigma_2(x)$ for all $x \in FV(A)$, then

$$\mathcal{M}[\![A]\!](\sigma_1) = \mathcal{M}[\![A]\!](\sigma_2).$$

In particular, the semantics $\mathcal{M}[\![A]\!](\sigma)$ of closed formulae $A \in FO(S)$ is independent of the assignment $\sigma \in D^V$:

either A is satisfied under all assigments or under none.

Semantics of First-Order Logic (Cont.)

Definition 4.12 (Satisfiability, Tautology)

Let $A \in FO(S)$, $\mathcal{M} = (D, I)$ and $\sigma \in D^V$.

- A is satisfied in $\mathcal{M}$ under σ, notation $\mathcal{M}, \sigma \models A$ if $\mathcal{M}[\![A]\!](\sigma) = 1$.
- If A is closed, the assignment is irrelevant due to theorem 4.11. Write $\mathcal{M} \models A$ and say $\mathcal{M}$ is a model of A.
- A is a tautology or valid, notation $\models A$, if for all S-structures and all assignments $\sigma \in D^V$: $\mathcal{M}, \sigma \models A$.
- A is satisfiable if there is a S-structure $\mathcal{M}$ and an assignment $\sigma \in D^V$ with $\mathcal{M}, \sigma \models A$.

Lemma 4.13

The formula $A \in FO(S)$ is valid iff $\neg A$ is unsatisfiable.

Semantics of First-Order Logic (Cont.)

Definition 4.14 (Logical Equivalence)

The formulae $A, B \in FO(S)$ are called logically equivalent, notation $A \models\!\models B$, if for all structures $\mathcal{M}$ and all assignments σ

$$\mathcal{M}[\![A]\!](\sigma) = \mathcal{M}[\![B]\!](\sigma).$$

Lemma 4.15

Logical equivalence is a congruence: if a subformula B of a formula $A \in FO(S)$ is replaced by $C \models\!\models B$, the result is $A' \models\!\models A$.

Semantics of First-Order Logic (Cont.)

Lemma 4.16 (Logical Equivalences)

Let $A, B \in FO(S)$. Then

$$\neg\forall xA \models\!\!\models \exists x\neg A \qquad \neg\exists xA \models\!\!\models \forall x\neg A \tag{8}$$

$$\forall xA \wedge \forall xB \models\!\!\models \forall x(A \wedge B) \qquad \exists xA \vee \exists xB \models\!\!\models \exists x(A \vee B) \tag{9}$$

$$\forall x\forall yA \models\!\!\models \forall y\forall xA \qquad \exists x\exists yA \models\!\!\models \exists y\exists xA. \tag{10}$$

If additionally $x \notin FV(B)$, then

$$QxA \text{ op } B \models\!\!\models Qx(A \text{ op } B) \text{ with } Q \in \{\forall, \exists\} \text{ and op} \in \{\wedge, \vee\}. \tag{11}$$

Note: The equivalences (8), (9) and (11) move quantifiers outwards.

Take care when using logical equivalences:

$$\forall xA \vee \forall xB \not\models\!\!\models \forall x(A \vee B) \qquad \exists xA \wedge \exists xB \not\models\!\!\models \exists x(A \wedge B).$$

Substitution

Susbtitutions replace variables with terms.

They are the syntactic counterpart of the semantic notion of assignments, more precisely: their modification.

Definition 4.17 (Substitution)

A substitution of the signature S is a finite mapping

$$\theta : V \to \mathit{Term}(S).$$

Substitutions are often directly denoted as $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$.

Substitution (Cont.)

Using substitutions on terms and formulae avoids introducing new bindings.

Definition 4.18 (Application of Substitutions)

The application of θ on $t \in \mathit{Term}(S)$ yields a new term $t\theta \in \mathit{Term}(S)$, which is inductively defined as follows:

$$x\theta := \theta(x) \qquad f(t_1, \ldots, t_n)\theta := f(t_1\theta, \ldots, t_n\theta).$$

For $A \in \mathit{FO}(S)$ the application of θ yields the formula $A\theta \in \mathit{FO}(S)$ with

$$(t_1 = t_2)\theta := t_1\theta = t_2\theta \qquad (\neg A)\theta := \neg(A\theta)$$
$$p(t_1, \ldots, t_n)\theta := p(t_1\theta, \ldots, t_n\theta) \qquad (A \text{ op } B)\theta := A\theta \text{ op } B\theta$$
$$(Qx.A)\theta := Qy(A\{x/y\}\theta),$$

where $y \notin V(A) \cup \mathit{Dom}(\theta) \cup \mathit{Ran}(\theta)$.

Substitution (Cont.)

The relation between substitutions and the modification of assignments is the following:

Lemma 4.19 (Substitution Lemma)

$\mathcal{M}[\![A\{x/t\}]\!](\sigma) = \mathcal{M}[\![A]\!](\sigma\{x/\mathcal{M}[\![t]\!](\sigma)\})$.

Proof via induction over the structure of terms and formulae.

Corollary 4.20

i) *If $A \in FO(S)$ is valid, then so is $A\{x/t\}$.*

ii) *The formula $\forall x.A \to A\{x/t\}$ is valid.*

Substitution (Cont.)

Similar to the substitution lemma one gets:

Lemma 4.21 (Bound Renaming preserves Logical Equivalence)

$QxA \models\models Qy(A\{x/y\})$.

Note: Bound renaming can make the occurrences of bound variables in a formula unique.

Normal Forms

Generate formulae of simpler structure, for which statements are easier to prove and more efficient algorithms can be designed.

Prenex normal form: All quantifiers are at the front of the formula (up to logical equivalence).
Skolem normal form: Prenex normal form and uses only universal quantifiers
(up to equisatisfiability).

Lemma 4.22 (Existential and Universal Closure)

Consider $A \in FO(S)$ with $FV(A) = \{x_1, \ldots, x_n\}$. Then

$$A \text{ is valid} \quad \text{iff} \quad \forall x_1 \ldots \forall x_n.A \text{ is valid}$$
$$A \text{ is satisfiable} \quad \text{iff} \quad \exists x_1 \ldots \exists x_n.A \text{ is satisfiable.}$$

The formula $\forall x_1 \ldots \forall x_n.A$ is the universal closure of A.
The formula $\exists x_1 \ldots \exists x_n.A$ is the existential closure of A.

Normal Forms (Cont.)

A formula $A \in FO(S)$ is called cleansed if

i) no variable occurs freely and bound and

ii) every variable is bound at most once.

By repeated application of bound renaming in Lemma 4.21 every formula can be turned into a cleansed formula.

Lemma 4.23

For every formula $A \in FO(S)$ there is a cleansed formula $B \in FO(S)$ with $A \models\!\!\models B$.

Normal Forms (Cont.)

Next goal: Move quantifiers outwards.
Trick: Use the equivalences from Lemma 4.16.

Definition 4.24

A formula of the shape $A \equiv Q_1 y_1 \dots Q_n y_n . B$ is in prenex normal form, where $Q_1, \dots, Q_n \in \{\forall, \exists\}$ and B quantifier-free.

We say $A \in FO(S)$ is in CPF if A is cleansed and in prenex normal form.

Theorem 4.25

For every formula $A \in FO(S)$ there is a formula $B \in FO(S)$ in CPF with $A \models\!\!\models B$.

The proof (see blackboard) is based on a recursive algorithm.
Try to work out this algorithm on your own as an exercise.

Normal Forms (Cont.)

Final step: Eliminate existential quantifiers.

Trick: Turn the nesting of the quantifiers for all $y_1 \dots y_n$ exists a z into a function $z = f(y_1, \dots, y_n)$:

$$\forall y_1 \dots \forall y_n \exists z.A \quad \text{yields} \quad \forall y_1 \dots \forall y_n.\,(A\{z/f(y_1, \dots, y_n)\})\,.$$

Here $f/_n$ is a fresh function symbol from the set Sko of Skolem functions. Fresh means Sko is disjoint from S.

The introduction of skolem functions for existentially quantified variables is called skolemization.

Skolemization preserves only equisatisfiability, logical equivalence is lost.

Skolemization is due to Thoralf Albert Skolem (1887 – 1963).

Normal Forms (Cont.)

Definition 4.26 (Skolem Formula)

For a formula $A \in FO(S)$ in CPF, the Skolem formula $B \in FO(S \uplus Sko)$ (again in CPF) is defined by the following method:

while A has existential quantifiers **do**
  Let $A \equiv \forall y_1 \ldots \forall y_n \exists z.B$ with B in CPF
  Let $f/n \in Sko$ be a Skolem symbol not in B
  Set $A \equiv \forall y_1 \ldots \forall y_n (B\{z/f(y_1, \ldots, y_n)\})$
end while

Note: Skolem functions are introduced from outside to inside.

Theorem 4.27 (Skolem)

For every formula $A \in FO(S)$ in CPF and the corresponding Skolem formula $B \in FO(S \uplus Sko)$:

$$A \text{ is satisfiable} \quad \text{iff} \quad B \text{ is satisfiable.}$$

The Problem of Validity

Consider computability of the **problem of validity**:

Given: A formula $A \in FO(S)$.
Question: Is A valid?

Goal: Validity is **complete in the class of semi-decidable problems**.
More precisely:

Upper bound:
Validity is semi-decidable.

Lower bound:
The problem of validity is hard in the class of semi-decidable problems.

In particular, validity is **undecidable**.

Herbrand Theory

To show semi-decidability of validity, use

$$A \in FO(S) \text{ is valid} \quad \text{iff} \quad \neg A \text{ is unsatisfiable.}$$

Goal: Unsatisfiability is semi-decidable.

Problem: When choosing $\mathcal{M} = (D, I)$, the domain is arbitrary.

- No statement about the cardinality of D.
- No information about the structure of I.

How to enumerate structures and check for model property?

Core idea: The search for models can be restricted to **canonical structures**.
To find a model for A, **it suffices** to search in the following domain:

$$D_{\mathcal{H}} = \text{All variable-free terms over signature } S.$$

Herbrand Theory (Cont.)

Assumption: $FO^{\neq}(S)$ with $S = (Func, Pred)$, where $Func$ contains a constant.

Definition 4.28 (Herbrand Structure)

A structure $\mathcal{H}$ of S is called Herbrand structure if $\mathcal{H} = (D_{\mathcal{H}}, I_{\mathcal{H}})$.
Here $D_{\mathcal{H}}$ is the smallest set that satisfies:

i) If $a/_0 \in Func$, then $a \in D_{\mathcal{H}}$

ii) If $f/_n \in Func$ and $t_1, \ldots, t_n \in D_{\mathcal{H}}$, then $f(t_1, \ldots, t_n) \in D_{\mathcal{H}}$.

The interpretation $I_{\mathcal{H}}(f) : D_{\mathcal{H}}^n \to D_{\mathcal{H}}$ of the function symbols $f/_n \in Func$ is defined as

$$I_{\mathcal{H}}(f)(t_1, \ldots, t_n) := f(t_1, \ldots, t_n).$$

The interpretation of the predicate symbols is still open, a Herbrand structure only needs to satisfy these two restrictions.

Herbrand Theory (Cont.)

Consider a closed formula $A \in FO^{\neq}(S)$. A Herbrand structure $\mathcal{H}$ with $\mathcal{H} \models A$ is also called **Herbrand model** of A.

Theorem 4.29 (Herbrand)

Let $A \in FO^{\neq}(S)$ be a closed formula in Skolem normal form. Then

A is satisfiable **iff** *A has a* **Herbrand model**.

Corollary 4.30 (Löwenheim-Skolem Theorem)

Let $A \in FO(S)$ be satisfiable. Then A has a model $\mathcal{M} = (D, I)$ with a **countable** *domain D.*

Semi-Decidability of Validity

Definition 4.31 (Herbrand Expansion)

Let $A \equiv \forall y_1 \ldots \forall y_n . B \in FO^{\neq}(S)$ be closed and in Skolem normal form. Then the Herbrand expansion $E(A)$ of A is defined as

$$E(A) := \{B\{y_1/t_1\} \ldots \{y_n/t_n\} \quad | \quad t_1, \ldots, t_n \in D_{\mathcal{H}}\}.$$

So, all variables in B are replaced by terms in $D_{\mathcal{H}}$

Observation:

- The formulae in $E(A)$ can be treated like propositional formulae, as they do not contain variables.
- Consider Herbrand structure for interpreting the formulae in $E(A)$.
- It gives truth values for the propositional variables in $E(A)$.

Semi-Decidability of Validity (Cont.)

Theorem 4.32 (Gödel-Herbrand-Skolem)

For a closed formula $A \in FO^{\neq}(S)$ in Skolem normal form:

$$A \text{ is satisfiable} \quad \text{iff} \quad E(A) \text{ is satisfiable in propositional logic.}$$

Intuition: The predicate formula A is approximated by the propositional formulae in $E(A)$.

Combine Theorem 4.32 with the compactness theorem of propositional logic.

Corollary 4.33

*A closed formula $A \in FO^{\neq}(S)$ in Skolem normal form is unsatisfiable iff there is a **finite** subset of $E(A)$ which is unsatisfiable.*

Semi-Decidability of Validity (Cont.)

From this follows the semi-decidability of validity:

$$A \in FO(S) \text{ is valid} \quad \text{iff} \quad \neg A \text{ is unsatisfiable.}$$

Transform $\neg A$ into a closed formula $B \in FO^{\neq}(S \uplus Sko)$ in Skolem normal fom.

Above argumentation yields **Gilmore's algorithm**, which semi-decides the unsatisfiability of B.

Semi-Decidability of Validity (Cont.)

Gilmore's Algorithm:

Input: $A \in FO^{\neq}(S \uplus Sko)$ closed and in Skolem normal form.
Let $E(A) = \{A_1, A_2, \ldots\}$ be an enumeration of $E(A)$.

n:=1
while $A_1 \wedge \ldots \wedge A_n$ is propositionally satisfiable **do**
  n:=n+1
end while
return unsatisfiable

With corollary 4.33:
Terminates and returns correct result on unsatisfiable formulae.
Does not terminate on satisfiable formulae.

Theorem 4.34

The problem of validity is semi-decidable.

Semi-Decidability of Validity (Cont.)

Note that the semi-decidability of validity does **not** imply the decidability via negation of the formula.

$$\not\models A \quad \text{is } \textbf{not} \text{ equivalent to} \quad \models \neg A.$$

Only the latter can be checked via Herbrand expansion.

Now, show **undecidability** of validity.

Lower Bound for Validity

Goal: The problem of validity is **hard** in the class of semi-decidable problems.

I.e. every semi-decidable problem has a many-one reduction to validity.

Consequence: Validity is **undecidable** (halting problem).

Definition 4.35 (Many-one Reduction)

A **many-one reduction** of a problem P_1 to a problem P_2 is a total and computable function $f : P_1 \to P_2$, which maps instances of P_1 to instances of P_2 so that

Instance K of P_1 has a solution iff instance $f(K)$ of P_2 has a solution.

Lower Bound for Validity (Cont.)

How to prove Hardness of validity? It is a universally quantified statement.

Consider a problem that is already known to be hard. Here we choose the **Post correspondence problem (PCP)**.

Give a many-one reduction **of PCP to validity**.

Why does this reduction show hardness of validity?

Let P be a semi-decidable problem and f_P its reduction to PCP. The reduction f_P exists, since PCP is hard.

Let f be the reduction of PCP to validity that is yet to be found.

Then:

$$\text{P} \xrightarrow{f_P} \text{PCP} \xrightarrow{f} \text{Validity} \quad \text{implies} \quad \text{P} \xrightarrow{f \circ f_P} \text{Validity}.$$

Post Correspondence Problem

Given: A finite sequence of word pairs

$$((x_1, y_1), \ldots, (x_n, y_n)) \quad \text{mit} \quad x_i, y_i \in \{0,1\}^+.$$

Question: Is there a non-empty sequence $i_1, \ldots, i_k \in \{1, \ldots, n\}$ with

$$x_{i_1} \ldots x_{i_k} = y_{i_1} \ldots y_{i_k}.$$

A given sequence of word pairs is a **PCP instance** K.
A sequence of indices $i_1, \ldots, i_k$ that satisfies the equality above is called **solution of the instance** K.

Theorem 4.36 (Post 1946)

PCP is complete in the class of semi-decidable problems, so

(a) *PCP is semi-decidable and*

(b) *every semi-decidable problem has a many-one reduction to PCP. In particular, PCP is* **undecidable**.

Lower Bound for Validity (Cont.)

Theorem 4.37 (Church)

The problem of validity is **hard** *— and with theorem 4.34* **complete** *— in the class of semi-decidable problems.*

Corollary 4.38

The problem of validity is undecidable.

Compactness Theorem of First-Order Logic

Definition 4.39 (Semantics of Sets of Formulae)

Let S be a signature, $\Sigma \subseteq FO(S)$, $\mathcal{M} = (D, I)$ and $\sigma \in D^V$.

(i) Σ is satisfied in $\mathcal{M}$ under σ, notation $\mathcal{M}, \sigma \models \Sigma$ if for all $A \in \Sigma$ we have $\mathcal{M}[\![A]\!](\sigma) = 1$.

(ii) Σ is satisfiable if there are $\mathcal{M}$ and σ with $\mathcal{M}, \sigma \models \Sigma$.

Theorem 4.40 (Compactness Theorem)

A set of formulae $\Sigma \subseteq FO(S)$ is satisfiable iff every finite subset of Σ is satisfiable.

Logical Consequence

Definition 5.1 (Logical Consequence)

The formula $A \in FO(S)$ is a logical consequence of $\Sigma \subseteq FO(S)$, notation $\Sigma \models A$, if for all $\mathcal{M}$ and σ:

$$\mathcal{M}, \sigma \models \Sigma \quad \text{implies} \quad \mathcal{M}, \sigma \models A.$$

The set of consequences of Σ is

$$\text{Cons}(\Sigma) := \{A \in FO(S) \mid \Sigma \models A\}.$$

Sets of formulae $\Sigma \subseteq FO(S)$ and $\Gamma \subseteq FO(S)$ are equivalent, notation $\Sigma \models\!\models \Gamma$, if

$$\Sigma \models A \text{ for all } A \in \Gamma \qquad \text{and} \qquad \Gamma \models B \text{ for all } B \in \Sigma.$$

Logical Consequence (Cont.)

Note 5.2

(a) $\Sigma \models A$ *iff* $\Sigma \cup \{\neg A\}$ *not satisfiable.*

(b) $\emptyset \models A$ *iff* $\models A$, *so* A *is valid.*

(c) Σ *not satisfiable iff* $\Sigma \models A$ *for all* $A \in FO(S)$.

(d) *If* $\Gamma \subseteq \Sigma$ *and* $\Gamma \models A$, *then* $\Sigma \models A$.

(e) *If* $\Gamma \models\!\models \Sigma$, *then* Γ *is satisfiable iff* Σ *is satisfiable.*

(f) *If* $\Gamma \models\!\models \Sigma$, *then* $Cons(\Gamma) = Cons(\Sigma)$.

(g) $A \models\!\models B$ *iff* $A \models B$ *and* $B \models A$ *iff* $\models A \leftrightarrow B$ *iff* $\mathcal{M}[\![A]\!](\sigma) = \mathcal{M}[\![B]\!](\sigma)$ *for all* $\mathcal{M}, \sigma$.

(h) *If* $A \models\!\models B$, *then* $\Sigma \models A$ *iff* $\Sigma \models B$.

Examples

Example 5.3

i) $\forall x A \models A$
Special case of $\forall x A \to A\{x/t\}$ valid.

ii) In general, $A \models \forall y A$ with $y \in FV(A)$ not valid.

Let $A \equiv p(y)$ and $\mathcal{M} = (\{0,1\}, I)$ with $I(p)(a) = 1$ iff. $a = 0$.
Choose $\sigma(y) = 0$, then $\mathcal{M}[\![A]\!](\sigma) = 1$.
But $\mathcal{M}[\![\forall y A]\!](\sigma) = 0$ with $\sigma\{y/1\}$.

iii) $\models \exists x(p(x) \to \forall x p(x))$
Let $\mathcal{M} = (D, I)$. We have $\mathcal{M}[\![\exists x(p(x) \to \forall x p(x))]\!] = 1$ if

there is a $d \in D$ with $I(p)(d) = 0$ or
for all $d \in D$ we have $I(p)(d) = 1$.

One of both has to be true.

iv) $\forall x(A \to B) \models \forall x A \to \forall x B$

Logical Consequence (Cont.)

Theorem 5.4 (Important Theorems)

Let $\Gamma \subseteq FO(S)$ *and* $A, B \in FO(S)$.

Deduction theorem $\Gamma, A \models B$ *iff* $\Gamma \models A \to B$

Modus Ponens rule $\Gamma \models A$ *and* $\Gamma \models A \to B$, *then* $\Gamma \models B$

Contraposition rule $\Gamma, A \models \neg B$ *iff* $\Gamma, B \models \neg A$

Generalization theorem

If $x \in V$ *does not occur freely in any formula of* Γ, *then*

$\Gamma \models A$ *iff* $\Gamma \models \forall x\ A$

In particular: $A \models \forall x\ A$ *or* $\models A \to \forall x\ A$,

if x *does not occur freely in* A.

Logical Consequence (Cont.)

Example 5.5 (Application of Theorems)

a) $\models \exists x \forall y A \to \forall y \exists x A$

iff	$\exists x \forall y A \models \forall y \exists x A$	Deduction theorem
iff	$\exists x \forall y A \models \exists x A$	Generalization theorem
iff	$\neg \forall x \neg \forall y A \models \neg \forall x \neg A$	Note 5.2 (logical equivalence)
iff	$\forall x \neg A \models \forall x \neg \forall y A$	Contraposition rule
iff	$\forall x \neg A \models \neg \forall y A$	Generalization theorem
iff	$\{\forall x \neg A, \forall y A\}$	not satisfiable

b) Variant of congruence

A' is created from A via permitted (consider quantifiers) replacement of some occurrences of x with y. Then

$$\models \forall x \forall y (x = y \to (A \leftrightarrow A')).$$

Example: $\forall x \forall y (x = y \to (f(x, y) = g(x) \leftrightarrow f(y, y) = g(x))$

The Deductive System $\mathcal{F}$

Goal: Construct a suitable deductive system $\mathcal{F}(Ax, R)$ for first order logic.

Suitable: Soundness ($\Rightarrow$) and **completeness ($\Leftarrow$)**

$$\vdash_{\mathcal{F}} A \quad \text{iff} \quad \models A$$

$$\Sigma \vdash_{\mathcal{F}} A \quad \text{iff} \quad \Sigma \models A$$

The definition of system $\mathcal{F}$ together with the proof of completeness is a great contribution by Kurt Gödel (1906 — 1978).

The Deductive System $\mathcal{F}$ (Cont.)

Let $FO_0(S)$ the subset of formulae from $FO(S)$ over $\neg, \rightarrow, \forall, =$.

Definition 5.6 (Deductive Systems)

The **deductive system** $\mathcal{F}(Ax, R)$ for $FO_0(S)$ consists of the axioms that can be generated as generalizations of the formulae which are described by the following schemata:

Ax1: Propositional tautologies
Ax2: $\forall x\ A \rightarrow A\{x/t\}$
Ax3: $\forall x\ (A \rightarrow B) \rightarrow (\forall x\ A \rightarrow \forall x\ B)$
Ax4: $A \rightarrow \forall x\ A$ if $x \notin FV(A)$
Ax5: $x = x$
Ax6: $x = y \rightarrow (A \rightarrow A')$, where A' is created from A via replacement of some free occurrences of x with y (if permitted).

The only rule schema is **Modus Ponens**: $\dfrac{A, A \rightarrow B}{B}$

The Deductive System $\mathcal{F}$ (Cont.)

Definition 5.7 (and Note)

Let $A \in FO(S)$ and $\{x_1, \ldots, x_n\} \subseteq FV(A)$. The formula

$$\forall x_1 \ldots \forall x_n . A$$

is a generalization of A.

With theorem 2.12 all propositional tautologies can be derived from three axiom schemata via Modus Ponens.

Deduction Theorem and Generalization Theorem

Theorem 5.8

Let $\Gamma \subseteq FO(S)$ *and* $A, B \in FO(S)$.

a) **Deduction theorem**

$$\Gamma \underset{\mathcal{F}}{\vdash} A \to B \qquad \textit{iff} \qquad \Gamma, A \underset{\mathcal{F}}{\vdash} B$$

b) **Generalization theorem:**

$$\textit{If } \Gamma \underset{\mathcal{F}}{\vdash} A \textit{ and } x \textit{ does not occur freely in } \Gamma, \textit{ then } \Gamma \underset{\mathcal{F}}{\vdash} \forall x\, A$$

c) **Contraposition theorem:**

$$\Gamma, A \vdash \neg B \qquad \textit{iff} \qquad \Gamma, B \vdash \neg A.$$

Hence, the theorems known from the deductive system $\mathcal{F}_0$ of propositional logic are valid for system $\mathcal{F}$.

Consistency

Definition 5.9

A set of formulae $\Gamma \subseteq FO(S)$ is called **consistent** if there is no $A \in FO(S)$ with $\Gamma \underset{\mathcal{F}}{\vdash} A$ and $\Gamma \underset{\mathcal{F}}{\vdash} \neg A$.

Note 5.10

- Γ *is consistent iff every finite subset of* Γ *is consistent.*
- *If* Γ *is inconsistent, then* $\Gamma \underset{\mathcal{F}}{\vdash} A$ *for every formula* A*.*
- *If* $\Gamma \underset{\mathcal{F}}{\vdash} A$*, then* $\Gamma \cup \{\neg A\}$ *is inconsistent.*
- *If* Γ *is inconsistent, then* Γ *is not satisfiable.*
- *The set of valid formulae is consistent.*
- *The set of theorems of* $\mathcal{F}$ *is consistent.*

The Deductive System $\mathcal{F}$ (Cont.)

Theorem 5.11 (Soundness and Completeness of $\mathcal{F}$, Gödel)

Let $A \in FO(S)$ and $\Sigma \subseteq FO(S)$, then:

a) $\vdash_{\mathcal{F}} A$ *iff* $\models A$.

b) $\Sigma \vdash_{\mathcal{F}} A$ *iff* $\Sigma \models A$.

c) Σ *consistent* *iff* Σ *satisfiable.*

The Theorem of Predicate Logic!

Proof:

Soundness: Ax contains only valid formulae and (MP) does not lead out of the set of valid formulae.

Completeness: See Enderton. ■

First Order Theories

Consider closed formulae from $FO_{closed}(S)$.

Definition 5.12

Let S be a signature. A set of formulae $\Gamma \subseteq FO_{closed}(S)$ is called a **first order theory** if Γ is closed under logical consequence:

$$A \in FO_{closed}(S) \quad \text{and} \quad \Gamma \models A \quad \text{implies} \quad A \in \Gamma.$$

Use T as identifier for theories.

Alternative definitions in literature:

- Γ set of formulae from $FO(S)$ instead of $FO_{closed}(S)$, closed under logical consequence.
- Γ theory if Γ closed under MP and generalization.

First Order Theories (Cont.)

Note 5.13

Let S be a signature.

a) $T_S = \{A \in FO_{closed}(S) \mid A \text{ valid}\}$ *is a theory.*

b) *Let $\Sigma \subseteq FO_{closed}(S)$. Then $T_\Sigma = \{A \in FO_{closed}(S) \mid \Sigma \models A\}$ is the* **theory generated by Σ** *oder theory defined by the axioms Σ.*

c) *Let $\mathcal{M}$ be a structure of the signature S. Then*

$$T_{\mathcal{M}} = \{A \in FO_{closed}(S) \mid \mathcal{M} \models A\}$$

is the **theory of $\mathcal{M}$**. *$Th(\mathcal{M})$ is also commonly used as symbol.*

First Order Theories (Cont.)

Lemma 5.14 (and Definition)

(i) *If T is a theory and $A \in FO_{closed}(S)$, then*

$$T \underset{\mathcal{F}}{\vdash} A \quad \textit{iff} \quad A \in T.$$

(ii) *A theory T is called* **inconsistent** *if there is a formula $A \in FO_{closed}(S)$ with $T \underset{\mathcal{F}}{\vdash} A$ and $T \underset{\mathcal{F}}{\vdash} \neg A$. In this case $T = FO_{closed}(S)$.*

(iii) *$T_{\mathcal{M}}$ is* **consistent** *for every structure $\mathcal{M}$.*

(iv) *T_S ist contained in each theory over S.*

First Order Theories (Cont.)

Definition 5.15

Let T be a first order theory over signature S.

a) T is called **complete** if for every formula $A \in FO_{closed}(S)$ we have: $A \in T$ oder $\neg A \in T$.

b) T is called **(finitely, enumerably) axiomatizable** if there is a (finite, enumerable) subset $\Sigma \subseteq FO_{closed}(S)$ with

$$T_\Sigma = T.$$

c) T is called **decidable** if T is a decidable subset of $FO_{closed}(S)$.

First Order Theories (Cont.)

Note 5.16

(a) *$T_{\mathcal{M}}$ is* **complete** *for every structure $\mathcal{M}$.*
With lemma 5.14 $T_{\mathcal{M}}$ is also consistent and complete.

(b) *T is satisfiable iff T is consistent.*

(c) *If T is enumerably axiomatizable, then T is enumerable.*

(d) *If T is complete and enumerably axiomatizable, then T is* **decidable**.

(e) *If T is complete and consistent, then $T = T_{\mathcal{M}}$ for a structure $\mathcal{M}$.*

Axiomatization

Goal: Find Axiomatizations of important theories.

In particular: When is $T_{\mathcal{M}} = T_{\Sigma}$ for **enumerable** Σ.

Motivation: Decidability!

Problem: When is T_{Σ} complete for enumerable Σ?

Axiomatization (Presburger)

Consider the signature of the arithmetic **without multiplication**:

$$S_{PA} = (\{0/_0, 1/_0, +/_2\}, \{=/_2\}).$$

The corresponding structure $\mathcal{M}_{PA} = (\mathbb{N}, I_{PA})$ with the usual interpretation is called **Presburger arithmetic**.

Let Σ_{PA} the set of the following axioms, with (induction) a schema:

$$\forall x : \neg(x + 1 = 0) \qquad \text{(zero)}$$

$$\forall x : x + 0 = x \qquad \text{(plus zero)}$$

$$\forall x \forall y : x + 1 = y + 1 \to x = y \qquad \text{(successor)}$$

$$\forall x \forall y : x + (y + 1) = (x + y) + 1 \qquad \text{(plus successor)}$$

$$A(0) \wedge (\forall x : \ A(x) \to A(x + 1)) \to \forall x : \ A(x), \qquad \text{(induction)}$$

where $A \in FO(S_{PA})$ is a formula with a free variable.

Axiomatization (Presburger)

Theorem 5.17 (Complete Axiomatization of Presburger Arithmetic)

It is $T_{\mathcal{M}_{PA}} = T_{\Sigma_{PA}}$. Since Σ_{PA} is enumerable, $T_{\mathcal{M}_{PA}}$ is **decidable**.

Completeness of the axiomatization is involved.
Decidability follows with note 5.16(d).

Hence, closed formulae from $FO(S_{PA})$ can be checked **automatically** for satisfaction in Presburger arithmetic.

For example:

$$\forall w \forall x \exists y \exists z : x + 2y + 3w = z + 13 \quad ?$$

Consider the quantifiers and compare with Gauss elimination.

Axiomatization (Gödel and Peano)

Consider the signature of the full arithmetic:

$$S_{Arith} = (\{0/_0, 1/_0, +/_2, \cdot/_2\}, \{=/_2\}).$$

The corresponding structure $\mathcal{M}_{Arith} = (\mathbb{N}, I_{Arith})$ with the usual interpretation is called **(first order) arithmetic**.

Theorem 5.18 (Gödel)

$T_{\mathcal{M}_{Arith}}$ is not decidable.

*Consequence 1: $T_{\mathcal{M}_{Arith}}$ is **not** enumerably axiomatizable*

*Consequence 2: Every enumerable system of axioms for $T_{\mathcal{M}_{Arith}}$ is **incomplete**.*

The consequences follow from note 5.16(a) und (d).

Axiomatization (Gödel and Peano)

In particular the Peano axioms Σ_{Peano} are **not** a complete axiomatization of $T_{\mathcal{M}_{Arith}}$:

$$\forall x : \neg(x + 1 = 0) \qquad \text{(zero)}$$

$$\forall x : x + 0 = x \qquad \text{(plus zero)}$$

$$\forall x \forall y : x + 1 = y + 1 \rightarrow x = y \qquad \text{(successor)}$$

$$\forall x \forall y : x + (y + 1) = (x + y) + 1 \qquad \text{(plus successor)}$$

$$A(0) \wedge (\forall x : \ A(x) \rightarrow A(x + 1)) \rightarrow \forall x : \ A(x) \qquad \text{(induction)}$$

$$\forall x : x \cdot 0 = 0 \qquad \text{(times zero)}$$

$$\forall x \forall y : x \cdot (y + 1) = x \cdot y + x \qquad \text{(times successor)}$$

So there are closed formulae $A \in FO(S_{Arith})$ with $\mathcal{M}_{Arith} \models A$, for which $\Sigma_{Peano} \models A$ is **not** true.

How can this be? $T_{\Sigma_{Peano}}$ has **non-standard models**!

Axiomatization (Arrays)

Given are functions for read and write accesses on arrays:

$$S_{McC} = (\{\text{read}/_2, \text{write}/_3\}, \{=/_2\}).$$

Consider **McCarthy's array axioms** Σ_{McC}:

$$\forall x : x = x \qquad \text{(Reflexivity)}$$

$$\forall x \forall y : x = y \rightarrow y = x \qquad \text{(Symmetry)}$$

$$\forall x \forall y \forall z : x = y \wedge y = z \rightarrow x = z \qquad \text{(Transitivity)}$$

$$\forall a \forall i \forall j : i = j \rightarrow \text{read}(a, i) = \text{read}(a, j) \qquad \text{(Array congruence)}$$

$$\forall a \forall v \forall i \forall j : i = j \rightarrow \text{read}(\text{write}(a, i, v), j) = v \qquad \text{(Read-Write 1)}$$

$$\forall a \forall v \forall i \forall j : i \neq j \rightarrow \text{read}(\text{write}(a, i, v), j) = \text{read}(a, j). \qquad \text{(Read-Write 2)}$$

Theorem 5.19

*$T_{\Sigma_{McC}}$ is **not decidable**, in particular **not complete**.*
Decidable fragments are an active area of research, Aaron Bradley'06.

Algorithms of Predicate Logic

Goal: **Practical** semi-decision procedures for unsatisfiability.

Applications:

Validity: $\models A$ iff $\neg A$ unsatisfiable.

Logical Consequence: $\Sigma \models A$ iff $\Sigma \cup \{\neg A\}$ unsatisfiable.

Idea: Systematic version of Gilmore's algorithm.

Specifically generate ground formulae to derive contradictions.

Semantic Tableaux

Consider closed formulae in $FO^{\neq}(S)$, i.e. formulae without $=$.

Definition 6.1

Formulae from $FO^{\neq}(S)$ can be divided into classes:

- (Negated) atomic formulae: $p(t_1, \ldots, t_n)$, $\neg p(t_1, \ldots, t_n)$.
- α-formulae: $A \wedge B$, $\neg(A \vee B)$, $\neg(A \to B)$, $\neg\neg A$.
- β-formulae: $\neg(A \wedge B)$, $(A \vee B)$, $(A \to B)$.
- **γ-formulae**: $\forall x A$, $\neg\exists x A$.
- **δ-formulae**: $\exists x A$, $\neg\forall x A$.

Semantic Tableaux (Cont.)

Tableau Construction:

α, β-**formulae:** as usual.

γ-**formulae:**

γ	$\forall x\ A$	$\neg\exists x\ A$
$\gamma[t]$	$A\{x/t\}$	$\neg A\{x/t\},$

where t is a ground term, so it contains no variables.

δ-**formulae:**

δ	$\exists x\ A$	$\neg\forall x\ A$
$\delta[c]$	$A\{x/c\}$	$\neg A\{x/c\},$

where c is a function constant and **fresh for that branch**.

Semantic Tableaux (Cont.)

Notes regarding the construction:

δ-formulae Must be "satisfied" **only once**.

Solutions of δ-formulae must not be restricted:
an x with property A does not have to function as y with property B.

γ-formulae Must be satisfied **for all objects** that are introduced. They have to be considered **always**.

Intuition: Systematic construction of a **Herbrand model**:

- δ-Formeln are skolemized.
  Introduce as many constants as necessary.
- Choose the terms over constants as domain.
- If the signature contains no function symbols,
  those are just the constants.

Semantic Tableaux (Const.)

The proofs of **soundness** and **completeness** are analog to those of propositional logic.

Lemma 6.2

Let $A \in FO^{\neq}(S)$ be closed and τ a tableau for A. Then

$$A \text{ is satisfiable} \quad \text{iff} \quad \exists \text{ branch } \Gamma \in \tau : \Gamma \text{ is satisfiable.}$$

Semantic Tableaux (Cont.)

Definition 6.3

A set of closed formulae $\Gamma \subseteq FO(S)$ is called **complete** if

1. for every α-formula in Γ we have $\alpha_1, \alpha_2 \in \Gamma$
2. for every β-formula in Γ we have $\beta_1 \in \Gamma$ or $\beta_2 \in \Gamma$
3. for every γ-formula in Γ we have $\gamma[t] \in \Gamma$ for all $t \in D_{\mathcal{H}(S)}$
4. for every δ-formula in Γ there is a $t \in D_{\mathcal{H}(S)}$ with $\delta[t] \in \Gamma$.

The set is called **closed** if there is a $B \in FO(S)$ with $B, \neg B \in \Gamma$.
Otherwise Γ is called **open**.

Note: Introduced constants are contained in the signature S and thus also in the terms $D_{\mathcal{H}(S)}$.

Lemma 6.4 (Hintikka)

Let $\Gamma \subseteq FO^{\neq}(S)$ be complete. Then Γ is satisfiable iff Γ is open.

Semantic Tableaux (Cont.)

Theorem 6.5

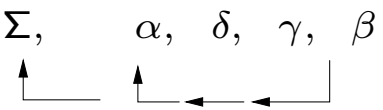
Let $A \in FO(S)$ and $\Sigma \subseteq FO(S)$.

a) $\models A$ *iff* *there is a closed tableau for* $\neg A$.

b) $\Sigma \models A$ *iff* *there is a closed tableau for* $\Sigma \cup \{\neg A\}$.

A **systematic tableau construction** guarantees that all branches are complete (possibly infinite).

Idea of a systematic tableau construction:



Assuming such a systematic tableau construction, we get a **semi-decision procedure** for validity.

Theorem 6.6

If $A \in FO(S)$ is valid, the systematic tableau construction generates a closed tableau for $\neg A$.

Semantic Tableaux (Cont.)

Tableaux are **not a decision procedure** for validity.

See **undecidability** in theorem 4.37.

Since the procedure is sound and complete, **it possibly will not terminate**.

$D_{\mathcal{H}(S)}$ can be infinite: Function symbols.

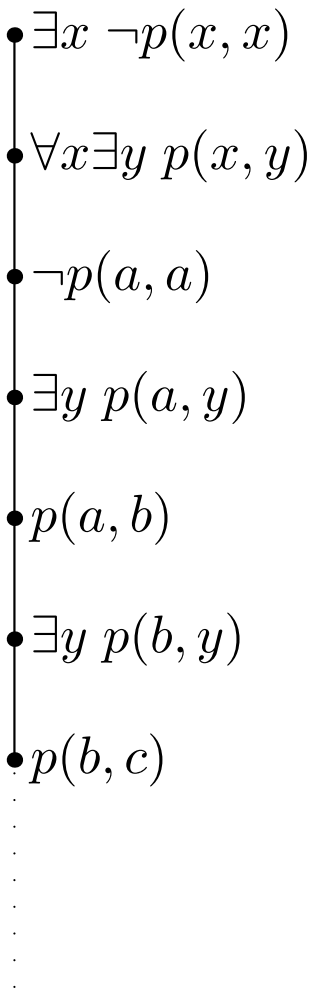
Heuristic for the construction of finite models:

Weaken the requirement **fresh** for the δ-formulae.

- Use existing constants first.
- If this choice leads to contradictions, introduce new constants.
- Otherwise model has been found.

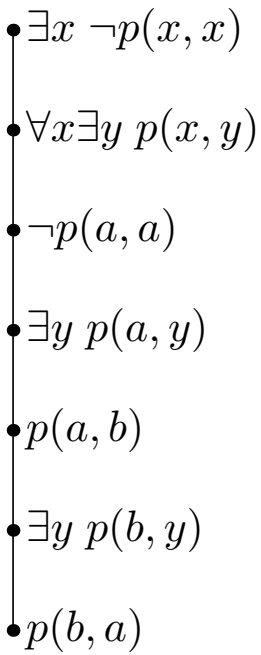
Examples for Reuse of Constants

Are there models for $\{\exists x\ \neg p(x,x), \forall x \exists y\ p(x,y)\}$?



Examples for Reuse of Constants (Cont.)

Reuse constant a:

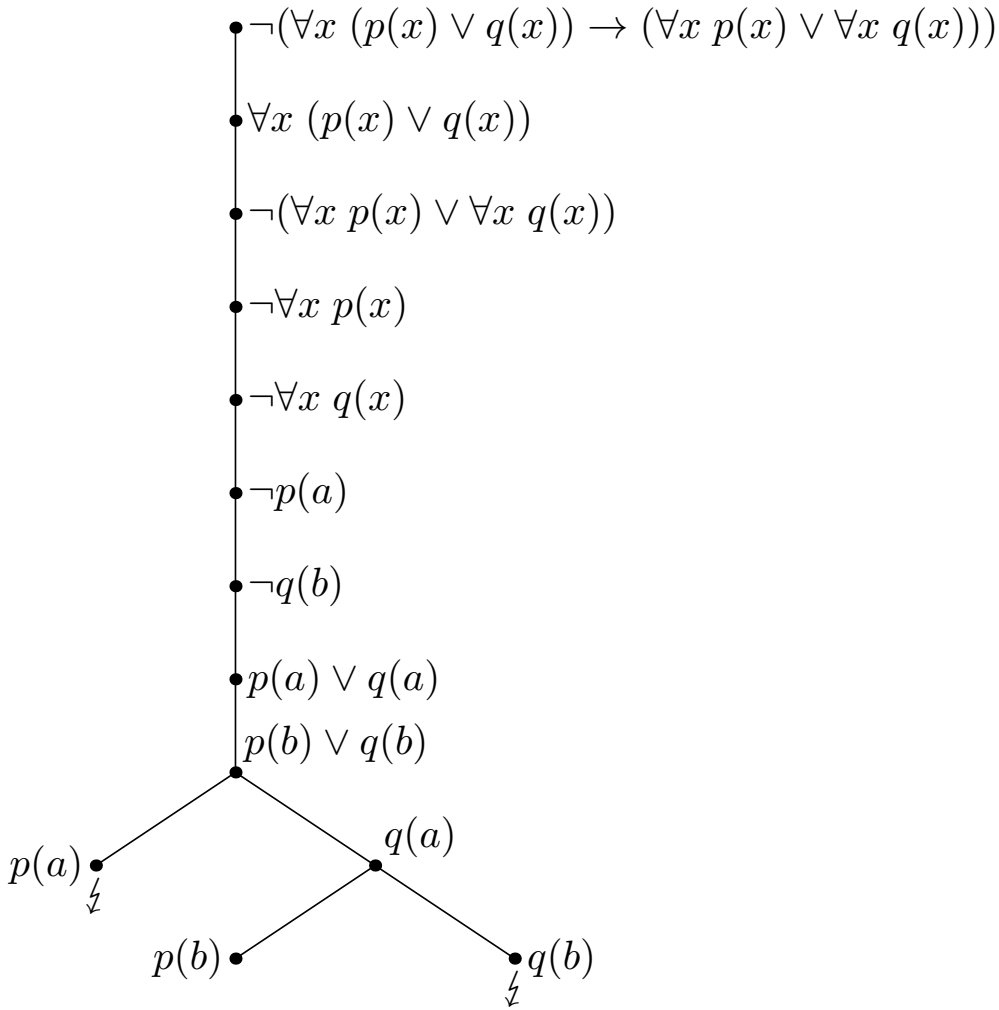


So there is a structure with two elements $\{a, b\}$ that is a model.
Interpretation of the predicate:

$p(x,y)$	a	b
a	0	1
b	1	$*$

Examples for Reuse of Constants (Cont.)

Is $\models \forall x(p(x) \vee q(x)) \rightarrow (\forall x p(x) \vee \forall x q(x))$?



$\mathcal{M} = (\{a, b\}, \quad I(p)(a) = I(q)(b) = 0, \quad I(p)(b) = I(q)(a) = 1)$

Idea of Predicate Logical Resolution

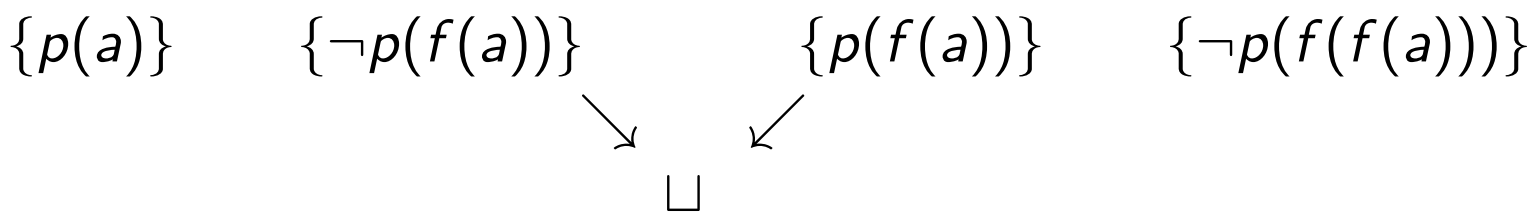
Goal: **Practical** semi-decision procedure for unsatisfiability based on Gilmore's algorithm:

To show unsatisfiability of $A \equiv \forall x_1 \ldots \forall x_n.B \in FO(S)$ in Skolem normal form, show unsatisfiability of the Herbrand expansion $E(A)$.

Example: Let $A \equiv \forall x.p(x) \wedge \neg p(f(x))$ over $S = (\{a/_0, f/_1\}, \{p/_1\})$. Then

$$E(A) = \{p(a) \wedge \neg p(f(a)), p(f(a)) \wedge \neg p(f(f(a))), \ldots\}.$$

Observation: Since $A \equiv \forall x_1 \ldots \forall x_n.B$ with B in **CNF**, unsatisfiability of $E(A)$ can be checked via **propositional resolution**:



Idea of Predicate Logical Resolution (Cont.)

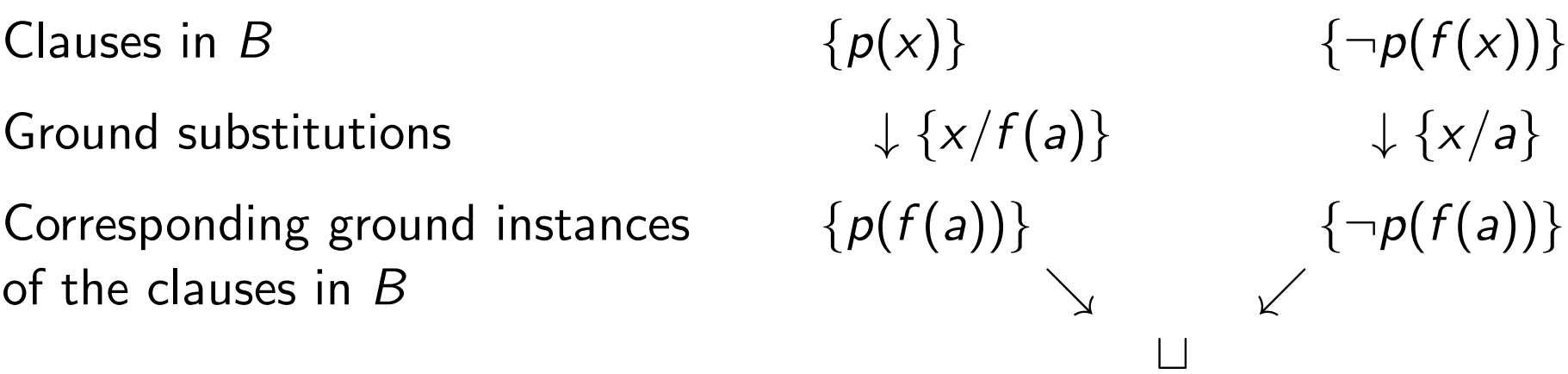
Observation: The substitutions $\{x/a\}$ and $\{x/f(a)\}$ already yield an unsatisfiable set of formulae.

But here two clauses are being generated which **aren't needed** for deriving the empty clause $\square$.

Idea: Generate fitting substitution for every clause in B — **individually**.

Apply the substitution only to this clause, not the entire B.

Example:



Idea of Predicate Logical Resolution (Cont.)

Problem: Algorithmic search for ground instances for the derivation of the empty clause $\sqcup$.

- Systematic testing of ground substitutions — expensive.
- Predictive decision for ground substitutions to enable resolutions that are needed later — hard.

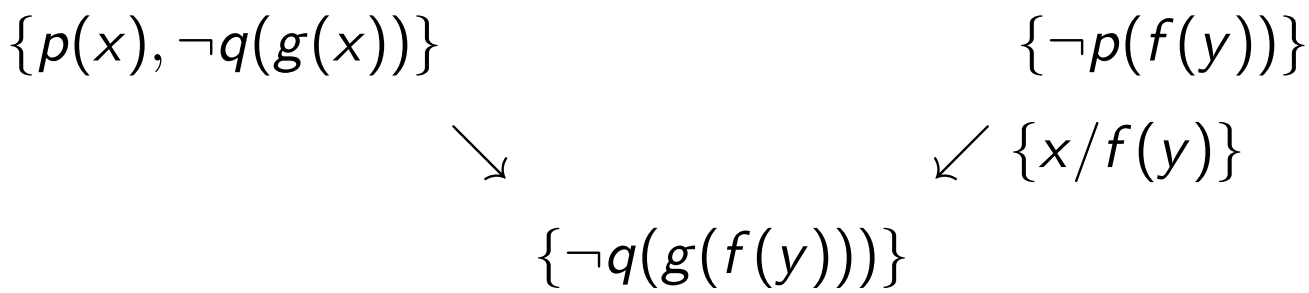
Approach: Apply substitutions reluctantly — only if they are needed for the next resolution step.

Example:

$$\{p(x), \neg q(g(x))\} \qquad \{\neg p(f(y))\}$$
$$\searrow \qquad \swarrow \{x/f(y)\}$$
$$\{\neg q(g(f(y)))\}$$

Idea of Predicate Logical Resolution (Cont.)

Example:



What happens?

Generate predicate logical resolvent from predicate logical clauses.

Resolution step comes with substitution, which makes literals in initial clauses complementary.

Apply substitutions reluctantly, no need for ground substitutions.

Unification

Goal: Compute unifier — a substitution that makes a set of literals identical.

Example: For $\{p(x), p(f(y))\}$ are

$$\Theta_1 = \{x/f(y)\} \quad \text{and} \quad \Theta_2 = \{x/f(a), y/a\}$$

unifiers. But Θ_2 substitutes more than necessary.

Definition 6.7 (Unifier)

A substitution $\Theta : \{x_1, \ldots, x_n\} \to \{t_1, \ldots, t_n\}$ is **unifier of a set of literals** $\{L_1, \ldots, L_n\}$ if

$$L_1\Theta \equiv \ldots \equiv L_n\Theta.$$

If Θ exists, the set of literals is called **unifiable**.

Unification (Cont.)

Definition 6.7 (Unifier (Cont.))

A unifier Θ of $\{L_1, \ldots, L_n\}$ is called **most general unifier** if for every unifier Θ' of $\{L_1, \ldots, L_n\}$ there is a substitution $\tilde{\Theta}$ so that

$$\Theta' = \Theta\tilde{\Theta}.$$

Visually, the following is true for a most general unifier:

$$\begin{array}{ccc} A & \xrightarrow{\Theta} & A\Theta \\ & {\scriptstyle\Theta'}\searrow & \downarrow \tilde{\Theta} \\ & & A\Theta' \equiv A\Theta\tilde{\Theta} \end{array}$$

for every formula $A \in FO(S)$.

Theorem 6.8 (Unification, Robinson)

Every unifiable set of literals has a most general unifier.

Unification Algorithm

Input: $\{L_1, \ldots, L_n\}$.

$\Theta := \{\}$
while $\exists i, j : L_i\Theta \not\equiv L_j\Theta$ **do**
  go through literals $L_1\Theta, \ldots, L_n\Theta$ from left to right,
  until first position is found where $L_k\Theta \not\equiv L_m\Theta$.
  if none of the symbols is a variable **then**
    return not unifiable
  end if
  let $x =$ the variable
  let $t =$ the term in the other literal
  if $x \in V(t)$ **then** //Occur Check
    return not unifiable
  end if
  $\Theta := \Theta\{x/t\}$
end while
return Θ

When the algorithm terminates positively, Θ is a most general unifier.

Resolution

Definition 6.9 (Resolvent)

Let K_1, K_2 be predicate logical clauses with disjoint variables.
If there are literals $L_1, \ldots, L_m \in K_1$ and $L'_1, \ldots, L'_n \in K_2$, so that

$$\{\overline{L_1}, \ldots, \overline{L_m}, L'_1, \ldots, L'_n\}$$

is unifiable with the most general unifier Θ, then

$$R := \big((K_1 \setminus \{L_1, \ldots, L_m\}) \cup (K_2 \setminus \{L'_1, \ldots, L'_n\})\big)\Theta$$

is called the **predicate logical resolvent of K_1 and K_2**.

Notation: $K_1, K_2 \underset{Res}{\vdash} R$ or

$$\begin{array}{ccc} K_1 & & K_2 \\ & \searrow \quad \swarrow \Theta & \\ & R & \end{array}$$

Note: Propositional resolution is a special case with $m = n = 1$ and $\Theta = \{\}$.

Resolution (Cont.)

Example:

$\{p(f(x)), \neg q(z), p(z)\}$ $\qquad$ $\{\neg p(y), r(g(y), a)\}$

$\searrow$ $\qquad$ $\swarrow \Theta = \{z/f(x), y/f(x)\}$

$\{\neg q(f(x)), r(g(f(x)), a)\}$

Theorem 6.10 (Soundness and Refutation Completeness, Robinson)

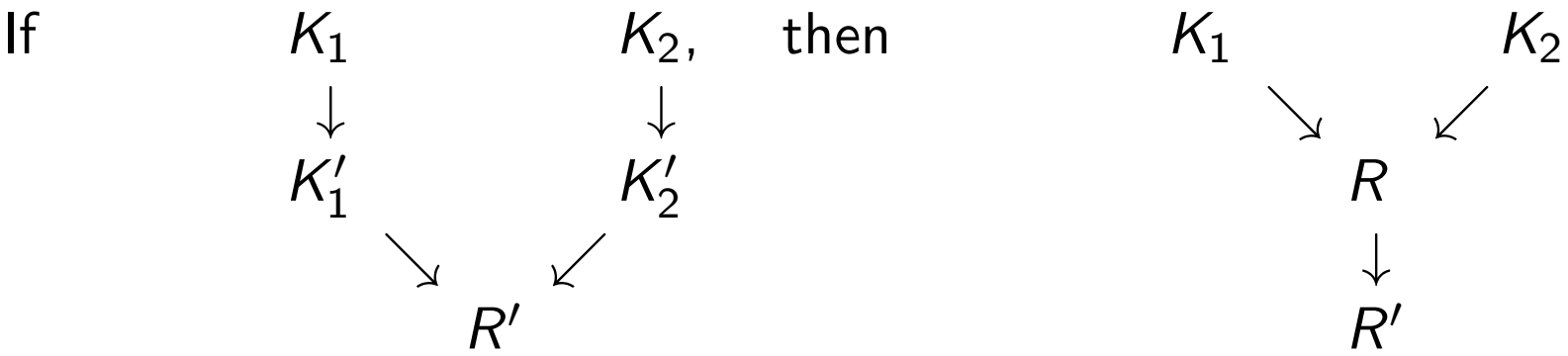
Let $A \equiv \forall x_1 \ldots \forall x_n . B \in FO(S)$ be in Skolem normal form with B in CNF. Then A is unsatisfiable iff $B \vdash_{Res} \sqcup$.

Note: The procedure does not necessarily terminate.
Unsatisfiability is undecidable.

Regarding the Proof of Refutation Completeness

Proof approach: Reduce predicate logical resolution to propositional ground resolution (as introduced above).

Technique: Propositional resolutions of ground instances can be lifted in predicate logical resolutions:



Lemma 6.11 (Lifting Lemma)

Let K_1, K_2 predicate logical clauses and K_1', K_2' ground instances with propositional resolvent R'.
Then there is a predicate logical resolvent R from K_1, K_2 so that R' is a ground instance of R.