Logic

David Suendermann

http://suendermann.com

Baden-Wuerttemberg Cooperative State University
Stuttgart, Germany

- **The most up-to-date version of this document as well as auxiliary material can be found online at**

  `http://suendermann.com`

- **propositional logic**
- **first-order logic**
- **Prolog**

- **propositional logic**
- **first-order logic**
- **Prolog**

- **Propositional logic (aka propositional calculus) is a system of formulas representing propositions.**
- **We are using the following logical operators (aka logical connectives):**
  - **$\neg$ (not)**
  - **$\wedge$ (and)**
  - **$\vee$ (or)**
  - **$\longrightarrow$ (if ... then)**
  - **$\leftrightarrow$ (if and only if (aka iff))**

- In order to build valid (well-formulated) formulas (WFFs) in propositional logic, we start with a set of propositional variables $P = \{p_1, \ldots, p_N\}$.
- Given P, we can derive the set of WWFs F inductively as follows:
  1. $1 \in F$ (true)
  2. $0 \in F$ (false)
  3. if $p \in P$ then $p \in F$ (every variable is a WFF)
  4. if $f \in F$ then $\neg f \in F$ (the negation of a WFF is a WFF)
  5. if $f, g \in F$ then
     a) $(f \wedge g) \in F$
     b) $(f \vee g) \in F$
     c) $(f \rightarrow g) \in F$
     d) $(f \leftrightarrow g) \in F$

- **Assume the following variables**

$$P = \{p, q, r\}. \tag{1}$$

- **These are WWFs:**

p

$(p \wedge q)$

$((\neg p \rightarrow q) \vee (q \rightarrow \neg r))$

these are not:

$p\neg$

$(p \leftarrow q)$

$\rightarrow q$

- **Outermost parenthesis can be dropped:**

$$(p \wedge q) \quad \Leftrightarrow \quad p \wedge q \tag{2}$$

- **Consider the following precedences:**

operator	precedence
$\neg$	**1 (strongest)**
$\wedge$	**2**
$\vee$	**3**
$\rightarrow$	**4**
$\leftrightarrow$	**5 (weakest)**

- **E.g., we have:**

$$(p \wedge q) \rightarrow (q \vee r) \quad \Leftrightarrow \quad p \wedge q \rightarrow q \vee r \tag{3}$$

- **Assume operators of the same precedence to be left-associative:**

$$(p \rightarrow q) \rightarrow r \quad \Leftrightarrow \quad p \rightarrow q \rightarrow r \tag{4}$$

- **Set parentheses to indicate the order of evaluation:**

$$p \leftrightarrow q \vee r \wedge s \longrightarrow t \wedge \neg u \wedge v \leftrightarrow w \tag{5}$$

- **Drop as many parentheses as possible:**

$$((\neg\neg((((((p \leftrightarrow q) \vee r) \vee s) \vee t) \leftrightarrow u) \wedge v) \wedge w) \wedge x) \tag{6}$$

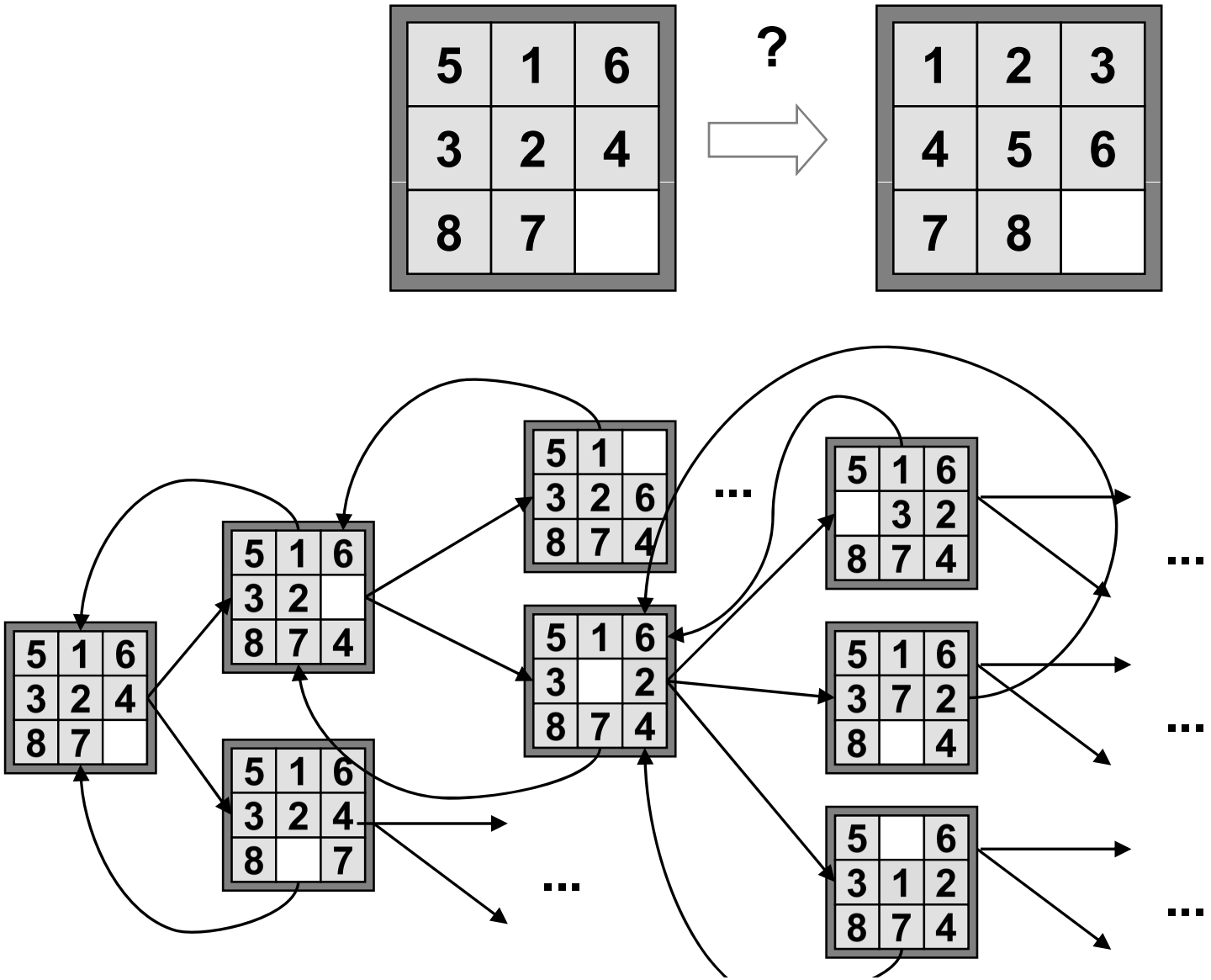
- **Design and analysis of digital circuits**
  - Several million logical gates (physical realizations of logical connectives) are implemented in nowadays' micropocessors.
  - A circuit comparison algorithm checks whether the propositional formulas representing two circuits are equivalent.
- **Planning**
  - In many planning tasks (such as in logistics, train or airline scheduling), multiple compulsory restrictions apply.
  - These restrictions can often be expressed in terms of propositional logic.
  - Optimal solutions may be determined by means of logical resolution techniques (we will learn about this later in this lecture).

- **Computer-assisted proof**
  - Often, proofs of conjectures in a mathematical discipline may be very complex (100s of pages).
  - If the discipline's axioms are given in form of logical formulas (the knowledge base), the (in)validity of conjectures may be proven by resolution.
  - This approach allows for proofs of a complexity humans are not able to handle (1,000,000s of pages).
- **Game theory**
  - Many one-, two-, or multi-player games can be expressed in terms of formulas of propositional logic.
  - Examples include chess, the 8-puzzle, or the 8-queens puzzle.
  - Again, resolution may be applied to solve these puzzles or derive "optimal" solutions.

Deep Blue

- chess-playing computer by IBM
- On May 11, 1997, Deep Blue won a six-game match against Garry Kasparov.
- based on brute-force computing power (30 nodes with 480 VLSI chess chips)
- written in C under AIX
- The evaluation function contained multiple parameters tuned on 700,000 grandmaster games.

- pic:
  - source: `http://flickr.com/photos/22453761@N00/592436598/`
  - author: James the photographer
  - license: Creative Commons Attribution 2.0 Generic

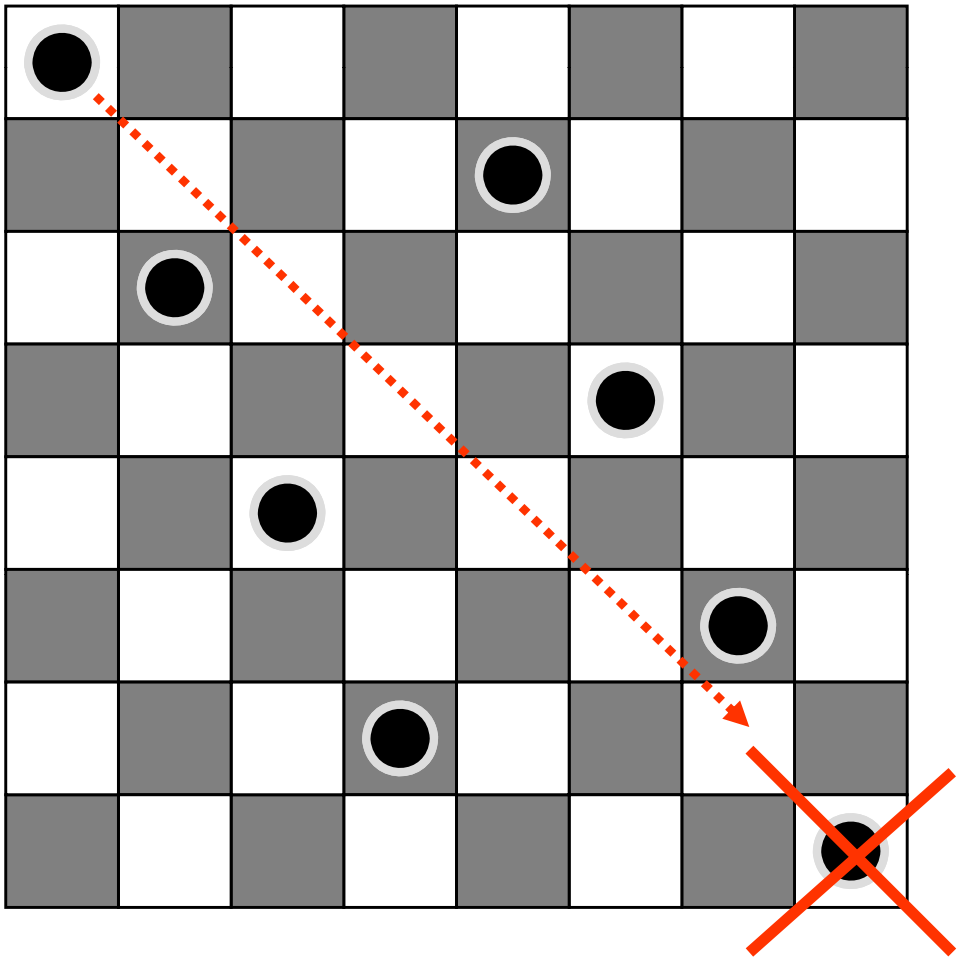

5 1 6
3 2 4
8 7
?
1 2 3
4 5 6
7 8
5 1 6
3 2 4
8 7
5 1 6
3 2
8 7 4
5 1
3 2 6
8 7 4
5 1 6
3 2 4
8 7
5 1 6
3
8 7 4
5 1 6
3 2
8 7 4
5 1 6
3 7 2
8 4
5 6
3 1 2
8 7 4
...

- Place 8 chess queens on a chessboard that no two queens attack each other.
- There are

$$\binom{64}{8} = 4,426,165,368$$

possible arrangments.

- But only 92 solutions.



- So far, we have investigated the syntax (structure) of propositional formulas.
- Now, we want to look at the semantics, that is the meaning, or truth, of propositional formulas.
- In general, without further knowledge about the variables of a formula, we cannot tell whether a formula is true or false.
- E.g., without knowing the truth value of p and q, the formula

$$p \vee q \tag{7}$$

  may be true or false.
- To that end, we introduce the valuation, or interpretation of a formula as the function

$$I : F \longrightarrow \mathbb{B} \quad \text{with} \quad \mathbb{B} = \{0, 1\}. \tag{8}$$

- We define the valuation I inductively as follows:
  1. $I(1) = 1$
  2. $I(0) = 0$
  3. $I(p) \in \mathbb{B}$ for all $p \in P$
     (values for all $p \in P$ appearing in the formula have to be provided)
  4. $I(\neg f) = I_{\neg}(I(f))$ for all $f \in F$
  5. $I(f \wedge g) = I_{\wedge}(I(f), I(g))$ for all $f, g \in F$
  6. $I(f \vee g) = I_{\vee}(I(f), I(g))$ for all $f, g \in F$
  7. $I(f \rightarrow g) = I_{\rightarrow}(I(f), I(g))$ for all $f, g \in F$
  8. $I(f \leftrightarrow g) = I_{\leftrightarrow}(I(f), I(g))$ for all $f, g \in F$

$I_\neg(p)$:

p	0	1
	1	0

$I_\wedge(p,q)$:

q \ p	0	1
0	0	0
1	0	1

$I_\vee(p,q)$:

q \ p	0	1
0	0	1
1	1	1

$I_\rightarrow(p,q)$:

q \ p	0	1
0	1	0
1	1	1

$I_\leftrightarrow(p,q)$:

q \ p	0	1
0	1	0
1	0	1

- We are given the formula

$$f := p \to q \to (\neg p \to q \to q). \tag{9}$$

- Now, we want to evaluate f for all possible interpretations of p and q.
- A handy way to do so is to use a truth table:

p	q	$g := \neg p$	$h := g \to q$	$i := h \to q$	$j := p \to q$	$j \to i$
0	0	1	0	1	1	1
0	1	1	1	1	1	1
1	0	0	1	0	0	1
1	1	0	1	1	1	1

with

$$\underbrace{p \to q}_{j} \to \underbrace{(\overbrace{\underbrace{\neg p}_{g} \to q}^{h} \to q)}_{i}. \tag{10}$$

- **Evaluate the following formulas for all possible interpretations**

  (a) $r \vee r \wedge \neg q \longrightarrow p \wedge (\neg q \wedge \neg\neg r)$

  (b) $(r \leftrightarrow p \longrightarrow p \longrightarrow r \wedge \neg\neg\neg\neg\neg\neg(\neg q \leftrightarrow q)) \wedge r$

- **Hint: You do not need to use the truth table excessively if you can apply simplifications derivable from the valuation table of logical connectives such as**

$$\neg\neg p \quad \Leftrightarrow \quad p \tag{11}$$

$$p \wedge 0 \quad \Leftrightarrow \quad 0 \tag{12}$$

$$p \wedge \neg p \quad \Leftrightarrow \quad 0 \tag{13}$$

$$p \longrightarrow p \quad \Leftrightarrow \quad 1 \tag{14}$$

$$p \leftrightarrow 0 \quad \Leftrightarrow \quad \neg p \tag{15}$$

$$p \leftrightarrow \neg p \quad \Leftrightarrow \quad 0 \tag{16}$$

- Inspector Watson is called to a jewelry store that has been subject to a robbery where three subjects, Austin, Brian, and Colin, were arrested.
- After evaluation of all facts, this is known:
  1. At least one of the subjects is guilty:
  $$f_1 := a \vee b \vee c. \tag{17}$$
  2. If Austin is guilty he had exactly one accomplice:
  $$f_2 := a \rightarrow b \wedge \neg c \vee \neg b \wedge c. \tag{18}$$
  3. If Brian is innocent, so is Colin:
  $$f_3 := \neg b \rightarrow \neg c. \tag{19}$$
  4. If exactly two subjects are guilty, Colin is one of them. Hence, out of three possible pairs of subjects, there is only one impossible:
  $$f_4 := \neg(a \wedge b \wedge \neg c). \tag{20}$$
  5. If Colin is innocent then Austin is guilty:
  $$f_5 := \neg c \rightarrow a. \tag{21}$$
- Exercise: Who are the culprits? (Hint: conjunctively combine $f_1, \ldots, f_5$)

- **We have seen that Formula 9 is true for every propositional valuation.**
- **Such a formula is called tautology (Ludwig Wittgenstein, 1921).**
- **Iff f is a tautology this is also denoted as**

$$\models f. \tag{22}$$

- **Wittgenstein on logic language and the mystery of the world:**

  `http://www.youtube.com/watch?v=Pv68v_reEQM`

- **Examples:**

  1. $\models p \vee \neg p$
  2. $\models p \rightarrow p$
  3. $\models p \wedge q \rightarrow p$
  4. $\models p \rightarrow p \vee q$
  5. $\models p \rightarrow 0 \leftrightarrow \neg p$
  6. $\models p \wedge q \leftrightarrow q \wedge p$

- **validity**: f is valid $\Leftrightarrow$ f is a tautology, i.e., all interpretations make f true.
- **satisfiability**: At least one interpretation makes f true.
- **unsatisfiability**, **contradiction**: All interpretations make f false.
- **contingency**: Interpretations of f are contingent upon the truth values of f's atomic parts. I.e., contingent propositions are neither necessarily true nor necessarily false.

- **Two formulas f and g are equivalent iff**

$$\models f \leftrightarrow g. \tag{23}$$

- **Examples:**

1)	$\models \neg 0 \leftrightarrow 1$	$\models \neg 1 \leftrightarrow 0$	
2)	$\models p \vee \neg p \leftrightarrow 1$	$\models p \wedge \neg p \leftrightarrow 0$	**tertium non datur**
3)	$\models p \vee 0 \leftrightarrow p$	$\models p \wedge 1 \leftrightarrow p$	**neutral element**
4)	$\models p \vee 1 \leftrightarrow 1$	$\models p \wedge 0 \leftrightarrow 0$	
5)	$\models p \wedge p \leftrightarrow p$	$\models p \vee p \leftrightarrow p$	**idempotency**
6)	$\models p \wedge q \leftrightarrow q \wedge p$	$\models p \vee q \leftrightarrow q \vee p$	**commutativity**
7)	$\models p \wedge (q \wedge r) \leftrightarrow (p \wedge q) \wedge r$	$\models p \vee (q \vee r) \leftrightarrow (p \vee q) \vee r$	**associativity**
8)	$\models \neg\neg p \leftrightarrow p$		**double-negation**
9)	$\models p \wedge (p \vee q) \leftrightarrow p$	$\models p \vee p \wedge q \leftrightarrow p$	**absorption**

Equivalence (cont.)

10)	$\models p \wedge (q \vee r) \leftrightarrow p \wedge q \vee p \wedge r$	$\models p \vee q \wedge r \leftrightarrow (p \vee q) \wedge (p \vee r)$	**distributivity**
11)	$\models \neg(p \wedge q) \leftrightarrow \neg p \vee \neg q$	$\models \neg(p \vee q) \leftrightarrow \neg p \wedge \neg q$	**de Morgan's rules**
12)	$\models (p \rightarrow q) \leftrightarrow \neg p \vee q$		**elimination of $\rightarrow$**
13)	$\models (p \leftrightarrow q) \Leftrightarrow (\neg p \vee q) \wedge (\neg q \vee p)$		**elimination of $\leftrightarrow$**

- **We are given the formula**

$$f := p \to q \to (\neg p \to q \to q). \tag{24}$$

Prove that f is a tautology using equivalence rules.

- Transformations such as the ones in the last exercise can be applied algorithmically turning an arbitrary formula into a normalized form.
- In order to show this, we need a couple of definitions.
- A propositional formula f is called a literal iff either of the following cases applies:
  1. $f = 0$ or $f = 1$.
  2. $f = p$ with $p \in P$ (positive literal)
  3. $f = \neg p$ with $p \in P$ (negative literal)
- The set of all literals is denoted as L.
- A propositional formula c is called a clause iff c has the form

$$c = l_1 \vee \cdots \vee l_m \tag{25}$$

  where $l_1, \ldots, l_m \in L$.
- That is, a clause is a disjunction of literals.
- The set of all clauses is denoted as C.

- **Due to the associativity, commutativity, and idempotency of the $\vee$ connective, in a clause c, the order of literals is arbitrary, and repeated literals can be dropped.**
- **This is why we can interprete the literals of c as the elements of a set:**

$$\{l_1, \ldots, l_m\}. \tag{26}$$

- **Due to the equivalence**

$$\models l_1 \vee \cdots \vee l_m \vee 0 \leftrightarrow l_1 \vee \cdots \vee l_m, \tag{27}$$

we have

$$\models \{l_1, \ldots, l_m, 0\} \leftrightarrow \{l_1, \ldots, l_m\} \tag{28}$$

(dropping of 0 from a clause).

- **A special case is**

$$\models \{0\} \leftrightarrow \{\} \tag{29}$$

(the empty clause).

- A clause c is called trivial, i.e.

$$\models c \tag{30}$$

iff we have one of the following cases:

1. Due to the equivalence

$$\models l_1 \vee \cdots \vee l_m \vee 1 \leftrightarrow 1 \tag{31}$$

we have

$$\models \{l_1, \ldots, l_m, 1\} \leftrightarrow \{1\} \tag{32}$$

(clause contains 1).

2. Due to the equivalence

$$\models l_1 \vee \cdots \vee l_m \vee p \vee \neg p \leftrightarrow l_1 \vee \cdots \vee l_m \vee 1 \tag{33}$$

we have (using Item 1)

$$\models \{l_1, \ldots, l_m, p, \neg p\} \leftrightarrow \{1\} \tag{34}$$

(clause contains complementary literals).

- **A formula f is in conjunctive normal form (CNF) iff f has the form**

$$f = c_1 \wedge \cdots \wedge c_n \quad \text{with} \quad c_1, \ldots, c_n \in C. \tag{35}$$

- **Due to the associativity, commutativity, and idempotency of the $\wedge$ connective, in a formula f, the order of clauses is arbitrary, and repeated clauses can be discarded.**

- **This is why we can interprete the clauses of f as the elements of a set which, in turn, are sets of literals:**

$$\{c_1, \ldots, c_n\}. \tag{36}$$

- **The formula**

$$(p \vee q \vee \neg r) \wedge (q \vee \neg r \vee p \vee q) \wedge (\neg r \vee p \vee \neg q) \tag{37}$$

is in CNF, and its set notation is

$$\{\{p, q, \neg r\}, \{p, \neg q, \neg r\}\}. \tag{38}$$

- Due to the equivalence

$$\models c_1 \wedge \cdots \wedge c_m \wedge 1 \leftrightarrow c_1 \wedge \cdots \wedge l_m, \tag{39}$$

we have

$$\models \{c_1, \ldots, c_m, 1\} \leftrightarrow \{c_1, \ldots, c_m\} \tag{40}$$

(dropping of 1 from a formula in CNF).

- A special case is

$$\models \{1\} \leftrightarrow \{\} \tag{41}$$

(empty CNF).

- That is, if $f = \{c_1, \ldots, c_n\}$ the following cases are equivalent

a) $\models f$

b) $\models c_i$ for all $i \in \{1, \ldots, n\}$

c) $c_i \leftrightarrow \{1\}$ for all $i \in \{1, \ldots, n\}$

d) $f \leftrightarrow \{\}$

(tautology properties).

- **This algorithm turns an arbitrary propositional formula f into CNF:**

  1. **Eliminate $\leftrightarrow$ by Equivalence 13:**

  $$\models (p \leftrightarrow q) \Leftrightarrow (\neg p \vee q) \wedge (\neg q \vee p) \tag{42}$$

  2. **Eliminate $\rightarrow$ by Equivalence 12:**

  $$\models (p \rightarrow q) \leftrightarrow \neg p \vee q \tag{43}$$

  3. **Simplify the formula using the Equivalences 1, 8, 11:**

     a) $\models \neg 0 \leftrightarrow 1$

     b) $\models \neg 1 \leftrightarrow 0$

     c) $\models \neg\neg p \leftrightarrow p$

     d) $\models \neg(p \wedge q) \leftrightarrow \neg p \vee \neg q$

     e) $\models \neg(p \vee q) \leftrightarrow \neg p \wedge \neg q$

     Now, the connective $\neg$ will only appear in front of propositional variables (the formula is in negation normal form).

4. **Expand the formula using the following variants of the law of distributivity (Equivalence 10):**
   - **a)** $\models p \vee q \wedge r \leftrightarrow (p \vee q) \wedge (p \vee r)$
   - **b)** $\models p \wedge q \vee r \leftrightarrow (p \vee r) \wedge (q \vee r)$

   This is to transform the formula into a conjunction of disjunctions.
5. **Transform the formula into set notation.**

- **Let us demonstrate the algorithm using the formula**

$$f := p \to q \to (\neg p \to \neg q). \tag{44}$$

1. (skipped)

2.
$$\begin{aligned} \models f &\Leftrightarrow \neg p \vee q \to (\neg p \to \neg q) \\ &\Leftrightarrow \neg(\neg p \vee q) \vee (\neg p \to \neg q) \\ &\Leftrightarrow \neg(\neg p \vee q) \vee (\neg\neg p \vee \neg q) \end{aligned} \tag{45}$$

3.
$$\begin{aligned} \models f &\Leftrightarrow \neg(\neg p \vee q) \vee (p \vee \neg q) \\ &\Leftrightarrow \neg\neg p \wedge \neg q \vee (p \vee \neg q) \\ &\Leftrightarrow p \wedge \neg q \vee (p \vee \neg q) \end{aligned} \tag{46}$$

4.
$$\models f \Leftrightarrow (p \vee (p \vee \neg q)) \wedge (\neg q \vee (p \vee \neg q)) \tag{47}$$

5.
$$\models f \Leftrightarrow \{\{p, \neg q\}\} \tag{48}$$

- In case there are not too many variables involved, the CNF can also be derived directly from the truth table.
- To demonstrate this, let us consider the above example

$$f := \underbrace{p \to q}_{g} \to \underbrace{(\neg p \to \neg q)}_{h}. \tag{49}$$

p	q	g	h	f
0	0	1	1	1
0	1	1	0	0
1	0	0	1	1
1	1	1	1	1

- From those rows where f evaluates to 0, we can derive the CNF as

$$\models f \Leftrightarrow \neg(\neg p \wedge q) \Leftrightarrow p \vee \neg q \tag{50}$$

or, alternatively, from rows turning 1 the disjunctive normal form (DNF)

$$\models f \Leftrightarrow \neg p \wedge \neg q \vee p \wedge \neg q \vee p \wedge q. \tag{51}$$

- **Transform into CNF:**

  a) $f := p \rightarrow (q \leftrightarrow r)$

  b) $g := p \rightarrow q \rightarrow r \leftrightarrow \neg s \vee t$

- **Proofs are one of the main notions of logic.**
- **Given**
  - **a set of formulas $\{f_1, \ldots, f_n\}$ (the knowledge base) and**
  - **a formula g (the conjecture),**

  we ask whether g can be proven based on (or inferred from the knowledge base.
- **That is, we ask whether**

$$f_1 \wedge \ldots \wedge f_n \rightarrow g \tag{52}$$

  is a tautology.
- **One possibility to answer this question is to convert Formula 52 into CNF and to check whether all its clauses are trivial.**

- **Given the knowledge base**

$$\begin{aligned} f_1 &:= p \rightarrow q \\ f_2 &:= q \rightarrow r \end{aligned} \tag{53}$$

prove the conjecture

$$g := p \rightarrow r \tag{54}$$

- **This is, we need to prove whether**

$$\models (p \rightarrow q) \wedge (q \rightarrow r) \rightarrow (p \rightarrow r) \tag{55}$$

- **This proof can be done using a truth table or by conversion into CNF.**
- **This specific rule is called hypothetical syllogism.**

- **Another way to prove a conjecture from a knowledge base is to use inference rules.**
- **An inference rule is a pair consisting of a set of premises $\{f_1, \ldots, f_n\}$ and a conclusion g written as**

$$\frac{f_1, \cdots, f_n}{\therefore g}$$

read as

From $f_1, \ldots, f_n$, we can infer that g.

- **simplification**

$$\frac{p \wedge q}{\therefore p}$$

- **modus ponens**

$$\frac{p \rightarrow q, p}{\therefore q}$$

- **biconditional elimination**

$$\frac{p \leftrightarrow q, p \vee q}{\therefore p \wedge q}$$

- **modus tollens**

$$\frac{\neg q, p \rightarrow q}{\therefore \neg p}$$

- **hypothetical syllogism**

$$\frac{p \rightarrow q, q \rightarrow r}{\therefore p \rightarrow r}$$

- **resolution**

$$\frac{p \vee q, \neg p \vee r}{\therefore q \vee r}$$

- **Returning to the set representation of CNF, a handy inference rule for clauses is the cut rule.**
- **Given a propositional variable p and two clauses c_1 and c_2, the cut rule is:**

$$\frac{\{p\} \cup c_1, \{\neg p\} \cup c_2}{\therefore c_1 \cup c_2}$$

- **We transform the conventional representation of the cut rule into CNF:**

$$
\begin{aligned}
f &:= (p \vee c_1) \wedge (\neg p \vee c_2) \rightarrow c_1 \vee c_2 \\
&\Leftrightarrow \neg((p \vee c_1) \wedge (\neg p \vee c_2)) \vee c_1 \vee c_2 \\
&\Leftrightarrow \neg(p \vee c_1) \vee \neg(\neg p \vee c_2) \vee c_1 \vee c_2 \\
&\Leftrightarrow \neg p \wedge \neg c_1 \vee \underbrace{p \wedge \neg c_2 \vee c_1 \vee c_2}_{a} \\
&\Leftrightarrow (\neg p \vee a) \wedge (\neg c_1 \vee a) \\
a &\Leftrightarrow \underbrace{(p \vee c_1 \vee c_2)}_{b} \wedge \underbrace{(\neg c_2 \vee c_1 \vee c_2)}_{c} \\
f &\Leftrightarrow ((\neg p \vee b) \wedge (\neg p \vee c)) \wedge ((\neg c_1 \vee b) \wedge (\neg c_1 \vee c)) \\
&\Leftrightarrow \{\{\neg p, p, c_1, c_2\}, \{\neg p, \neg c_2, c_1, c_2\}, \\
&\qquad \{\neg c_1, p, c_1, c_2\}, \{\neg c_1, \neg c_2, c_1, c_2\}\} \qquad (56)
\end{aligned}
$$

- **Since all clauses in f are trivial,**

$$\models f. \qquad (57)$$

- **Using Equivalence 12 (elimination of $\rightarrow$), the following of our inference rule examples can be regarded as special cases of the cut rule:**
  - **Modus ponens. With $c_1 := \{\}$ and $c_2 := \{q\}$, we have**

$$\frac{\{p\} \cup \{\}, \{\neg p\} \cup \{q\}}{\therefore \{\} \cup \{q\}}$$

  - **Modus tollens (show this).**
  - **Hypothetical syllogism (show this).**
  - **Resolution.**

- **Given a set of clauses M (the assumptions; a knowledge base) and a single clause f, we define a relation**

$$M \vdash f \tag{58}$$

read as

f is provable from M.

- **This relation is inductively defined as**

  a) M proves all its assumptions:

$$\text{If } f \in M \text{ then } M \vdash f. \tag{59}$$

  b) If two clauses $\{p\} \cup c_1$ and $\{\neg p\} \cup c_2$ are provable from M then the clause $c_1 \cup c_2$ is provable (applying the cut rule):

$$\text{If } M \vdash \{p\} \cup c_1 \text{ and } M \vdash \{\neg p\} \cup c_2 \text{ then } M \vdash c_1 \cup c_2. \tag{60}$$

- We want to show that

$$\{\{\neg p, q\}, \{\neg q, \neg p\}, \{\neg q, p\}, \{q, p\}\} \vdash \{0\}. \tag{61}$$

- This is one way to do so:
  1. $\{\{\neg p, q\}, \{\neg q, \neg p\}\} \vdash \{\neg p\}$
  2. $\{\{\neg q, p\}, \{q, p\}\} \vdash \{p\}$
  3. $\{\{\neg p\}, \{p\}\} \vdash \{\}$

- **We are given the following knowledge base**

$$
\begin{aligned}
f_1 &:= p \to q \\
f_2 &:= q \to r \\
f_3 &:= r \to s \\
f_4 &:= s \to t \\
f_5 &:= t \to u \\
f_6 &:= u \to p \\
f_7 &:= p \vee q \vee r \vee s \vee t \vee u \\
f_8 &:= \neg p \vee \neg q \vee \neg r \vee \neg s \vee \neg t \vee \neg u
\end{aligned} \tag{62}
$$

- **Show that 0 can be inferred from the knowledge base.**

- In multiple applications, it is necessary to determine a variable interpretation rendering all the clauses in a set C_0 true.
- That is, we want to determine whether C_0 is satisfiable and, if so, a solution, i.e., a satisfying interpretation.
- Consider the following obvious cases
  1) $C_1 = \{\{p\}, \{\neg q\}, \{r\}, \{\neg s\}, \{\neg t\}\}$ (satisfiable)
  2) $C_2 = \{\{\}, \{p\}, \{\neg q\}, \{r\}\}$ (unsatisfiable)
  3) $C_3 = \{\{p\}, \{\neg q\}, \{\neg p\}\}$ (unsatisfiable)

- A clause is called unit clause iff it contains one literal:

$$c = \{p\} \quad \text{or} \quad c = \{\neg p\} \quad \text{with} \quad p \in P. \tag{63}$$

- A set of clauses C_0 is called trivial in one of these cases:

  1) C_0 contains the empty clause:

$$\{\} \in C_0. \tag{64}$$

  That is, C_0 is unsatisfiable.

  2) C_0 contains only unit clauses of different propositional variables:

$$\forall c(c \in C_0 \rightarrow \exists p(p \in P \wedge (c = \{p\} \vee c = \{\neg p\})))\wedge$$
$$\forall p(p \in P \rightarrow \neg(\{p\} \in C_0 \wedge \{\neg p\} \in C_0)). \tag{65}$$

  That is, C_0 is satisfiable as determined by C_0's unit clauses.

- **The Davis and Putnam (DP) algorithm is to find a satisfying clause given a set of clauses by applying**
  1) **cut rule**
  2) **subsumption**
  3) **case distinction**

- **The DP algorithm uses a special case of the cut rule:**

$$\frac{\{l_1, \ldots, l_n, \neg l\}, \{l\}}{\therefore \{l_1, \ldots, l_n\}}$$

reducing the original clause length by one.

- **Generally, this operation is performed by the function**

$$\textbf{cut} : 2^C \times L \rightarrow 2^C \tag{66}$$

defined as

$$\textbf{cut}(C_0, l) = \{c \backslash \{\neg l\} | c \in C_0 \wedge \neg l \in c\}. \tag{67}$$

- **Example:**

$$\textbf{cut}(\{\{p, q\}, \{\neg p, r\}, \{\neg q\}\}, \neg p) = \{\{q\}\} \tag{68}$$

- **Assume the example**

$$C_0 := \{\{p, q, \neg r\}, \{p\}\}. \tag{69}$$

As we have

$$\models p \rightarrow p \vee q \vee \neg r, \tag{70}$$

we can drop (subsume) $\{p, q, \neg r\}$ **from** C_0**.**

- **Formally, we introduce the function**

$$\textbf{sub} : 2^C \times L \rightarrow 2^C \tag{71}$$

that subsumes all clauses in C_0 **containing the unit clause** $\{l\}$**:**

$$\textbf{sub}(C_0, l) = \{c \in C_0 | l \notin c\} \cup \{\{l\}\}. \tag{72}$$

- **Example:**

$$\textbf{sub}(\{\{p, q\}, \{\neg p, r\}, \{\neg q\}\}, \neg p) = \{\{p, q\}, \{\neg q\}, \{\neg p\}\} \tag{73}$$

- **Since cut rule and subsumption have the same argument spaces they can be combined to a general reduction function.**
- **In doing so, consider that applying the cut rule returns only cut versions of those clauses that originally contained $\neg l$.**
- **Hence, we add all those clauses *not* containing $\neg l$ before applying subsumption.**
- **This yields**

$$\textbf{red}(C_0, l) = \textbf{sub}(C_1 \cup C_2, l) \tag{74}$$

with

$$C_1 = \textbf{cut}(C_0, l) = \{c \backslash \{\neg l\} | c \in C_0 \wedge \neg l \in c\} \tag{75}$$

and

$$C_2 = \{c \in C_0 | \neg l \notin c\}. \tag{76}$$

- Assuming C_0 is not trivial, we know that applying subsumption to C_1 does not affect any of its clauses since they originally contained $\neg l$, i.e., they do not contain l.
- Hence, subsumption can only affect C_2 which is why we can write

$$\begin{aligned}\mathbf{red}(C_0, l) &= \mathbf{sub}(C_1 \cup C_2, l) \\ &= C_1 \cup \mathbf{sub}(C_2, l) \\ &= \{c \backslash \{\neg l\} | c \in C_0 \wedge \neg l \in c\} \cup \\ &\quad \{c \in C_0 | l \notin c \wedge \neg l \notin c\} \cup \{\{l\}\}. \end{aligned} \tag{77}$$

- Example:

$$\mathbf{red}(\{\{p, q\}, \{\neg p, r\}, \{\neg q\}\}, \neg p) = \{\{q\}, \{\neg q\}, \{\neg p\}\} \tag{78}$$

- **This principle is based on the following proposition:**

  C_0 **is satisfiable iff** $C_0 \cup \{\{p\}\}$ **or** $C_0 \cup \{\{\neg p\}\}$ **are satisfiable.**

  Here, C_0 **is a set of clauses;** p **is a propositional variable occurring in** C_0**.**

- **This sounds like a trivial principle since if there is a variable interpretation satisfying** C_0 **it will include the variable** p **which must be either 0 or 1 in this specific interpretation.**

- **The case distinction principle is, however, essential to the DP algorithm as it injects unit clauses required for cut rule and subsumption.**

- We are given a set of clauses C_0 and search for a propositional valuation satisfying C_0.

1. Iteratively apply $C_{i+1} := \text{red}(C_i, l)$ for all
   $l \in \{p, \neg p | p \in P\} \cap C_i, \quad i = 0, 1, \ldots$
   If C_{i+1} turns out to be trivial, stop.

2. Otherwise (case distinction), choose a variable p from C_i (should not be a unit clause since those had been tried in 1.)

   a) Iteratively apply $C_{i+1} := \text{red}(C_i \cup \{\{p\}\}, l)$ for all
   $l \in \{p, \neg p | p \in P\} \cap C_i$. If C_{i+1} is found to be satisfiable, stop.

   b) Iteratively apply $C_{i+1} := \text{red}(C_i \cup \{\{\neg p\}\}, l)$ for all
   $l \in \{p, \neg p | p \in P\} \cap C_i$. If C_{i+1} is found to be satisfiable, stop.

   c) Otherwise, C_0 is not satisfiable.

- Step 2 might need to be recursively applied multiple times until a solution (satisfiability or contradiction) is found.

- **We are given**

$$C_0 := \{\{p, q, s\}, \{\neg p, r, \neg t\}, \{r, s\}, \{\neg r, q, \neg p\}, \{\neg s, p\}$$
$$\{\neg p, \neg q, s, \neg r\}, \{p, \neg q, s\}, \{\neg r, \neg s\}, \{\neg p, \neg s\}\} \quad (79)$$

1. **(skipped—C_0 contains no unit clauses)**
2. **pick p**

a)
$$\begin{aligned} C_1 &:= \textbf{red}(C_0 \cup \{\{p\}\}, p) \\ &= \{\{r, \neg t\}, \{r, s\}, \{\neg r, q\}, \{\neg q, s, \neg r\}, \{\neg r, \neg s\}, \{\neg s\}, \{p\}\} \\ C_2 &:= \textbf{red}(C_1, \neg s) \\ &= \{\{r, \neg t\}, \{r\}, \{\neg r, q\}, \{\neg q, \neg r\}, \{\neg s\}, \{p\}\} \\ C_3 &:= \textbf{red}(C_2, r) \\ &= \{\{r\}, \{q\}, \{\neg q\}, \{\neg s\}, \{p\}\} \\ C_4 &:= \textbf{red}(C_3, q) \\ &= \{\{r\}, \{q\}, \{\}, \{\neg s\}, \{p\}\} \quad \textbf{(unsatisfiable)} \end{aligned} \quad (80)$$

b)

$$\begin{aligned} C_5 &:= \textbf{red}(C_0 \cup \{\{\neg p\}\}, \neg p) \\ &= \{\{q, s\}, \{r, s\}, \{\neg s\}, \{\neg q, s\}, \{\neg r, \neg s\}, \{\neg p\}\} \\ C_6 &:= \textbf{red}(C_5, \neg s) \\ &= \{\{q\}, \{r\}, \{\neg s\}, \{\neg q\}, \{\neg p\}\} \\ C_7 &:= \textbf{red}(C_6, q) \\ &= \{\{q\}, \{r\}, \{\neg s\}, \{\}, \{\neg p\}\} \quad \textbf{(unsatisfiable)} \end{aligned} \tag{81}$$

c) C_0 **is unsatisfiable (contradiction).**

- **Apply the DP algorithm to the provability exercise with**

  a) $C_0 := \{f_1, \ldots, f_8\}$,

  b) $C_0 := \{f_1, \ldots, f_7\}$.

- **In order to prove a conjecture based on a knowledge base, we can apply the resolution technique:**
  1. **All sentences in the knowledge base and the negation of the sentence to be proven (the conjecture) are conjunctively connected.**
  2. **The resulting sentence is transformed into CNF.**
  3. **If the empty clause can be derived after an application of the DP algorithm (or, alternatively, the provability technique), the original sentence is unsatisfiable, i.e., the conjecture follows from the knowledge base.**

- **Let us revisit the criminal case from earlier.**
- **Remember, we were given the following facts (our knowledge base):**

$$\begin{aligned} f_1 &:= a \vee b \vee c \\ f_2 &:= a \rightarrow b \wedge \neg c \vee \neg b \wedge c \\ f_3 &:= \neg b \rightarrow \neg c \\ f_4 &:= \neg(a \wedge b \wedge \neg c) \\ f_5 &:= \neg c \rightarrow a \end{aligned} \tag{82}$$

- **Dr. Watson's gut feeling is that Brian and Colin are the culprits.**
- **Prove his conjecture using the resolution technique.**

- **Using the DP algorithm, show that**
  - **the 2 queens puzzle has no solution,**
  - **the 3 queens puzzle has no solution,**
  - **the 4 queens puzzle has a solution and which one it is.**

- **propositional logic**
- **first-order logic**
- **Prolog**

- **Terms denote objects.**
- **Terms are composed of variables or function symbols:**

$$\textbf{father}(x), \quad \textbf{mother}(\textbf{isaac}), \quad x + \mathbf{7}. \tag{83}$$

with the variable x and the function symbols father, mother, isaac, $+$, 7

- **Objects can be put into relation by predicate symbols producing atomic formulas:**

$$\textbf{isBrother}(\textbf{adam}, \textbf{father}(\textbf{brian})), \quad x + \mathbf{7} < x \cdot \mathbf{7}, \quad n \in \mathbb{I}. \tag{84}$$

with the predicate symbols IsBrother, $<$, $\in$

- **Formulas can be combined using logical connectives:**

$$\textbf{isHuman}(x) \rightarrow \textbf{isMortal}(x), \quad x > 1 \rightarrow x + 7 < x \cdot 7. \tag{85}$$

- **Formulas can contain quantifiers defining the semantics (scope) of variables:**

$$\forall x(\textbf{isHuman}(x) \rightarrow \exists y(y = \textbf{mother}(x))). \tag{86}$$

- A signature is a quatruple defining a specific system of first-order logic:

$$\Sigma = \langle V, F, P, \text{arity} \rangle. \tag{87}$$

- The members of Σ are
  - the set of variables V,
  - the set of function symbols F,
  - the set of predicate symbols P, and
  - a function assigning an arity to all function and predicate symbols:

$$\text{arity} : F \cup P \longrightarrow \mathbb{I}. \tag{88}$$

- Given a signature Σ, we define the set of Σ terms T_Σ inductively as:
  1. $\forall x(x \in V \longrightarrow x \in T_\Sigma)$.
  2. If $f \in F$ and $n = \text{arity}(f)$ and $t_1, \ldots, t_n \in T_\Sigma$ then

$$f(t_1, \ldots, t_n) \in T_\Sigma. \tag{89}$$

- Special case: If $f \in F$ and $\text{arity}(f) = 0$ then we can write f instead of $f()$ (aka constant).

- We are given the signature $\Sigma_a = \langle V, F, P, \text{arity} \rangle$ with

  $$V = \{x, y, z\}$$
  $$F = \{0, 1, +, \cdot\}$$
  $$P = \{=, \leq\}$$
  $$\text{arity} = \{\langle 0, 0\rangle\ \langle 1, 0\rangle\ \langle +, 2\rangle\ \langle \cdot, 2\rangle\ \langle =, 2\rangle\ \langle \leq, 2\rangle\}$$

- Now, we can derive Σ_a terms as follows:
  1. $x, y, z \in T_{\Sigma_a}$
  2. $0, 1 \in T_{\Sigma_a}$ (0-ary function symbols)
  3. $+(1, x) \in T_{\Sigma_a}$
  4. $\cdot(+(1, x), y) \in T_{\Sigma_a}$

- In the following, we will use an infix notation for binary relation and function symbols.

- E.g., Term 4 would read

  $$(1 + x) \cdot y \tag{90}$$

- **Given the signature $\Sigma = \langle V, F, P, \text{arity} \rangle$, if $p \in P$ and $n = \text{arity}(p)$ and $t_1, \ldots, t_n \in T_\Sigma$ then**

$$p(t_1, \ldots, t_n) \in A_\Sigma, \tag{91}$$

  the set of all atomic formulas.

- **Special case: If $p \in P$ and $\text{arity}(p) = 0$ then we can write p instead of $p()$ (aka propositional variable).**

- **Continuing with our above example:**

$$= (\cdot(+(1, x), y), 0) \in A_{\Sigma_a} \tag{92}$$

  or in infix notation

$$(1 + x) \cdot y = 0 \in A_{\Sigma_a}. \tag{93}$$

- Consider an example from calculus:

$$\int_0^x yz\mathrm{d}z = \frac{1}{2}yx^2. \tag{94}$$

- Eq. 94 features the three variables x, y and t.
- Trying to substitute variables on both sides of the equation, e.g., x by y

$$\int_0^y yz\mathrm{d}z = \frac{1}{2}yy^2 \tag{95}$$

or z by y

$$\int_0^x yy\mathrm{d}y = \frac{1}{3}x^3 \stackrel{!}{\neq} \frac{1}{2}xx^2 \tag{96}$$

shows that there can be two types of variables in formulas: free (x, y) and bound (z) ones.

- The variable z in this example is bound by the differential operator d.

- **In first-order logic, variables are bound by the quantifiers $\forall$ and $\exists$.**
- **Accordingly, in the following example**

$$\forall x, y(P(x) \rightarrow Q(x, f(x), z)) \tag{97}$$

  x and y are bound variables, and z is free.

- **A formula with no free variables is called a sentence.**

- **Consider the following examples:**
  - **a)** $\forall x(F(x) \rightarrow \exists y(G(y,z)))$
  - **b)** $\forall x(F(x) \rightarrow \exists y(G(x,y)))$
  - **c)** $\exists x(y + x \leq y)$
- **Which variables are free, and which ones are bound?**
- **Which of these formulas are sentences?**

- The set of variables $\mathrm{var}(t)$ occurring in a term t is inductively defined as
  1. $\mathrm{var}(x) := \{x\}$ for all $x \in V$.
  2. $\mathrm{var}(f(t_1, \ldots, t_n)) := \mathrm{var}(t_1) \cup \cdots \cup \mathrm{var}(t_n)$.
- Given a signature Σ, in the following, we denote
  - the set of formulas as F_Σ,
  - the set of all bound variables of a formula $f \in F_\Sigma$ as $\mathrm{bound}(f)$, and
  - the set of all free variables of f as as $\mathrm{free}(f)$.
- These sets are inductively defined as

1. $0, 1 \in F_\Sigma$ (truth values) and
$$\mathrm{free}(0) = \mathrm{free}(1) = \mathrm{bound}(0) = \mathrm{bound}(1) = \{\} \tag{98}$$
2. $\forall f(f \in A_\Sigma \rightarrow f \in F_\Sigma)$ (atomic formulas) and
$$\mathrm{free}(f) = \mathrm{var}(t_1) \cup \cdots \cup \mathrm{var}(t_n); \quad \mathrm{bound}(f) = \{\} \text{ with } f = p(t_1, \ldots, t_n) \tag{99}$$

3. $\forall f(f \in \boldsymbol{F_\Sigma} \rightarrow \neg f \in \boldsymbol{F_\Sigma})$ **(negation) and**

$$\textbf{free}(\neg f) = \textbf{free}(f); \quad \textbf{bound}(\neg f) = \textbf{bound}(f) \tag{100}$$

4. $\forall f, g(f, g \in \boldsymbol{F_\Sigma} \wedge \textbf{free}(f) \cap \textbf{bound}(g) = \{\} \wedge \textbf{bound}(f) \cap \textbf{free}(g) = \{\} \rightarrow (f \vee g) \in \boldsymbol{F_\Sigma})$ **(logical connectives) and**

$$\textbf{free}(f \vee g) = \textbf{free}(f) \cup \textbf{free}(g); \quad \textbf{bound}(f \vee g) = \textbf{bound}(f) \cup \textbf{bound}(g) \tag{101}$$

applying to all logical connectives ($\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$)

5. $\forall f, x(f \in \boldsymbol{F_\Sigma} \wedge x \in \boldsymbol{V} \backslash \textbf{bound}(f) \rightarrow (\forall x(f)), (\exists x(f)) \in \boldsymbol{F_\Sigma})$ **(quantifiers) and**

$$\textbf{free}(\forall x(f)) = \textbf{free}(\exists x(f)) = \textbf{free}(f) \backslash \{x\}; \tag{102}$$

$$\textbf{bound}(\forall x(f)) = \textbf{bound}(\exists x(f)) = \textbf{bound}(f) \cup \{x\} \tag{103}$$

- In literature you encounter expressions of the form

$$\forall x \in R : \exists n \in N : x < n. \tag{104}$$

- These abbreviations can be transformed into the formerly introduced terminology by

$$\begin{aligned} \forall x \in M : f &\overset{def}{\Longleftrightarrow} \forall x(x \in M \rightarrow f) \\ \exists x \in M : f &\overset{def}{\Longleftrightarrow} \exists x(x \in M \wedge f) \end{aligned} \tag{105}$$

- Accordingly, Formula 104 can be written as

$$\forall x(x \in R \rightarrow \exists n(n \in N \wedge x < n)). \tag{106}$$

- Sequences of identical quantifiers can be abbreviated:

$$\forall x, y(f) \overset{def}{\Longleftrightarrow} \forall x(\forall y(f)). \tag{107}$$

- Quantifiers have a higher precedence than logical connectives:

$$\forall x(f) \wedge g \overset{def}{\Longleftrightarrow} (\forall x(f)) \wedge g. \tag{108}$$

- So far, we have learned how to create (well-formulated) formulas in first-order logic, i.e., we have dealed with syntax.
- The next step is the semantics of first-order logic.
- To that end, given a signature Σ, we introduce the notion of a structure

$$S = \langle U, J \rangle \tag{109}$$

with

1. the universe U, a non-empty set containing all the values that can occur when evaluating terms,
2. the interpretation J of all function and predicate symbols of Σ.

- **Formally, J is defined as**
  1. **Every function symbol $f \in F$ with the arity $n =$ arity(f) is mapped to an n-ary function**
  $$f^J : \underbrace{U \times \cdots \times U}_{n \text{ times}} \to U. \tag{110}$$
  2. **Every predicate symbol $p \in P$ with the arity $n =$ arity(f) is mapped to an n-ary relation**
  $$p^J \subseteq U^n. \tag{111}$$
  3. **If $= \in P$ then its interpretation should be natural, i.e.**
  $$=^J = \{\langle u, v\rangle | u, v \in U \wedge u = v\}. \tag{112}$$

- **Using Σ_a as in the above example, we define a structure**

$$S_a = \langle U, J \rangle \tag{113}$$

as follows:

1. $U = \{a, b\}$
2. $0^J = a$
3. $1^J = b$
4. $+^J = \{\langle\langle a, a\rangle, a\rangle, \langle\langle a, b\rangle, b\rangle, \langle\langle b, a\rangle, b\rangle, \langle\langle b, b\rangle, a\rangle\}$
5. $\cdot^J = \{\langle\langle a, a\rangle, a\rangle, \langle\langle a, b\rangle, a\rangle, \langle\langle b, a\rangle, a\rangle, \langle\langle b, b\rangle, b\rangle\}$
6. $=^J = \{\langle a, a\rangle, \langle b, b\rangle\}$
7. $\leq^J = \{\langle a, a\rangle, \langle a, b\rangle, \langle b, b\rangle\}$

- **Given a signature Σ and a structure $S = \langle U, J \rangle$, we define a variable assignment**

$$I : V \to U \tag{114}$$

assigning a value from the universe U to every variable in V.

- **E.g., using the signature Σ_a and the structure S_a, a possible variable assignment is**

$$I_a = \{\langle x, a \rangle, \langle y, b \rangle, \langle z, a \rangle\}. \tag{115}$$

- **Furthermore, we introduce the variable replacement**

$$I[x_1/c_1]\cdots[x_n/c_n](y) = \begin{cases} c_1 & : \quad y = x_1; \\ & \vdots \\ c_n & : \quad y = x_n; \\ I(y) & : \quad \text{otherwise} \end{cases} \tag{116}$$

with $x_1, \ldots, x_n, y \in V$ and $c_1, \ldots, c_n \in U$ replacing the value of a variable y by c_i if the variable happens to be identical to x_i.

- **Using the above example, we have**

$$I_a[y/a][z/b] = \{\langle x, a\rangle, \langle y, a\rangle, \langle z, b\rangle\}. \tag{117}$$

- **Given a signature Σ, a structure $S = \langle U, J \rangle$, and a variable assignment I, for every term t, its value (written as $S(I, t)$) can be derived inductively as**

  1. $\forall x(x \in V \rightarrow S(I, x) = I(x))$
  2. **If $f \in F$ and $n =$ arity(f) and $t_1, \ldots, t_n \in T_\Sigma$ then**

$$S(I, f(t_1, \ldots, t_n)) = f^J(S(I, t_1), \ldots, S(I, t_n)). \tag{118}$$

- **Exercise: Using the above signature Σ_a, structure S_a, and variable assignment I_a, what is the evaluation of Term 4:**

$$S(I_a, (1 + x) \cdot y)? \tag{119}$$

- **Given a signature Σ, a structure $S = \langle U, J \rangle$, and a variable assignment I, for every atomic formula $p(t_1, \ldots, t_n)$, its value can be derived as**

$$S(I, p(t_1, \ldots, t_n)) = \begin{cases} 1 & : \quad \langle S(I, t_1), \ldots, S(I, t_n) \rangle \in p^J, \\ 0 & : \quad \text{otherwise.} \end{cases} \tag{120}$$

- **Exercise: Using the above signature Σ_a, structure S_a, and variable assignment I_a, what is the evaluation of Formula 93:**

$$S(I, (1 + x) \cdot y = 0)? \tag{121}$$

- Given a signature Σ, a structure $S = \langle U, J \rangle$, and a variable assignment I, for every formula $f \in F_\Sigma$, its value can be derived inductively as

  1. $S(I, 0) = 0; S(I, 1) = 1$
  2. $S(I, \neg f) = I_\neg(S(I, f))$
  3. $S(I, f \vee g) = I_\vee(S(I, f), S(I, g))$
  4. $S(I, f \wedge g) = I_\wedge(S(I, f), S(I, g))$
  5. $S(I, f \rightarrow g) = I_\rightarrow(S(I, f), S(I, g))$
  6. $S(I, f \leftrightarrow g) = I_\leftrightarrow(S(I, f), S(I, g))$
  7. $S(I, \forall x(f) = \begin{cases} 1 & : \quad \forall c(c \in U \rightarrow S(I[x/c], f) = 1) \\ 0 & : \quad \text{otherwise} \end{cases}$
  8. $S(I, \exists x(f) = \begin{cases} 1 & : \quad \exists c(c \in U \wedge S(I[x/c], f) = 1) \\ 0 & : \quad \text{otherwise} \end{cases}$

- **Using the above signature Σ_a, structure S_a, and variable assignment I_a, determine the evaluation of the following formulas:**
  1. $\exists x((1 + x) \cdot y = 0)$,
  2. $\forall x((1 + x) \cdot y = 0)$,
  3. $\forall x, y((1 + x) \cdot y = 0)$,
  4. $\forall x \exists y((1 + x) \cdot y = 0)$,
  5. $\forall y \exists x((1 + x) \cdot y = 0 \rightarrow \forall z(x \cdot z = 0))$.

- If f is a formula resulting in

$$S(I, f) = 1 \tag{122}$$

for every possible variable assignment I, then f is universally valid.

- Being the equivalent to a tautology in propositional logic, universal validity of a formula f is written as

$$\models f. \tag{123}$$

- If $\text{free}(f) = \{\}$, then $S(I, f)$ does not depend on I. In this case, f is called a closed formula.

- For closed formulas, we use an abbreviated terminology:

$$S(f) := S(I, f) \quad \text{if} \quad \text{free}(f) = \{\}. \tag{124}$$

- Also, in the case that

$$S(f) = 1, \tag{125}$$

we say that the structure S is a model for f written as

$$S \models f. \tag{126}$$

- Also the notions of equivalence and (un)satisfiability are inherited from propositional logic.
- Two formulas f and g are equivalent iff

$$\models f \leftrightarrow g. \tag{127}$$

- A set of formulas $M \subseteq F_{\Sigma}$ is satisfiable, if there is a variable assignment I such that

$$\forall m(m \in M \rightarrow S(I, m) = 1). \tag{128}$$

- Otherwise, M is called unsatisfiable written as

$$M \models 0. \tag{129}$$

- Our next goal is to define an algorithm that checks whether

$$M \models 0. \tag{130}$$

- In general, this question is undecidable (as shown later).
- We will, however, be able to define a calculus $\vdash$ for which we have

$$M \vdash 0 \leftrightarrow M \models 0. \tag{131}$$

- This calculus will be based on a semi-decision algorithm.
- I.e., if indeed $M \models 0$, the algorithm will eventually discover this fact.
- If, however, M is satisfiable, the algorithm may run eternally.
- Motivation of semi-decidability: Goldbach conjecture.

- **In order to define the calculus $\vdash$, we will transform formulas into first-order clauses.**
- **This transformation will make use of the following equivalences (in addition to the ones we know from propositional logic):**

21. $\models \neg\forall x(f) \leftrightarrow \exists x(\neg f)$

22. $\models \neg\exists x(f) \leftrightarrow \forall x(\neg f)$

23. $\models \forall x(f) \wedge \forall x(g) \leftrightarrow \forall x(f \wedge g)$

24. $\models \exists x(f) \vee \exists x(g) \leftrightarrow \exists x(f \vee g)$

25. $\models \forall x, y(f) \leftrightarrow \forall y, x(f)$

26. $\models \exists x, y(f) \leftrightarrow \exists y, x(f)$

27. If $x \in V \wedge x \notin \mathbf{free}(f)$ **then**

a) $\models \forall x(f) \leftrightarrow f$

b) $\models \exists x(f) \leftrightarrow f$

28. If $x \in V \wedge x \notin \text{free}(g) \cup \text{bound}(g)$ then
    a) $\models \forall x(f) \vee g \leftrightarrow \forall x(f \vee g)$
    b) $\models \exists x(f) \wedge g \leftrightarrow \exists x(f \wedge g)$
    c) $\models g \vee \forall x(f) \leftrightarrow \forall x(g \vee f)$
    d) $\models g \wedge \exists x(f) \leftrightarrow \exists x(g \wedge f)$

- In certain situations, it is necessary to rename bound variables.
- If $f \in F_\Sigma$ and $x, y \in V$ then $f[x/y]$ is the formula which we obtain by replacing every occurrence of x by z.
- For example

$$(\forall u \exists v(P(u, v)))[u/z] = \forall z \exists v(P(z, v)). \tag{132}$$

- This leads us to our last equivalence:

29. If $x \in \text{bound}(f) \wedge y \notin \text{free}(f) \cup \text{bound}(f)$ then
    $\models f \leftrightarrow f[x/y]$

- **The above equivalences can be used to rewrite an arbitrary formula such that all quantifiers appear at the beginning of the formula.**
- **This representation is called prenex normal form.**
- **Example 1:**

$$\begin{aligned}
& \forall x(p(x)) \longrightarrow \exists x(p(x)) \quad \textbf{(closed)} \\
\overset{\mathbf{12}}{\leftrightarrow} \quad & \neg\forall x(p(x)) \vee \exists x(p(x)) \\
\overset{\mathbf{21}}{\leftrightarrow} \quad & \exists x(\neg p(x)) \vee \exists x(p(x)) \\
\overset{\mathbf{24}}{\leftrightarrow} \quad & \exists x(\neg p(x) \vee p(x)) \\
\overset{\mathbf{2}}{\leftrightarrow} \quad & \exists x(1) \\
\overset{\mathbf{27}b}{\leftrightarrow} \quad & 1 \quad \textbf{(universally valid)}
\end{aligned} \tag{133}$$

- **Example 2:**

$$\begin{aligned}
& \exists x(p(x)) \longrightarrow \forall x(p(x)) \quad \textbf{(closed)} \\
\overset{\mathbf{12}}{\leftrightarrow} \quad & \neg\exists x(p(x)) \vee \forall x(p(x)) \\
\overset{\mathbf{21}}{\leftrightarrow} \quad & \forall x(\neg p(x)) \vee \forall x(p(x)) \\
\overset{\mathbf{29}}{\leftrightarrow} \quad & \forall x(\neg p(x)) \vee \forall y(p(y)) \\
\overset{\mathbf{28}a}{\leftrightarrow} \quad & \forall x(\neg p(x) \vee \forall y(p(y))) \\
\overset{\mathbf{28}c}{\leftrightarrow} \quad & \forall x, y(\neg p(x) \vee p(y))
\end{aligned} \tag{134}$$

- **This formula says, if there is at least one x turning $p(x)$ true then $p(x)$ is always true.**
- **In turn, if there is *no* x turning $p(x)$ true then $p(x)$ is, logically, always false.**
- **Consequently, $p(x)$ is independently of x either true or false (a propositional variable).**

- The next normalization step requires a wider notion of equivalence.
- We will refer to it as equisatisfiability.
- Consider the example formulas

$$f_1 = \forall x \exists y (p(x, y)) \tag{135}$$

and

$$f_2 = \forall x (p(x, s(x))). \tag{136}$$

- f_1 and f_2 are not equivalent.
- E.g., $p(x, y) := x > y$.
- They do not even use the same signature (f_2 uses the function symbol s not existing in f_1).

- However, we can relate f_1 and f_2 as follows:
- If a structure S_1 is a model for f_1, i.e.,

$$S_1(f_1) = 1 \tag{137}$$

  then it can be extended to a structure S_2 being a model for f_2,

$$S_2(f_2) = 1. \tag{138}$$

- This can be done by defining an interpretation s^J such that

$$p(x, s(x)) \tag{139}$$

  is true for all possible x of the universe.
- In our above example, this could for instance be $s(x) := x - 1$.
- Formally, two formulas f_1 and f_2 are equisatisfiable if

$$\models S_1(f_1) \leftrightarrow S_2(f_2) \tag{140}$$

  also written as

$$f_1 \approx f_2. \tag{141}$$

- We are given a signature $\Sigma = \langle V, F, P, \text{arity} \rangle$ and a closed formula f of the form

$$f = \forall x_1, \ldots, x_n \exists y (g(x_1, \ldots, x_n, y)). \tag{142}$$

- We choose a new n-ary function symbol $s \notin F$ and extend the signature:

$$\Sigma' = \langle V, F \cup \{s\}, P, \text{arity} \cup \{\langle s, n \rangle\} \rangle. \tag{143}$$

- Now, we define the formula f' as follows:

$$f' := \text{skolem}(f) := \forall x_1, \ldots, x_n (g(x_1, \ldots, x_n, s(x_1, \ldots, x_n))). \tag{144}$$

- Here, we have dropped the existential quantifier.
- Every occurrence of the variable y has been replaced by the term $s(x_1, \ldots, x_n)$.
- For the resulting formula f', we have

$$f' \approx f. \tag{145}$$

$$
\begin{aligned}
f_1 &= \forall x, y \exists z (p(x, y, z)), \\
\textbf{skolem}(f_1) &= \forall x, y (p(x, y, s_z^1(x, y)));
\end{aligned}
$$

$$
\begin{aligned}
f_2 &= \forall x \exists y \forall z (p(x, y, z)), \\
\textbf{skolem}(f_2) &= \forall x, z (p(x, s_y^2(x), z));
\end{aligned}
$$

$$
\begin{aligned}
f_3 &= \exists x \forall y \exists z (p(x, y, z)), \\
\textbf{skolem}(f_3) &= \forall y \exists z (p(s_x^3, y, z)), \\
\textbf{skolem}(\textbf{skolem}(f_3)) &= \forall y (p(s_x^3, y, s_z^3(y))) \qquad (146)
\end{aligned}
$$

- **This algorithm turns an first-order formula f into clausal normal form:**
  1. **Convert f into prenex normal form.**
  2. **Eliminate all existential quantifiers by skolemization. The result is a formula of the form**
  $$f' = \forall x_1, \ldots, x_n(g) \tag{147}$$
  where g contains no quantifiers.
  3. **Since g contains only atomic formulas connected by logical connectives, it can be turned into conjunctive normal form which produces**
  $$f'' = \forall x_1, \ldots, x_n(d_1 \wedge \cdots \wedge d_m) \tag{148}$$
  where d_i are disjunctions of literals (in first-order logic, literals are atomic formulas or their negations).

4. **Applying Equivalence 23, the universal quantifiers can be distributed onto all d_is resulting in the clausal normal form**

$$f''' = c_1 \wedge \cdots \wedge c_m \quad \text{with} \quad c_i = \forall x_1, \ldots, x_n(d_i), \tag{149}$$

a conjunction of the first-order clauses $c_1, \ldots, c_m$.

5. **This notation can further be simplified by agreeing that all free variables are implicitely universally quantified:**

$$\begin{aligned} f^{(4)} &= d_1 \wedge \cdots \wedge d_m \\ &\Leftrightarrow \{d_1, \ldots, d_m\}. \end{aligned} \tag{150}$$

- **Turn the following formulas into clausal normal form:**
  1. $f_1 = \forall y \exists x((1 + x) \cdot y = 0 \rightarrow \forall z(x \cdot z = 0))$,
  2. $f_2 = \exists x, y(p(x) \wedge p(y) \leftrightarrow \exists z(p(z)))$,
  3. $\neg f_2$.
- **For Task 1, given the structure S_a, find a functioning interpretation of the skolem function replacing the variable x.**

- **Our goal is to find out whether a formula f is universally valid:**

$$\models f. \tag{151}$$

- **This is the same as to tell whether f's negation is unsatisfiable:**

$$\{\neg f\} \models 0. \tag{152}$$

- **This can be done by**
  1. **transforming $\neg f$ into clausal normal form**

$$c_1 \wedge \cdots \wedge c_m \approx \neg f, \tag{153}$$

  2. **trying to prove inconsistency from the set of clauses:**

$$\{c_1, \ldots, c_m\} \vdash \{\}. \tag{154}$$

- We want to find out whether the formula

$$f := \exists x \forall y(p(x,y)) \rightarrow \forall y \exists x(p(x,y)) \tag{155}$$

  is universally valid.

- This can be done by using the provability algorithm:

  1. Transforming $\neg f$ into clausal normal form:

     1.1 Conversion into prenex normal form:

$$\begin{aligned}
& \neg(\exists x \forall y(p(x,y)) \rightarrow \forall y \exists x(p(x,y))) \\
\overset{\mathbf{12}}{\leftrightarrow} \quad & \neg(\neg \exists x \forall y(p(x,y)) \vee \forall y \exists x(p(x,y))) \\
\overset{\mathbf{11}}{\leftrightarrow} \quad & \exists x \forall y(p(x,y)) \wedge \neg \forall y \exists x(p(x,y)) \\
\overset{\mathbf{21}}{\leftrightarrow} \quad & \exists x \forall y(p(x,y)) \wedge \exists y \neg \exists x(p(x,y)) \\
\overset{\mathbf{22}}{\leftrightarrow} \quad & \exists x \forall y(p(x,y)) \wedge \exists y \forall x(\neg p(x,y)) \\
\overset{\mathbf{29}}{\leftrightarrow} \quad & \exists x \forall y(p(x,y)) \wedge \exists v \forall u(\neg p(u,v)) \\
\overset{\mathbf{28}b}{\leftrightarrow} \quad & \exists x(\forall y(p(x,y)) \wedge \exists v \forall u(\neg p(u,v)))
\end{aligned}$$

$$
\begin{aligned}
&\overset{28d}{\leftrightarrow} && \exists x, v(\forall y(p(x,y)) \wedge \forall u(\neg p(u,v))) \\
&\overset{27a}{\leftrightarrow} && \exists x, v(\forall y(p(x,y)) \wedge \forall y, u(\neg p(u,v))) \\
&\overset{23}{\leftrightarrow} && \exists x, v \forall y(p(x,y) \wedge \forall u(\neg p(u,v))) \\
&\overset{27a}{\leftrightarrow} && \exists x, v \forall y(\forall u(p(x,y)) \wedge \forall u(\neg p(u,v))) \\
&\overset{23}{\leftrightarrow} && \exists x, v \forall y, u(p(x,y) \wedge \neg p(u,v)) \\
&=: && f' \qquad (156)
\end{aligned}
$$

1.2 Skolemization:

- **f' is of the form**

$$f' = \exists x(g(x)) \quad \textbf{with} \quad g = \exists v \forall y, u(p(x,y) \wedge \neg p(u,v)). \quad (157)$$

- **Here, g is 1-ary since there are no universal quantifiers in front of the existential quantifier.**

– **Consequently, skolemization introduces the 0-ary function s_x replacing x:**

$$\begin{aligned} f'' &= \text{skolem}(f') \\ &= g(s_x()) \\ &= g(s_x) \\ &= \exists v \forall y, u(p(s_x, y) \wedge \neg p(u, v)). \end{aligned} \tag{158}$$

– **In a second step, the existential quantifier in front of v is replaced by skolemization:**

$$\begin{aligned} f''' &= \text{skolem}(f'') \\ &= \forall y, u(\underbrace{p(s_x, y) \wedge \neg p(u, s_v)}_{h}). \end{aligned} \tag{159}$$

1.3-5 Clausal normal form:

Since h is already in conjunctive normal form, we have

$$f^{(4)} = p(s_x, y) \wedge \neg p(u, s_v). \tag{160}$$

2. **Proving inconsistency:**
   - **We are given clausal normal form**
     $$M := \{\{p(s_x, y)\}, \{\neg p(u, s_v)\}\}. \tag{161}$$
   - **Since all variables in first-order clauses are implicitely universally bound, we can substitute y by s_v and u by s_x resulting in**
     $$M \vdash \{\{p(s_x, s_v)\}, \{\neg p(s_x, s_v)\}\}. \tag{162}$$
   - **Application of the cut rule gives**
     $$M \vdash \{\}. \tag{163}$$
   - **Consequently, we have shown that**
     $$\models f. \tag{164}$$

- In the provability example, we have "guessed" the terms s_x and s_v to be able to apply the cut rule.
- We now want to study a method to do systematically calculate terms which the cut rule can be applied to.
- In order to do so, for a given signature Σ, we define a substitution

$$\sigma = \{\langle x_1, t_1\rangle, \ldots, \langle x_n, t_n\rangle\} =: [x_1 \mapsto t_1, \ldots, x_n \mapsto t_n] \tag{165}$$

with

$$x_i \in V, t_i \in T_\Sigma \text{ for } i \in \{1, \ldots, n\} \text{ and } x_i \neq x_j \text{ for } i \neq j. \tag{166}$$

- The domain of the substitution is defined as

$$\mathbf{dom}(\sigma) = \{x_1, \ldots, x_n\}. \tag{167}$$

- Given a term t and a substitution σ, we define the application of σ to t (written $t\sigma$) inductively as
  1. $x_i\sigma := t_i$ for $x_i \in \mathbf{dom}(\sigma)$,
  2. $y\sigma := y$ for $y \in V \wedge y \notin \mathbf{dom}(\sigma)$,
  3. $f(s_1, \ldots, s_m)\sigma := f(s_1\sigma, \ldots, s_m\sigma)$ otherwise.
- Example: We are given the substitution

$$\sigma = [x \mapsto c, y \mapsto f(d)]. \tag{168}$$

- Then, we have the following applications:
  1. $z\sigma = z$,
  2. $f(y)\sigma = f(f(d))$,
  3. $h(x, g(y))\sigma = h(c, g(f(d)))$,
  4. $\{p(y), q(d, h(z, x))\}\sigma = \{p(f(d)), q(d, h(z, c))\}$.

- A syntactical equation (SE) is a construct of the form $s \doteq t$ where s and t are either both terms or both atomic formulas.
- A system of syntactical equations (SSE) is a set of SEs:

$$E := \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}. \tag{169}$$

- Given an SSE E and a substitution σ, we define the application of σ to E as

$$E\sigma := \{s_1\sigma \doteq t_1\sigma, \ldots, s_n\sigma \doteq t_n\sigma\}. \tag{170}$$

- A substitution σ solves an SE $s \doteq t$ iff we have $s\sigma = t\sigma$.
- If E is an SSE then the substitution σ is called a unifier iff it solves every SE in E.
- Example: Show that

$$\sigma = [x_1 \mapsto x_2, x_3 \mapsto f(x_4)] \tag{171}$$

is a unifier of the SSE

$$E = \{p(f(x_4)) \doteq p(x_3), q(x_1, x_2) \doteq q(x_2, x_1)\}. \tag{172}$$

- We now want to study an algorithm calculating a unifier σ for a given SSE E.
- A number of reduction rules will transform a tuple consisting of an SSE and a substitution into another such tuple:

$$\langle E_1, \sigma_1 \rangle \rightsquigarrow \langle E_2, \sigma_2 \rangle. \tag{173}$$

- Here come the reduction rules:
  1. If $y \in V \wedge t \in T_\Sigma \wedge y \notin \mathrm{var}(t)$ then

$$\langle E \cup \{y \doteq t\}, \sigma \rangle \rightsquigarrow \langle E[y \mapsto t], \sigma \cup \{\langle y, t \rangle\} \rangle. \tag{174}$$

     - If an SSE contains an SE of the form $y \doteq t$ where y is a variable not contained in term t then the SE is solved by the substitution $[y \mapsto t]$.
     - Consequently, the SE can be dropped in favor of applying the substitution to all other SEs in the SSE and adding it to σ.
     - E.g., $x \doteq s(y)$.

2. If $y \in V \wedge t \in T_\Sigma \wedge y \in \text{var}(t) \wedge y \neq t$ then

$$\langle E \cup \{y \doteq t\}, \sigma \rangle \rightsquigarrow \Omega. \tag{175}$$

   - If an SSE contains an SE of the form $y \doteq t$ where y is a variable contained in term t but not t itself then the SE is not solvable.
   - That is because every possible term we can substitute for y will also affect t and render it different.
   - E.g., $x \doteq s(x)$.

3. If $y \in V \wedge t \in T_\Sigma \wedge t \notin V$ then

$$\langle E \cup \{t \doteq y\}, \sigma \rangle \rightsquigarrow \langle E \cup \{y \doteq t\}, \sigma \rangle. \tag{176}$$

   - Moves the variable to the front.
   - E.g., $s(y) \doteq x$.

4. $$\langle E \cup \{t \doteq t\}, \sigma \rangle \rightsquigarrow \langle E, \sigma \rangle. \tag{177}$$

   - Drops trivial SEs.
   - E.g., $s(y) \doteq s(y)$.

5. If $f \in F \cup P \wedge n = \text{arity}(f) \wedge s_1, \ldots, s_n, t_1, \ldots, t_n \in T_\Sigma$ then

$$\langle E \cup \{f(s_1, \ldots, s_n) \doteq f(t_1, \ldots, t_n)\}, \sigma\rangle \rightsquigarrow \langle E \cup \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}, \sigma\rangle. \quad (178)$$

   - From the arguments of an n-ary function (predicate) present on both sides of an SE, n individual SEs are derived.
   - E.g., $s(s(x), y) \doteq s(y, z)$.

6. If $f, g \in F \cup P \wedge f \neq g \wedge m = \text{arity}(f) \wedge n = \text{arity}(g)$ $\wedge s_1, \ldots, s_m, t_1, \ldots, t_n \in T_\Sigma$ then

$$\langle E \cup \{f(s_1, \ldots, s_m) \doteq g(t_1, \ldots, t_n)\}, \sigma\rangle \rightsquigarrow \Omega. \quad (179)$$

   - Different functions (predicates) cannot be unified.
   - E.g., $f(x) \doteq g(x)$.

- Given an SSE E, to determine a unifier, we start off with an empty substitution:

$$\langle E, [] \rangle. \tag{180}$$

- Now, we apply the reduction rules until we have either of the following cases:

  a) We can derive Ω meaning that E is not solvable.

  b) We can show that

$$\langle E, [] \rangle \rightsquigarrow \langle \{\}, \sigma \rangle. \tag{181}$$

  Here, σ is the unifier of E also written as

$$\sigma = \mathbf{mgu}(E) \tag{182}$$

  (the most general unifier) with the special case

$$\mathbf{mgu}(s, t) := \mathbf{mgu}(\{s \doteq t\}). \tag{183}$$

- We are given an SSE with a single SE:

$$E = \{p(x_1, f(x_4)) \doteq p(x_2, x_3)\} \quad \textbf{with} \tag{184}$$
$$V = \{x_1, \ldots\}, F = \{f\}, P = \{\}.$$

- Now, we attempt to determine a unifier:

$$\begin{aligned}
& \langle \{p(x_1, f(x_4)) \doteq p(x_2, x_3)\}, [] \rangle \\
\overset{5}{\rightsquigarrow} \quad & \langle \{x_1 \doteq x_2, f(x_4) \doteq x_3\}, [] \rangle \\
\overset{1}{\rightsquigarrow} \quad & \langle \{f(x_4) \doteq x_3\}, [x_1 \mapsto x_2] \rangle \\
\overset{3}{\rightsquigarrow} \quad & \langle \{x_3 \doteq f(x_4)\}, [x_1 \mapsto x_2] \rangle \\
\overset{1}{\rightsquigarrow} \quad & \langle \{\}, [x_1 \mapsto x_2, x_3 \mapsto f(x_4)] \rangle
\end{aligned} \tag{185}$$

- That is, we have

$$\begin{aligned}
\textbf{mgu}(E) &= \textbf{mgu}(p(x_1, f(x_4)), p(x_2, x_3)) \\
&= [x_1 \mapsto x_2, x_3 \mapsto f(x_4)].
\end{aligned} \tag{186}$$

- **Determine, if possible, a unifier of the following SSEs:**

  a) $E_1 = \{p(d, x_4) \doteq p(x_2, h(c, d)), q(h(d, x_1)) \doteq q(x_4)\}$,

  b) $E_2 = \{p(h(x_1, c)) \doteq p(x_2), q(x_2, d) \doteq q(h(d, c), x_4)\}$,

  c) $E_3 = \{p(h(x_2, d)) \doteq p(h(x_1, d)), q(x_1, c) \doteq q(h(x_2, d), c)\}$.

  In these exercises, assume

  $$V = \{x_1, \ldots\}, F = \{c, d, h\}, P = \{p, q\}.$$

- The first-order calculus makes use of two inference rules we will define next.
- The first-order resolution rule expects
  1. two first-order clauses c_1 and c_2 and
  2. two atomic formulas $p(s_1, \ldots, s_n)$ and $p(t_1, \ldots, t_n)$ whose syntactical equation is solvable, i.e., we can determine the unifier

$$\mu = \text{mgu}(p(s_1, \ldots, s_n), p(t_1, \ldots, t_n)). \tag{187}$$

- Then, this is the definition of the resolution rule:

$$\frac{c_1 \cup \{p(s_1, \ldots, s_n)\}, \quad c_2 \cup \{\neg p(t_1, \ldots, t_n)\}}{\therefore c_1\mu \cup c_2\mu}$$

- The resolution rule makes use of the cut rule and the substitution rule

$$\frac{c}{\therefore c\sigma}$$

where c is a first-order clause and σ is a substitution.

- When applying the resultion rule, it is sometimes necessary to rename variables.
- In the following first-order clause set

$$M = \{\{p(x)\}, \{\neg p(f(x))\}\}, \tag{188}$$

unification will result in Ω since both involved clauses use the same variable x.

- Substituting x by y in one of the clauses does not change semantics

$$M' = \{\{p(x)\}, \{\neg p(f(y))\}\} \tag{189}$$

simply because

$$\forall x(\neg p(f(x))) \overset{29}{\leftrightarrow} \forall y(\neg p(f(y))). \tag{190}$$

- Now, we are able to determine the unifier

$$\mu = \mathbf{mgu}(p(x), p(f(y))) = [x \mapsto f(y)] \tag{191}$$

- In conclusion, we can prove contradiction by applying the resultion rule:

$$\{\{p(x)\}, \{\neg p(f(y))\}\} \overset{\text{res. } [x \mapsto f(y)]}{\vdash} \{\}. \tag{192}$$

- **Consider the following example**

$$M = \{\{p(x), p(y)\}, \{\neg p(x), \neg p(y)\}\} \tag{193}$$

- **Application of the resolution rule using $c_1 = p(x)$ and $c_2 = \neg p(x)$ and the unifier $\mu = \text{mgu}(p(y), p(y)) = []$ produces**

$$\begin{aligned} & \{\{p(x), p(y)\}, \{\neg p(x), \neg p(y)\}\} \\ \overset{\text{res. } []}{\vdash} \quad & \{\{p(y), \neg p(y)\}\} \\ \vdash \quad & \mathbf{1}. \end{aligned} \tag{194}$$

- **This result does not help since we are attempting to prove contradiction.**
- **Instead, one could factorize M as follows:**

$$
\begin{aligned}
& \{\{p(x), p(y)\}, \{\neg p(x), \neg p(y)\}\} \\
\leftrightarrow \quad & \{\{p(x), p(y)\}, \{\neg p(v), \neg p(w)\}\} \\
\leftrightarrow \quad & \forall x, y, v, w((p(x) \vee p(y)) \wedge (\neg p(v) \vee \neg p(w))) \\
\leftrightarrow \quad & \forall x, y, v, w(p(x) \wedge \neg p(v) \vee p(x) \wedge \neg p(w) \vee \\
& p(y) \wedge \neg p(v) \vee p(y) \wedge \neg p(w)) \\
\leftrightarrow \quad & \forall x, v(p(x) \wedge \neg p(v)) \vee \forall x, w(p(x) \wedge \neg p(w)) \vee \\
& \forall y, v(p(y) \wedge \neg p(v)) \vee \forall y, w(p(y) \wedge \neg p(w))) \\
\leftrightarrow \quad & \forall x(p(x)) \wedge \forall x(\neg p(x)) \vee \forall x(p(x)) \wedge \forall x(\neg p(x)) \vee \\
& \forall y(p(y)) \wedge \forall y(\neg p(y)) \vee \forall y(p(y)) \wedge \forall y(\neg p(y))) \\
\leftrightarrow \quad & \forall x(p(x) \wedge \neg p(x)) \vee \forall y(p(y) \wedge \neg p(x)) \\
\leftrightarrow \quad & \mathbf{0}. \qquad (195)
\end{aligned}
$$

- **Another possibility is, as in propositional logic, to apply case distinction to account for multiple occurrences of negated literals:**
- **Pick $\{p(x)\}$:**

$$\begin{aligned} & \{\{p(x)\}, \{p(x), p(y)\}, \{\neg p(x), \neg p(y)\}\} \\ \overset{\text{res. } []}{\vdash} \quad & \{\{p(x)\}, \{p(x), p(y)\}, \{\neg p(y)\}\} \\ \overset{\text{res. } [x \mapsto y]}{\vdash} \quad & \{\{\}, \{p(x), p(y)\}\} \\ \vdash \quad & \mathbf{0}. \end{aligned} \tag{196}$$

- **Pick** $\neg\{p(x)\} \leftrightarrow \neg\forall x(p(x)) \leftrightarrow \exists x(\neg p(x)) \approx \neg p(s) \leftrightarrow \{\neg p(s)\}$:

$$\begin{aligned} & \{\{\neg p(s)\}, \{p(x), p(y)\}, \{\neg p(x), \neg p(y)\}\} \\ \overset{\text{res. } [x \mapsto s]}{\vdash} \quad & \{\{\neg p(s)\}, \{p(y)\}, \{\neg p(x), \neg p(y)\}\} \\ \overset{\text{res. } [y \mapsto s]}{\vdash} \quad & \{\{\}, \{\neg p(x), \neg p(y)\}\} \\ \vdash \quad & \mathbf{0}. \end{aligned} \tag{197}$$

- **So, in conclusion, we have**

$$\{\{p(x), p(y)\}, \{\neg p(x), \neg p(y)\}\} \vdash \mathbf{0}. \tag{198}$$

- In general, we account for multiple occurrences of unifiable predicates in a clause by means of factorization rules.
- Given
  1. the first-order clause c and
  2. two atomic formulas $p(s_1, \ldots, s_n)$ and $p(t_1, \ldots, t_n)$ whose syntactical equation is solvable, i.e., we can determine the unifier

$$\mu = \text{mgu}(p(s_1, \ldots, s_n), p(t_1, \ldots, t_n)). \tag{199}$$

- Then, these are the definitions of the factorization rules:

$$\frac{c \cup \{p(s_1, \ldots, s_n), p(t_1, \ldots, t_n)\}}{\therefore c\mu \cup \{p(s_1, \ldots, s_n)\mu\}}$$

and

$$\frac{c \cup \{\neg p(s_1, \ldots, s_n), \neg p(t_1, \ldots, t_n)\}}{\therefore c\mu \cup \{\neg p(s_1, \ldots, s_n)\mu\}}$$

- **Returning to the above example, application of the factorization rules produces**

$$
\begin{array}{ll}
 & \{\{p(x), p(y)\}, \{\neg p(x), \neg p(y)\}\} \\
\overset{\text{fact.}}{\vdash} & \{\{p(y)\}, \{\neg p(x), \neg p(y)\}\} \\
\overset{\text{fact.}}{\vdash} & \{\{p(y)\}, \{\neg p(y)\}\} \\
\overset{\text{res. }[]}{\vdash} & \mathbf{0}.
\end{array}
$$

(200)

- **Can you derive contradiction from the following clause sets?**
  - **a)** $M_1 = \{\{\neg p(y, y)\}, \{p(f(x), y), p(y, g(v))\}\}$
  - **b)** $M_2 = \{\{p(x, y), p(g(u), v), p(z, v)\}, \{\neg p(x, y), \neg p(g(u), v)\}\}$
  - **c)** $M_3 = \{\{p(x, f(z)), p(g(u), v), p(g(u), f(z))\}\}$
  - **d)** $M_4 = \{\{p(f(g(u)), v), \neg p(f(z), v)\}, \{\neg p(x, y), \neg p(g(u), v)\}\}$

- Similarly to the function cut introduced for the definition of the DP algorithm in propositional logic, we introduce the function res for first-order logic.
- The function res applies the resolution rule to a set of clauses C_0 with respect to a literal (unit clause).
- As opposed to cut, it does not drop any clauses from C_0 but simply adds conclusions from the resolution rule to the clause set.
- The reason that clauses are not dropped is that different atomic formulas may be based on the same predicate but different arguments which may produce a different unifier and, hence, a different result clause.
- If the original clause would have been dropped, this different clause would have been missed.

- **Example 1:**

$$\mathsf{res}(\{\{p(x), \neg q(f(t))\}, \{q(x)\}\}, q(x)) = \{\{p(x), \neg q(f(t))\}, \{p(f(t))\}, \{q(x)\}\} \quad (201)$$

- **Example 2:**

$$\mathsf{res}(\{\{p(y), \neg q(x)\}, \{p(f(y)), \neg q(x)\}, \{\neg p(x)\}\}, \neg p(x)) = \{\{p(y), \neg q(x)\}, \{\neg q(x)\}, \{p(f(y)), \neg q(x)\}, \{\neg q(f(y))\}, \{\neg p(x)\}\} \quad (202)$$

- **Example 3 (infinite loop):**

$$\begin{aligned} C_0 &:= \{\{\neg p(x), p(f(x))\}, \{p(x)\}\} \\ C_1 &:= \mathsf{res}(C_0, p(x)) \\ &= \{\{\neg p(x), p(f(x))\}, \{p(f(x))\}, \{p(x)\}\} \\ &= \{\{\neg p(x), p(f(x))\}, \{p(f(y))\}, \{p(x)\}\} \\ C_2 &:= \mathsf{res}(C_1, p(f(y))) \\ &= \{\{\neg p(x), p(f(x))\}, \{p(f(f(y)))\}, \{p(f(y))\}, \{p(x)\}\} \\ C_3 &:= \mathsf{res}(C_2, p(f(f(y)))) \quad \ldots \end{aligned} \quad (203)$$

- **We have now learned all the pieces necessary to apply first-order calculus in a systematic way.**
- **Let us reiterate on the involved steps by means of an example.**
- **This is our knowledge base:**
  - **a)** *Every dragon is happy if all his children know how to fly.*
  - **b)** *Every red dragon knows how to fly.*
  - **c)** *Children of red dragons are red.*
- **Now, let us try to prove whether or not**
  - **d)** *All red dragons are happy.*

- To get started, we define a signature $\Sigma_a = \langle V, F, P, \text{arity} \rangle$ with

  $V = \{x, y\}$

  $F = \{\}$

  $P = \{r, f, h, c\}$

  $\text{arity} = \{\langle r, 1\rangle \langle f, 1\rangle \langle h, 1\rangle \langle c, 2\rangle\}$

- We assume that the universe contains all dragons.

- The predicates have the following interpretations:
  - $r(x)$ is 1 iff x is red.
  - $f(x)$ is 1 iff x knows how to fly.
  - $h(x)$ is 1 iff x is happy.
  - $c(x, y)$ is 1 iff x is y's child.

- **Formalizing knowledge base and claim:**

  a) $f_1 := \forall x(\forall y(c(y,x) \rightarrow f(y)) \rightarrow h(x))$

  b) $f_2 := \forall x(r(x) \rightarrow f(x))$

  c) $f_3 := \forall x(\exists y(c(x,y) \wedge r(y)) \rightarrow r(x))$

  d) $f_4 := \forall x(r(x) \rightarrow h(x))$

- **Now, to see whether the claim d) can be derived from the knowledge base a) to c) is to prove that**

$$f := f_1 \wedge f_2 \wedge f_3 \rightarrow f_4 \tag{204}$$

  is universally valid.

- **This is the same as to prove that $\neg f$ is a contradiction:**

$$\begin{aligned}
\neg f &\leftrightarrow \neg(f_1 \wedge f_2 \wedge f_3 \longrightarrow f_4) \\
&\leftrightarrow \neg(\neg(f_1 \wedge f_2 \wedge f_3) \vee f_4) \\
&\leftrightarrow \neg\neg(f_1 \wedge f_2 \wedge f_3) \vee \neg f_4) \\
&\leftrightarrow \neg\neg(f_1 \wedge f_2 \wedge f_3) \wedge \neg f_4 \\
&\leftrightarrow f_1 \wedge f_2 \wedge f_3 \wedge \neg f_4 \qquad (205)
\end{aligned}$$

- **Next, we need to turn $\neg f$ into a set of clauses.**
- **This can be done by turning f_1, f_2, f_3, and $\neg f_4$ individually into clausal normal form.**

$$
\begin{aligned}
f_1 &\leftrightarrow \forall x(\forall y(c(y,x) \rightarrow f(y)) \rightarrow h(x)) \\
&\leftrightarrow \forall x(\forall y(\neg c(y,x) \vee f(y)) \rightarrow h(x)) \\
&\leftrightarrow \forall x(\neg \forall y(\neg c(y,x) \vee f(y)) \vee h(x)) \\
&\leftrightarrow \forall x(\exists y(\neg(\neg c(y,x) \vee f(y))) \vee h(x)) \\
&\leftrightarrow \forall x(\exists y(\neg\neg c(y,x) \wedge \neg f(y)) \vee h(x)) \\
&\leftrightarrow \forall x(\exists y(c(y,x) \wedge \neg f(y)) \vee h(x)) \\
&\leftrightarrow \forall x \exists y(c(y,x) \wedge \neg f(y) \vee h(x)) \\
&\approx \forall x(c(s(x),x) \wedge \neg f(s(x)) \vee h(x)) \\
&\leftrightarrow \forall x((c(s(x),x) \vee h(x)) \wedge (\neg f(s(x)) \vee h(x))) \\
&\leftrightarrow \{\{c(s(x),x), h(x)\}, \{\neg f(s(x)), h(x)\}\} \qquad (206)
\end{aligned}
$$

$$
\begin{aligned}
f_2 &\leftrightarrow \forall x(r(x) \rightarrow f(x)) \\
&\leftrightarrow \forall x(\neg r(x) \vee f(x)) \\
&\leftrightarrow \{\{\neg r(x), f(x)\}\} \qquad (207)
\end{aligned}
$$

$$
\begin{aligned}
f_3 &\leftrightarrow \forall x(\exists y(c(x,y) \wedge r(y)) \rightarrow r(x)) \\
&\leftrightarrow \forall x(\neg\exists y(c(x,y) \wedge r(y)) \vee r(x)) \\
&\leftrightarrow \forall x(\forall y(\neg(c(x,y) \wedge r(y))) \vee r(x)) \\
&\leftrightarrow \forall x(\forall y(\neg c(x,y) \vee \neg r(y)) \vee r(x)) \\
&\leftrightarrow \forall x,y(\neg c(x,y) \vee \neg r(y) \vee r(x)) \\
&\leftrightarrow \{\{\neg c(x,y), \neg r(y), r(x)\}\} \qquad (208)
\end{aligned}
$$

$$
\begin{aligned}
\neg f_4 &\leftrightarrow \neg\forall x(r(x) \rightarrow h(x)) \\
&\leftrightarrow \neg\forall x(\neg r(x) \vee h(x)) \\
&\leftrightarrow \exists x(\neg(\neg r(x) \vee h(x))) \\
&\leftrightarrow \exists x(r(x) \wedge \neg h(x)) \\
&\approx r(t) \wedge \neg h(t) \\
&\leftrightarrow \{\{r(t)\}, \{\neg h(t)\}\} \qquad (209)
\end{aligned}
$$

- **Applying the DP algorithm:**

$$
\begin{aligned}
C_0 &:= \{\{c(s(x),x),h(x)\},\{\neg f(s(x)),h(x)\},\{\neg r(x),f(x)\},\\
&\quad \{\neg c(x,y),\neg r(y),r(x)\},\{r(t)\},\{\neg h(t)\}\}\\
C_1 &:= \mathsf{res}(C_0, r(t))\\
&= \{\{c(s(x),x),h(x)\},\{\neg f(s(x)),h(x)\},\{\neg r(x),f(x)\},\{f(t)\},\\
&\quad \{\neg c(x,y),\neg r(y),r(x)\},\{\neg c(x,t),r(x)\},\{r(t)\},\{\neg h(t)\}\}\\
C_2 &:= \mathsf{res}(C_1, \neg h(t))\\
&= \{\{c(s(x),x),h(x)\},\{c(s(t),t)\},\{\neg f(s(x)),h(x)\},\{\neg f(s(t))\},\\
&\quad \{\neg r(x),f(x)\},\{f(t)\},\{\neg c(x,y),\neg r(y),r(x)\},\{\neg c(x,t),r(x)\},\\
&\quad \{r(t)\},\{\neg h(t)\}\}
\end{aligned}
\tag{210}
$$

$$
\begin{aligned}
C_3 &:= \textbf{res}(C_2, c(s(t), t)) \\
&= \{\{c(s(x), x), h(x)\}, \{c(s(t), t)\}, \{\neg f(s(x)), h(x)\}, \{\neg f(s(t))\}, \\
&\quad \{\neg r(x), f(x)\}, \{f(t)\}, \{\neg c(x, y), \neg r(y), r(x)\}, \\
&\quad \{\neg r(t), r(s(t))\}, \{\neg c(x, t), r(x)\}, \{r(s(t))\}, \{r(t)\}, \{\neg h(t)\}\} \\
C_4 &:= \textbf{res}(C_3, r(s(t))) \\
&= \{\{c(s(x), x), h(x)\}, \{c(s(t), t)\}, \{\neg f(s(x)), h(x)\}, \underline{\{\neg f(s(t))\}}, \\
&\quad \{\neg r(x), f(x)\}, \underline{\{f(s(t))\}}, \{f(t)\}, \{\neg c(x, y), \neg r(y), r(x)\}, \\
&\quad \{\neg c(x, s(t)), r(x)\}, \{\neg r(t), r(s(t))\}, \{\neg c(x, t), r(x)\}, \{r(s(t))\}, \\
&\quad \{r(t)\}, \{\neg h(t)\}\} \quad \textbf{(unsatisfiable)}
\end{aligned} \tag{211}
$$

- **We know that**

  Sets can only be contained in sets not contained in the Suendermann set.

  Now, prove that

  The Suendermann set does not contain itself.

- **Given the three properties of a total order**

  1) $a < b$ **and** $b < c$ **implies** $a < c$ **(transitivity),**

  2) $a < b$**,** $b < a$**, or** $a = b$ **is true (trichotomy), and**

  3) $a < a$ **is not true (anti-reflexivity),**

  prove the reflexivity of the equivalence (i.e. $a = a$**).**

- Prolog (programming in logic) is programming language associated with artificial intelligence as well as computer linguistics.
- In accordance with the architecture of XPSs, the main components of logical programming are
  1. a knowledge base (facts and rules),
  2. an inference engine.
- Advantage of logical programming is that one does not have to develop an algorithm to solve the problem since this job is done by the inference engine.
- Instead, we describe the problem by means of logical formulas.
- The open-source SWI-Prolog is available as part of the major Linux distributions as well as Cygwin (`http://cygwin.com`) or can be obtained from

  `http://www.swi-prolog.org`

- **Facts are atomic formulas with the Prolog syntax**

$$p(t_1, \ldots, t_n) \tag{212}$$

featuring the predicate p and the terms $t_1, \ldots, t_n$.

- **All the variables in facts are universally bound, i.e., Eq. 212 represents the logical formula**

$$\forall x_1, \ldots, x_m(p(t_1, \ldots, t_n)). \tag{213}$$

- **Rules are conditional propositions with the Prolog syntax**

$$A :- B_1, \ldots, B_n. \tag{214}$$

featuring the atomic formulas $A, B_1, \ldots, B_n$.

- **Again, all the variables in rules are universally bound, so, Eq. 214 represents the formula**

$$\forall x_1, \ldots, x_m(B_1 \wedge \ldots \wedge B_n \rightarrow A). \tag{215}$$

- **This generally requires formulas to be given as Horn clauses.**

- The first character of variables is a capital letter or an underscore.
- The first character of predicates or functions is a lower-case letter.
- The predicate `true` represents validity.
- The symbols `+`, `-`, `*`, `/`, `.` are function symbols you can use in infix notation.
- The symbols `<`, `>`, `=`, `=<`, `>=`, `\=`, `==`, `\==` are predicate symbols you can use in infix notation. Note that

  `==` tests for equality,

  `\==` tests for inequality, and

  `=` is the unification operator.
- The symbol `\+` (or, alternatively, `not()`) is the negation operator.
- The symbol `%` is used for comments.
- The symbols `,` and `;` is used for conjunction and disjunction, respectively.

- **The following derivation shows that disjunctions in Prolog rules are effectively no additional feature:**

$$
\begin{aligned}
A :- B_1; \ldots; B_n. \quad &\Leftrightarrow \quad B_1 \vee \ldots \vee B_n \rightarrow A. \\
&\Leftrightarrow \quad \neg(B_1 \vee \ldots \vee B_n) \vee A \\
&\Leftrightarrow \quad \neg B_1 \wedge \ldots \wedge \neg B_n \vee A \\
&\Leftrightarrow \quad (\neg B_1 \vee A) \wedge \ldots \wedge (\neg B_n \vee A) \\
&\Leftrightarrow \quad A :- B_1. \\
&\qquad\quad \ldots \\
&\qquad\quad A :- B_n.
\end{aligned} \tag{216}
$$

- **Let us now consider a realistic example:**
  - **All students are smart.**
  - **Whoever is smart is powerful.**
  - **Whoever is computer scientist and professor is powerful.**
  - **Computer scientists are crazy.**
  - **Alan is a student.**
  - **Brad is a student.**
  - **Colin is a computer scientist.**
  - **Colin is a professor.**

- **This is the respective Prolog code (see `student.pl` in the auxiliary package `kbs_*.zip`):**

```
1  smart(X):-student(X).
2  powerful(X):-smart(X).
3  powerful(X):-cs(X),prof(X).
4  crazy(X):-cs(X).
5  student(alan).
6  student(brad).
7  cs(colin).
8  prof(colin).
```

- **We want to find out whether**

  there is a powerful and crazy individual.

- **The respective logical formula is**

$$\exists x(\texttt{powerful}(x) \wedge \texttt{crazy}(x)). \tag{217}$$

- **In order to find out, we first launch Prolog with the command**

```
pl
```

  and get the command prompt

```
?-
```

- **To load our knowledge base, we type**

```
consult(student).
```

- **Now, we can use the Prolog syntax of Eq. 217 to check the validity of our conjecture:**

```
powerful(X),crazy(X).
```

- **We obtain the response**

  ```
  X = colin
  ```

  telling us that Colin is a powerful and crazy individual.

- **In order to identify other potential candidates, we type**

  ```
  ;
  ```

  resulting in the response

  ```
  No
  ```

  which indicates that there are no more solutions to the problem.

- We are given the Prolog program P consisting of a number of rules of the form

$$R := A :- B_1, \ldots, B_m \tag{218}$$

and a query of the form

$$G = Q_1, \ldots, Q_n. \tag{219}$$

- Here, facts are expanded to rules by

$$A \leftrightarrow A :- \texttt{true}. \tag{220}$$

- The inference algorithm works as follows:
  1. Search (in order of appearance) all the rules A in P, for which there exists a unifier

$$\mu = \begin{cases} [] & \text{if } Q_1 = \texttt{true} \\ \textbf{mgu}(Q_1, A) & \text{otherwise} \end{cases} \tag{221}$$

2. In case there are multiple such rules,
   a) select the first rule (in order of appearance),
   b) set a choice point (CP) to perform a different selection at this point in case it becomes necessary at a later moment.
3. Here, two cases are distinguished:
   a) $m + n = 1$: This means success, and Prolog returns the last non-empty μ.
   b) Otherwise, we recursively continue with the query
   $$G := B_1\mu, \ldots, B_m\mu, Q_2\mu, \ldots, Q_n\mu. \tag{222}$$
   If we do not find a solution, we return to the last choice point reversing the replacements $G := G\mu$ accordingly.

- Negation is implemented in Prolog as negation as failure.
- I.e., if Q_1 in 1 is of the syntax $\texttt{not}(Q_1')$ the algorithm tries to prove Q_1'.
- If it succeeds, we know that Q_1 is false, otherwise, we assume it is true.

ID	CP	G	R	μ
1	1	$\texttt{powerful}(X), \texttt{crazy}(X)$	$\texttt{powerful}(X) :- \texttt{smart}(X)$	[]
2	1	$\texttt{smart}(X), \texttt{crazy}(X)$	$\texttt{smart}(X) :- \texttt{student}(X)$	[]
3	3	$\texttt{student}(X), \texttt{crazy}(X)$	$\texttt{student}(\texttt{alan}) :- \texttt{true}$	$[X \mapsto \texttt{alan}]$
4	3	$\texttt{true}, \texttt{crazy}(\texttt{alan})$		[]
5	3	$\texttt{crazy}(\texttt{alan})$	$\texttt{crazy}(X) :- \texttt{cs}(X)$	$[X \mapsto \texttt{alan}]$
6	3	$\texttt{cs}(\texttt{alan})$	$\texttt{cs}(\texttt{colin}) :- \texttt{true}$	Ω
7	1	$\texttt{student}(X), \texttt{crazy}(X)$	$\texttt{student}(\texttt{brad}) :- \texttt{true}$	$[X \mapsto \texttt{brad}]$
8	3	$\texttt{true}, \texttt{crazy}(\texttt{brad})$		[]
9	1	$\texttt{crazy}(\texttt{brad})$	$\texttt{crazy}(X) :- \texttt{cs}(X)$	$[X \mapsto \texttt{brad}]$
10	1	$\texttt{cs}(\texttt{brad})$	$\texttt{cs}(\texttt{colin}) :- \texttt{true}$	Ω

ID	CP	G	R	μ
11		$\texttt{powerful}(X), \texttt{crazy}(X)$	$\texttt{powerful}(X) :- \texttt{cs}(X), \texttt{prof}(X)$	[]
12		$\texttt{cs}(X), \texttt{prof}(X), \texttt{crazy}(X)$	$\texttt{cs}(\texttt{colin}) :- \texttt{true}$	$[X \mapsto \texttt{colin}]$
13		$\texttt{true}, \texttt{prof}(\texttt{colin}), \texttt{crazy}(\texttt{colin})$		[]
14		$\texttt{prof}(\texttt{colin}), \texttt{crazy}(\texttt{colin})$	$\texttt{prof}(\texttt{colin}) :- \texttt{true}$	[]
15		$\texttt{true}, \texttt{crazy}(\texttt{colin})$		[]
16		$\texttt{crazy}(\texttt{colin})$	$\texttt{crazy}(X) :- \texttt{cs}(X)$	$[X \mapsto \texttt{colin}]$
17		$\texttt{cs}(\texttt{colin})$	$\texttt{cs}(\texttt{colin}) :- \texttt{true}$	[]
18		$\texttt{true}$		[]

Prolog's response is hence: $[X \mapsto \texttt{colin}]$.

- **Consider the Prolog program:**

```
1  a:-not(true).
```

- **Let us query whether** `a`**:**

ID	CP	G	R	μ
1		`a`	`a :-not(true)`	[]
2*		`true`		[]

- **The query infers false by negation as failure (indicated by $*$ which means that a result derived from this step needs to be inverted).**
- **This, however, does not coincide with our understanding of the semantics of the implication: $\bot \longrightarrow$ `a` is true independent of whether `a` or not.**
- **The reason is Prolog's closed-world assumption: It assumed the database is complete; I.e., if the answer cannot be deduced, it is false.**
- **Even worse, the response to the query `not(a)` is true due to two applications of inversion.**

- **Consider the Prolog program:**

```
1  a:-b.
2  b:-a.
```

- **Let us query whether** `a, b`**:**

ID	CP	G	R	μ
1		a, b	a :− b	[]
2		b, b	b :− a	[]
3		a, b	a :− b	[]
4		b, b	b :− a	[]

...

- **The program enters an infinite loop even though the query could be proven true in a few steps:**

$$(\mathtt{b} \longrightarrow \mathtt{a}) \wedge (\mathtt{a} \longrightarrow \mathtt{b}) \longrightarrow \mathtt{a} \wedge \mathtt{b} \Leftrightarrow \top. \tag{223}$$

- **The nature of Prolog being based on Horn logic and its negation and loop handling show a considerable weakness of its inference algorithm.**

- **Apart from Prolog's inference engine, a predominant feature is its list handling.**
- **Lists can be written in three ways:**
  1. **$.(s, t)$ defines a list with the element s and the tail t;**
  2. **$[s|t]$ does the same;**
  3. **$[s_1, \ldots, s_n]$ defines a list with the elements $s_1, \ldots, s_n$**
- **Accordingly, these are equivalent lists:**

$$.(1, .(2, .(3, []))) \tag{224}$$

$$[1|[2|[3|[]]]] \tag{225}$$

$$[1, 2, 3] \tag{226}$$

- **We want to design a function** `cat` **that concatenates two lists** `L1` **and** `L2` **resulting in the list** `L2`**.**
- **In the world of logical programming, this could be conceived as the 3-ary function** `cat(L1,L2,L3)` **which becomes true iff** `L3` **is the concatenation of** `L2` **and** `L2`**.**
- **A respective Prolog program is:**

```
1  cat([X|L1],L2,[X|L3]):-cat(L1,L2,L3).
2  cat([],L,L).
```

- **This program reads**

  An empty list concatenated with a list L results in the same list L (Fact 2). Furthermore, if the concatenation of the lists L_1 and L_2 results in L_3, then L_1 with an preceding element X concatenated with L_2 must result in L_3 with the same preceding element X (Rule 1).

- **In the following, we run an example to understand the program's functionality.**

ID	CP	G	R	μ
1		$\mathtt{cat}([1,2],[3,4],\boldsymbol{Y})$	$\mathtt{cat}([\boldsymbol{X}\|\boldsymbol{L_1}],\boldsymbol{L_2},[\boldsymbol{X}\|\boldsymbol{L_3}]) :-$ $\mathtt{cat}(\boldsymbol{L_1},\boldsymbol{L_2},\boldsymbol{L_3})$	$[\boldsymbol{X}\mapsto[1],\boldsymbol{L_1}\mapsto[2],$ $\boldsymbol{L_2}\mapsto[3,4],\boldsymbol{Y}\mapsto[1\|\boldsymbol{L_3}]]$
2		$\mathtt{cat}([2],[3,4],\boldsymbol{L_3})$	$\mathtt{cat}([\boldsymbol{X'}\|\boldsymbol{L'_1}],\boldsymbol{L'_2},[\boldsymbol{X'}\|\boldsymbol{L'_3}]) :-$ $\mathtt{cat}(\boldsymbol{L'_1},\boldsymbol{L'_2},\boldsymbol{L'_3})$	$[\boldsymbol{X'}\mapsto[2],\boldsymbol{L'_1}\mapsto[],$ $\boldsymbol{L'_2}\mapsto[3,4],\boldsymbol{L_3}\mapsto[2\|\boldsymbol{L'_3}]]$
3		$\mathtt{cat}([],[3,4],\boldsymbol{L'_3})$	$\mathtt{cat}([\boldsymbol{X''}\|\boldsymbol{L''_1}],\boldsymbol{L''_2},[\boldsymbol{X''}\|\boldsymbol{L''_3}]) :-$ $\mathtt{cat}(\boldsymbol{L''_1},\boldsymbol{L''_2},\boldsymbol{L''_3})$	$\boldsymbol{\Omega}$
4		$\mathtt{cat}([],[3,4],\boldsymbol{L'_3})$	$\mathtt{cat}([],\boldsymbol{L},\boldsymbol{L}) :-$	$[\boldsymbol{L}\mapsto[3,4],\boldsymbol{L'_3}\mapsto[3,4]]$

Prolog's response is hence:

$$\begin{aligned}\boldsymbol{Y} &\mapsto [1|\boldsymbol{L_3}]\\ &\mapsto [1|[2|\boldsymbol{L'_3}]]\\ &\mapsto [1|[2|[3,4]]]\\ &= [1,2,3,4]\end{aligned} \tag{227}$$

- **You may perceive some flavor of Prolog's elegance if you consider which use cases the above example function features:**
  - **Concatenate two lists:**

    $$\texttt{cat}([1,2],[3,4],\boldsymbol{Y}). \tag{228}$$

  - **Check whether a list resulted from another list by way of concatenation:**

    $$\texttt{cat}([1,2],\boldsymbol{Y},[1,2,3,4]). \tag{229}$$

  - **Find all possible splits of a list into two lists:**

    $$\texttt{cat}(\boldsymbol{X},\boldsymbol{Y},[1,2,3,4]). \tag{230}$$

- **Consider the following program:**

```
1  a.
2  a:-b.
3  b:-a.
```

- **We get the query result**

```
a.  ⟶  Yes.
```
(231)

- **Now, we reorder the rules:**

```
1  a:-b.
2  a.
3  b:-a.
```

- **This, time, the query result is**

```
a.  ⟶  ERROR : Out of local stack.
```
(232)

- **The inference algorithm keeps accessing Rule 1 over and over again.**
- **Other than the example on Page 145, this time, we do not get an infinite loop but a stack overflow.**
- **This is because Prolog has to create a choice point for every recursion due to the presence of the alternative Rule 3.**

- **Consider the following program:**

```
1  s([X],[X]).
2  s([A,B],[A,D]):-s([B],[D]),A<D.
```

- **We get the query result**

$$\texttt{s([1,2],X).} \quad \longrightarrow \quad \texttt{X} = [1,2]. \tag{233}$$

- **Now, we switch the elements in Rule 2's body:**

```
1  s([X],[X]).
2  s([A,B],[A,D]):-A<D,s([B],[D]).
```

- **This time, we get**

$$\texttt{s([1,2],X).} \quad \longrightarrow \quad \texttt{ERROR : Arguments are not sufficiently instantiated.}$$

- **The inference algorithm tries to evaluate** $\texttt{A} < \texttt{D}$ **first, before** D **had been determined by way of evaluating** $\texttt{s([B],[D])}$.

- **Consider the following program:**

```
1  p(X,Y):-Y==X+1.
2  q(X,Y):-Y>X+0.9999999,Y<X+1.0000001.
3  r(X,Y):-Y is X+1.
4  s(X,Y):-Y=X+1.
```

- **We get the following query results:**

`p(1,2).`	$\longrightarrow$	`No.`
`q(1,2).`	$\longrightarrow$	`Yes.`
`r(1,2).`	$\longrightarrow$	`Yes.`
`s(1,2).`	$\longrightarrow$	`No.`
`p(1,Y).`	$\longrightarrow$	`No.`
`q(1,Y).`	$\longrightarrow$	`ERROR : Arguments are not sufficiently instantiated.`
`r(1,Y).`	$\longrightarrow$	`Y` $= 2.$
`s(1,Y).`	$\longrightarrow$	`Y` $= 1 + 1.$

- The check for equality (==) fails due to issues with Prolog's numerical precision.
- Rather than for equality, `q` checks for a small range around the expected value and thereby succeeds. When queried with the free parameter `Y`, however, Prolog is not able to limit the (real-valued) search space and complains about insufficient instantiation.
- Prolog's keyword `is` assigns the exact value of `X + 1` to `Y` and therefore succeeds. Accordingly, the free parameter `Y` gets assigned the sum of 1 and 1.
- The unification operator $=$ tries to solve the syntactical equation $2 = 1 + 1$ which is not possible since different function symbols cannot be unified. Hence, it fails. When queried with a free parameter, however, the syntactical equation is $\texttt{Y} = 1 + 1$ whose solution is the result set.

- **Write programs to**
  1) **determine the maximum of two numbers (2 lines)**
  2) **calculate the factorial (2 lines)**
  3) **uniq a list (3 lines)**
  4) **find identical elements in two lists (3 lines)**
  5) **sort a list (4 lines)**

Solutions to selected exercises

- **Transform into CNF:**

$$g := \underbrace{p \rightarrow q \rightarrow r}_{a} \leftrightarrow \underbrace{\neg s \vee t}_{b} \tag{234}$$

1.
$$g \Leftrightarrow \underbrace{(\neg a \vee b)}_{c} \wedge \underbrace{(\neg b \vee a)}_{d} \tag{235}$$

2.
$$a \Leftrightarrow \neg p \vee q \rightarrow r$$
$$\Leftrightarrow \neg(\neg p \vee q) \vee r \tag{236}$$

3.
$$a \Leftrightarrow p \wedge \neg q \vee r \tag{237}$$
$$\neg a \Leftrightarrow \neg(p \wedge \neg q) \wedge \neg r$$
$$\Leftrightarrow (\neg p \vee q) \wedge \neg r \tag{238}$$
$$\neg b \Leftrightarrow s \wedge \neg t \tag{239}$$

4.

$$
\begin{aligned}
c &\Leftrightarrow (\neg p \vee q) \wedge \neg r \vee b \\
&\Leftrightarrow (\neg p \vee q \vee b) \wedge (\neg r \vee b) && (240) \\
&\Leftrightarrow (\neg p \vee q \vee \neg s \vee t) \wedge (\neg r \vee \neg s \vee t) \\
d &\Leftrightarrow s \wedge \neg t \vee a \\
&\Leftrightarrow \underbrace{(s \vee a)}_{e} \wedge \underbrace{(\neg t \vee a)}_{f} && (241) \\
e &\Leftrightarrow p \wedge \neg q \vee r \vee s \\
&\Leftrightarrow (p \vee r \vee s) \wedge (\neg q \vee r \vee s) && (242) \\
f &\Leftrightarrow (p \vee r \vee \neg t) \wedge (\neg q \vee r \vee \neg t) && (243)
\end{aligned}
$$

5.

$$
\begin{aligned}
g &\Leftrightarrow \{\{\neg p, q, \neg s, t\}, \{\neg r, \neg s, t\}, \{p, r, s\}, \{\neg q, r, s\}, \\
&\quad \{p, r, \neg t\}, \{\neg q, r, \neg t\}\} && (244)
\end{aligned}
$$

- **We know that**

  Everybody loves only those people who do not love a gardener.

- **Let us use the predicates**
  - $l(x, y)$ **which is** 1 **iff** x **loves** y **and**
  - $g(x)$ **which is** 1 **iff** x **is a gardener.**
- **Let us further use an auxiliary predicate**
  - $n(x)$ **which is** 1 **iff** x **does not love a gardener.**
- **Using** $n(x)$**'s definition, the above axiom could be written as**

  Every x *loves only those* y *for which* $n(y)$

  which can be formally expressed as

$$f := \forall x, y(l(x, y) \rightarrow n(y)). \tag{245}$$

- **Expressing $n(y)$ in terms of $l(y,z)$ and $g(z)$:**

$$n(y) \leftrightarrow \neg\exists z(l(y,z) \wedge g(z)). \tag{246}$$

- **Hence, Formula 248 becomes**

$$\begin{aligned} f &\leftrightarrow \forall x,y(l(x,y) \rightarrow \neg\exists z(l(y,z) \wedge g(z))) \\ &\leftrightarrow \forall x,y(\neg l(x,y) \vee \neg\exists z(l(y,z) \wedge g(z))) \\ &\leftrightarrow \forall x,y(\neg l(x,y) \vee \forall z(\neg(l(y,z) \wedge g(z)))) \\ &\leftrightarrow \forall x,y,z(\neg l(x,y) \vee \neg l(y,z) \vee \neg g(z)) \\ &\leftrightarrow \{\{\neg l(x,y), \neg l(y,z), \neg g(z)\}\} \end{aligned} \tag{247}$$

- **Our conjecture is that**

  Gardeners do not love themselves

  which can be formally expressed as

$$g := \forall x(g(x) \rightarrow \neg l(x,x)). \tag{248}$$

- **To prove g, we need to conjunctively combine f with g's negation, so, let us attack the latter now:**

$$\begin{aligned}
\neg g &\leftrightarrow \neg(\forall x(g(x) \rightarrow \neg l(x,x))) \\
&\leftrightarrow \neg(\forall x(\neg g(x) \vee \neg l(x,x))) \\
&\leftrightarrow \exists x(\neg(\neg g(x) \vee \neg l(x,x))) \\
&\leftrightarrow \exists x(g(x) \wedge l(x,x)) \\
&\approx g(s) \wedge l(s,s) \\
&\leftrightarrow \{\{g(s)\},\{l(s,s)\}\} \qquad (249)
\end{aligned}$$

- **Applying the DP algorithm:**

$$
\begin{aligned}
C_0 &:= \{\{\neg l(x,y), \neg l(x,z), \neg g(z)\}, \{g(s)\}, \{l(s,s)\}\} \\
C_1 &:= \mathsf{res}(C_0, g(s)) \\
&= \{\{\neg l(x,y), \neg l(x,z), \neg g(z)\}, \{\neg l(x,y), \neg l(x,s)\}, \\
&\quad \{g(s)\}, \{l(s,s)\}\} \\
C_2 &:= \mathsf{res}(C_1, l(s,s)) \\
&= \{\{\neg l(x,y), \neg l(x,z), \neg g(z)\}, \{\neg l(s,z), \neg g(z)\}, \\
&\quad \{\neg l(x,y), \neg l(x,s)\}, \underline{\{\neg l(s,s)\}}, \{g(s)\}, \underline{\{l(s,s)\}}
\end{aligned} \tag{250}
$$

- **This proves that the conjecture is true.**