

# Logic for Computer Scientists

Pascal Hitzler

CS 2210 Lecture, Fall Semester 2013  
Wright State University, Dayton, OH, U.S.A.  
<http://www.pascal-hitzler.de>

[version: December 3, 2013]

## Contents

<b>1</b>	<b>Datalog</b>	<b>3</b>
1.A	Informal Examples . . . . .	3
1.B	Syntax and Formal Semantics . . . . .	7
1.C	Fixed-point Semantics . . . . .	13
<b>2</b>	<b>Propositional Logic</b>	<b>18</b>
2.A	Syntax . . . . .	18
2.B	Semantics . . . . .	19
2.C	Datalog Revisited: Semantics By Grounding . . . . .	22
2.D	Equivalence . . . . .	24
2.E	Normal Forms . . . . .	25
2.F	Tableaux Algorithm . . . . .	27
<b>3</b>	<b>First-order Predicate Logic</b>	<b>31</b>
3.A	Syntax . . . . .	31
3.B	Semantics . . . . .	32
3.C	Datalog Revisited . . . . .	37
3.D	Equivalence . . . . .	37
3.E	Normal Forms . . . . .	38
3.F	Tableaux Algorithm . . . . .	39
3.G	Theoretical Aspects . . . . .	41
<b>4</b>	<b>Application: Knowledge Representation for the World Wide Web</b>	<b>43</b>

## References

- [Ben-Ari, 1993] Ben-Ari, M. (1993). *Mathematical Logic for Computer Science*. Springer.
- [Cryan et al., 2004] Cryan, D., Shatil, S., and Mayblin, B. (2004). *Introducing Logic: A Graphic Guide*. Icon Books, 4th edition.
- [Hitzler et al., 2009] Hitzler, P., Krötzsch, M., and Rudolph, S. (2009). *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC.
- [Schöning, 1989] Schöning, U. (1989). *Logic for Computer Scientists*. Birkhäuser.

# 1 Datalog

[no textbook reference]

## 1.A Informal Examples

### 1.1 Example

We want to formalize the following statements.

- Marian is the mother of Michelle.
- Craig is the brother of Michelle.
- Ann is the mother of Barack.
- Barack is the father of Malia.
- Michelle is the mother of Malia.
- Barack is the father of Natasha.
- Michelle is the mother of Natasha.
- Craig is male.
- Natasha is female.

We can write these as so-called *Datalog facts*:

motherOf(marian, michelle)	(1)
brotherOf(craig, michelle)	(2)
motherOf(ann, barack)	(3)
fatherOf(barack, malia)	(4)
motherOf(michelle, malia)	(5)
fatherOf(barack, natasha)	(6)
motherOf(michelle, natasha)	(7)
male(craig)	(8)
female(natasha)	(9)

Say, we also want to formalize the following.

- Every father of a person is also a parent of that person.

- Every mother of a person is also a parent of that person.
- If somebody is the mother of another person, who in turn is the parent of a third person, then this first person is the grandmother of this third person.
- If a person is the brother of another person, and this other person is the parent of a third person, then this first person is the uncle of this third person.
- Every father is male.

We can write these as so-called *Datalog rules*:

$$\text{fatherOf}(x, y) \rightarrow \text{parentOf}(x, y) \quad (10)$$

$$\text{motherOf}(x, y) \rightarrow \text{parentOf}(x, y) \quad (11)$$

$$\text{motherOf}(x, y) \wedge \text{parentOf}(y, z) \rightarrow \text{grandmotherOf}(x, z) \quad (12)$$

$$\text{brotherOf}(x, y) \wedge \text{parentOf}(y, z) \rightarrow \text{uncleOf}(x, z) \quad (13)$$

$$\text{fatherOf}(x, y) \rightarrow \text{male}(x) \quad (14)$$

If we take all statements (1) to (14) together, then we can derive new knowledge, which is *implicit* in these statements, e.g. the following.

$$\text{from (4) and (10):} \quad \text{parentOf}(\text{barack}, \text{malia}) \quad (15)$$

$$\text{from (2), (7), (11) and (13):} \quad \text{uncleOf}(\text{craig}, \text{natasha}) \quad (16)$$

$$\text{from (3), (15) and (12):} \quad \text{grandmotherOf}(\text{ann}, \text{malia}) \quad (17)$$

Note that we reused (15) to derive (17). Derived knowledge can be used to derive even further knowledge.

## 1.2 Example

Consider the following sentences.

- Every human is mortal.
- Socrates is a human.

We can write these in Datalog as follows.

$$\text{human}(x) \rightarrow \text{mortal}(x)$$

$$\text{human}(\text{socrates})$$

From these two rules we can derive

$$\text{mortal}(\text{socrates}).$$

### 1.3 Example

Consider the following facts.

newsFrom(*Merkel is Chancellor*, berlin)  
newsFrom(*Obamacare is constitutional*, dc)  
⋮

And furthermore assume there is a set of facts about locations of cities.

locatedIn(berlin, germany)  
locatedIn(dc, usa)  
⋮

We can also state the following Datalog rule.

$$\text{newsFrom}(x, y) \wedge \text{locatedIn}(y, z) \rightarrow \text{newsFrom}(x, z)$$

Derived knowledge is then, e.g., the following.

newsFrom(*Merkel is Chancellor*, germany)  
newsFrom(*Obamacare is constitutional*, usa)

### 1.4 Example

In Datalog, we can state, e.g., that locatedIn is *transitive*:

$$\text{locatedIn}(x, y) \wedge \text{locatedIn}(y, z) \rightarrow \text{locatedIn}(x, z)$$

**Exercise 1 (hand-in)** Can you derive the following from (1) to (14)? Justify your answers.

- (a) Michelle is a parent of Malia.
- (b) Ann is a grandmother of Natasha.

**Exercise 2 (hand-in)** Write the following sentences as Datalog rules.

- (a) Every mother is female.
- (b) If somebody is the father of a female person, then that female person is the daughter of this father.
- (c) If a person is the daughter of somebody's daughter, then this first person is the granddaughter of this "somebody."

**Exercise 3 (hand-in)** In the context of (1) to (14), write Datalog rules

- (a) which define what an aunt is

(b) and which define what a niece is.

Explain your answers.

**Exercise 4 (hand-in)** In the context of (1) to (14),

(a) define siblingOf and

(b) state that siblingOf is symmetric.

Explain your answers.

### 1.5 Example

We can write directed graphs as Datalog facts, e.g., as follows. If  $V = \{a, b, c, d\}$  is the set of vertices of the graph, and  $E = \{(a, b), (b, b), (b, c), (d, d), (d, a), (d, b)\}$  is the set of edges of the graph (see Figure 1), then we can write it as follows.

edge( $a, b$ )  
edge( $b, b$ )  
edge( $b, c$ )  
edge( $d, d$ )  
edge( $d, a$ )  
edge( $d, b$ )

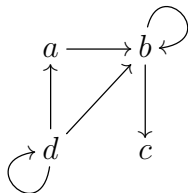


Figure 1: Figure for Example 1.5.

We can now formally define what it means that there is a path from a vertex to another:

$$\begin{aligned} \text{edge}(x, y) &\rightarrow \text{path}(x, y) \\ \text{path}(x, y) \wedge \text{path}(y, z) &\rightarrow \text{path}(x, z) \end{aligned}$$

Then we can derive, e.g., the following.

path( $a, b$ )  
path( $b, c$ )  
path( $a, c$ )

We can also specify that two vertices are connected if there is a path in either direction.

$$\begin{aligned} \text{path}(x, y) &\rightarrow \text{connected}(x, y) \\ \text{connected}(x, y) &\rightarrow \text{connected}(y, x) \end{aligned}$$

Then we can derive, e.g., the following.

connected( $a, b$ )  
connected( $b, a$ )  
connected( $a, c$ )  
connected( $c, a$ )

**Exercise 5 (hand-in)** A vertex  $v$  in a graph is *self-connected* if there is a path from  $v$  to  $v$  in the graph.

By extending the Datalog facts and rules from Example 1.5, complete the datalog rule

$\dots \rightarrow \text{sc}(x)$

such that a vertex  $v$  is self-connected if and only if  $\text{sc}(v)$  can be derived. Justify your answer.

9/5/13

## 1.B Syntax and Formal Semantics

### 1.6 Definition

A *Datalog language*  $L = (V, C, R)$  consists of the following.

- A finite set  $V$  of *variables*:  $x_1, x_2, \dots, x_n$  (also  $y, z, \dots$ ).
- A finite non-empty set  $C$  of *constants*:  $a, b, c, \dots$
- A finite non-empty set  $R$  of *predicate symbols*:  $p_1, p_2, \dots$  (also  $q, r, \dots$ ), each with an *arity* ( $\in \mathbb{N}$ ) (number of parameters).

An *atom* (or *atomic formula*) is of the form

$p(v_1, \dots, v_n)$ ,

where  $p$  is a predicate symbol of arity  $n$  and each of the  $v_i$  is either a constant or a variable. An atom is called a *ground atom* if all the  $v_i$  are constants.

### 1.7 Example

Let  $L$  consist of constants  $a, b$ , of variables  $x, y$ , and of predicate symbols  $p$  with arity 1 and  $q$  with arity 2.

Then the following are examples for atomic formulas over  $L$ .

$p(a), \quad p(y), \quad q(a, b), \quad q(b, b), \quad q(b, x), \quad q(y, y)$

Of these,  $p(a)$ ,  $q(a, b)$  and  $q(b, b)$  are ground atoms.

The following are *not* atomic formulas over  $L$ :

$p(a, b), \quad q(x), \quad p(c), \quad a(x)$

**Exercise 6 (hand-in)** Let  $L = (V, C, R)$  with  $V = \{w, y\}$ ,  $C = \{d, e\}$  and  $R = \{r, s\}$  where  $r$  has arity 1 and  $s$  has arity 2. Which of the following are atoms over  $L$ ? Which are ground atoms? Justify your answers.

- (a)  $d(w, w)$
- (b)  $r(d, e)$
- (c)  $s(w, w)$
- (d)  $r(y)$

**Exercise 7 (hand-in)** Let  $L = (V, C, R)$  with  $V = \{x, y\}$ ,  $C = \{\text{barack, michelle, craig, malia}\}$  and  $R = \{\text{motherOf, parentOf, grandmotherOf}\}$ , all with arity 2. Which of the Datalog facts (1) to (9) from Example 1.1 are atoms over  $L$ ? Justify your answers.

### 1.8 Definition

A *Datalog rule* is a statement of the form

$$B_1 \wedge \cdots \wedge B_n \rightarrow A,$$

where the  $B_i$  and  $A$  are atoms.  $B_1 \wedge \cdots \wedge B_n$  is called the *body* of the rule, each  $B_i$  is called a *body atom* of the rule, and  $A$  is called the *head* of the rule.

A rule with  $n = 0$ , i.e. with no body, is called a *fact*, and the arrow is omitted in this case. A *Datalog program* is a set of Datalog rules.

### 1.9 Example

The following are examples of Datalog rules.

$$\begin{aligned} \text{newsFrom}(x, y) \wedge \text{locatedIn}(y, z) &\rightarrow \text{newsFrom}(x, z) \\ p(a, x) \wedge q(x, y) &\rightarrow r(a, x, y) \\ p_2(a) \wedge p_3 &\rightarrow q_2(a) \end{aligned}$$

Note that in this example,  $p_3$  is a predicate symbol of arity 0.

### 1.10 Example

The statements (1) to (14) from Example 1.1 constitute a Datalog program. Statements (1) to (9) are facts.

**Exercise 8 (hand-in)** Write a Datalog program which captures the following natural language sentences.

- (a) Every human being has a parent.
- (b) If somebody is an orphan, then all his parents are dead.
- (c) Every orphan is a human being.



- (d) Somebody's father is also that person's parent.
- (e) Harry Potter is an orphan.
- (f) James Potter is the father of Harry Potter.

### 1.11 Definition

Given a Datalog language  $L$  and a Datalog program  $P$  over  $L$ , a *Herbrand interpretation* for  $P$  is a set of ground atoms over  $L$ .

### 1.12 Example

Consider  $P$  to consist only of the following statements from Example 1.1 (with abbreviated notation,  $c, m, n$  are constants).

$$\begin{aligned} \text{mOf}(x, y) &\rightarrow \text{pOf}(x, y) \\ \text{bOf}(x, y) \wedge \text{pOf}(y, z) &\rightarrow \text{uOf}(x, z) \\ &\text{bOf}(c, m) \\ &\text{mOf}(m, n) \end{aligned}$$

Then the following are examples for Herbrand interpretations.

$$\begin{aligned} I_1 &= \{\text{bOf}(c, m), \text{mOf}(m, n), \text{pOf}(m, n), \text{uOf}(c, n)\} \\ I_2 &= \{\text{bOf}(m, c), \text{mOf}(c, n), \text{pOf}(m, n), \text{uOf}(n, c)\} \end{aligned}$$

### 1.13 Example

Some examples for interpretations of the following Datalog program, where  $a, b, c, d$  are constants.

$$\begin{aligned} &p(a, b) \\ &p(b, c) \\ &p(c, a) \\ &p(d, d) \\ &p(x, y) \rightarrow q(x, y) \\ &q(x, y) \wedge q(y, z) \rightarrow q(x, z) \\ &q(x, y) \rightarrow r(x, y) \\ &r(x, y) \rightarrow r(y, x) \\ &r(x, x) \rightarrow t(x) \end{aligned}$$

are the following.

$$\begin{aligned} I_1 &= \{p(c, a), p(c, b), t(a)\} \\ I_2 &= \emptyset \\ I_3 &= \{p(a, b), p(b, c), p(c, a), p(d, d), q(a, b), r(a, b), r(b, a), t(a), t(b)\} \end{aligned}$$

**Exercise 9 (hand-in)** Give three distinct Herbrand interpretations for the following Datalog program, where  $a, b$  are constants.

$$\begin{aligned} & q(a) \\ & p(b) \\ & q(x) \rightarrow p(x) \\ & q(y) \wedge p(y) \rightarrow r(b) \end{aligned}$$

### 1.14 Definition

A *substitution*  $[x_1/c_1, \dots, x_n/c_n]$ , where the  $x_i$  are variables and the  $c_i$  are constants, is a mapping which maps each Datalog rule  $R$  to the formula  $R[x_1/c_1, \dots, x_n/c_n]$ , which is obtained from  $R$  by replacing all occurrences of  $x_i$  by  $c_i$ , for all  $i = 1, \dots, n$ .

### 1.15 Example

In the following,  $a, b, c, d$  are constants, while  $x, y, z$  are variables.

$$(p(x, y) \wedge q(y, z) \rightarrow r(x, z))[x/a, z/b] = p(a, y) \wedge q(y, b) \rightarrow r(a, b) \quad (18)$$

$$(q(x) \wedge r(x, y) \rightarrow p(y))[x/b, y/c] = q(b) \wedge r(b, c) \rightarrow p(c) \quad (19)$$

$$q(x, z, y)[x/b, z/b, y/a] = q(b, b, a) \quad (20)$$

$$\begin{aligned} (p(x, y) \wedge q(y, z, z) \rightarrow r(x, y))[x/b][x/a][y/c] &= (p(b, y) \wedge q(y, z, z) \rightarrow r(b, y))[x/a][y/c] \\ &= (p(b, y) \wedge q(y, z, z) \rightarrow r(b, y))[y/c] \\ &= (p(b, c) \wedge q(c, z, z) \rightarrow r(b, c)) \end{aligned} \quad (21)$$

**Exercise 10 (hand-in)** Evaluate the following.

(a)  $(p(x, y, x) \wedge q(x, y, y) \wedge r(y, y) \rightarrow t(x))[x/a, y/b] = \dots$

(b)  $(p(x) \wedge q(x) \rightarrow r(x))[x/c][x/d] = \dots$

(c)  $(q(a, x) \wedge p(x, y) \wedge q(y, a) \rightarrow r(y))[x/a][x/b] = \dots$

(d)  $(p(x, x) \wedge q(x, y) \rightarrow p(x, y))[y/b][y/c][x/b] = \dots$

9/10/13

### 1.16 Definition

A *ground rule* is a Datalog rule which contains no variables. A substitution  $\varphi$  for a Datalog rule  $R$  is called a *ground substitution* for  $R$  if  $R\varphi$  is a ground rule.

### 1.17 Example

In Example 1.15, the substitutions in (19) and (20) are ground substitutions for these rules, while those in (18) and (21) are not.

**Exercise 11 (hand-in)** Which of the substitutions in Exercise 10 are ground substitutions?

### 1.18 Definition

Given a Datalog rule  $R$ , we define  $\text{ground}(R)$  to be the Datalog program which consists of all ground rules  $R\varphi$  which can be obtained from  $R$  via a ground substitution  $\varphi$  for  $R$ . In other words,

$$\text{ground}(R) = \{R\varphi \mid \varphi \text{ is a ground substitution for } R\}.$$

Each  $S \in \text{ground}(R)$  is called a *grounding* of  $R$ .

Given a Datalog program  $P$ , we define the *grounding* of  $P$  as

$$\text{ground}(P) = \bigcup_{R \in P} \text{ground}(R).$$

### 1.19 Remark

$\text{ground}(P)$  is always a finite set if  $P$  is finite, since the underlying language  $L$  by definition has only a finite number of constants.

If  $L$  is not explicitly given, then we assume that the set  $C$  of constants in  $L$  contains exactly the constants occurring in  $P$ .

### 1.20 Example

For the program  $P$  in Example 1.12,  $\text{ground}(P)$  consists of the following rules.

$$\begin{aligned} & \text{mOf}(c, c) \rightarrow \text{pOf}(c, c) \\ & \text{mOf}(c, m) \rightarrow \text{pOf}(c, m) \\ & \text{mOf}(c, n) \rightarrow \text{pOf}(c, n) \\ & \text{mOf}(m, c) \rightarrow \text{pOf}(m, c) \\ & \text{mOf}(m, m) \rightarrow \text{pOf}(m, m) \\ & \text{mOf}(m, n) \rightarrow \text{pOf}(m, n) \\ & \text{mOf}(n, c) \rightarrow \text{pOf}(n, c) \\ & \text{mOf}(n, m) \rightarrow \text{pOf}(n, m) \\ & \text{mOf}(n, n) \rightarrow \text{pOf}(n, n) \\ & \text{bOf}(c, c) \wedge \text{pOf}(c, c) \rightarrow \text{uOf}(c, c) \\ & \text{bOf}(c, m) \wedge \text{pOf}(m, n) \rightarrow \text{uOf}(c, m) \\ & \text{bOf}(c, n) \wedge \text{pOf}(n, m) \rightarrow \text{uOf}(c, n) \\ & \qquad \qquad \qquad \vdots \qquad \text{overall 27 groundings of this rule} \\ & \text{bOf}(c, m) \\ & \text{mOf}(m, n) \end{aligned}$$

**Exercise 12 (hand-in)** Give the grounding of the Datalog program from Exercise 9.

### 1.21 Definition

A Herbrand interpretation  $I$  of  $P$  is called a *Herbrand model* of  $P$  if the following condition holds: For every rule

$$B_1 \wedge \cdots \wedge B_n \rightarrow A$$

in  $\text{ground}(P)$  with  $\{B_1, \dots, B_n\} \subseteq I$ , we also have  $A \in I$ .

### 1.22 Example

In Example 1.12,  $I_1$  is a Herbrand model of  $P$ , while  $I_2$  is not a model of  $P$ .

### 1.23 Example

For the Datalog program  $P$  consisting of the rules

$$\begin{aligned} & p(a) \\ & q(a, b) \\ & q(b, c) \\ & p(x) \rightarrow r(x) \\ & r(x) \wedge q(x, y) \rightarrow r(y) \\ & r(x) \wedge q(y, x) \rightarrow q(x, y) \end{aligned}$$

the following are Herbrand models:

$$\begin{aligned} & \{p(a), q(a, b), q(b, c), r(a), r(c), q(c, b), r(b), q(b, a)\} \\ & \{p(a), q(a, b), q(b, c), r(a), r(c), q(c, b), r(b), q(b, a), q(c, c)\} \end{aligned}$$

However,

$$\{p(a), q(a, b), q(b, c), r(a), r(c), q(c, b), r(b), q(b, a), q(a, c)\}$$

is *not* a Herbrand model.

### 1.24 Example

The Datalog program  $P$  consisting of the rules

$$\begin{aligned} & p(a) \\ & q(b) \\ & q(x) \rightarrow q(x) \end{aligned}$$

has the following Herbrand models:

$$\begin{aligned} & \{p(a), q(b)\} \\ & \{p(a), q(b), q(a)\} \\ & \{p(a), q(b), q(a), p(b)\} \end{aligned}$$

**Exercise 13 (hand-in)** Give a Herbrand model for the Datalog program in Exercise 9.

**Exercise 14 (hand-in)** Give three distinct Herbrand models for the Datalog program  $P$  consisting of the following rules.

$$\begin{aligned} & p(a, b) \\ & q(c) \\ & p(x, y) \rightarrow q(x) \end{aligned}$$

## 1.C Fixed-point Semantics

### 1.25 Example

Consider the program  $P$  from Example 1.23. There is a systematic way of obtaining a Herbrand model, as follows. In order to do this, consider  $\text{ground}(P)$ .

First, collect all facts from  $\text{ground}(P)$ :

$$I_1 = \{p(a), q(a, b), q(b, c)\}.$$

Next, collect all heads of rules in  $\text{ground}(P)$  for which all body atoms are in  $I_1$ , and add them to  $I_1$ :

$$I_2 = I_1 \cup \{r(a)\}.$$

Next, collect all heads of rules in  $\text{ground}(P)$  for which all body atoms are in  $I_2$ , and add them to  $I_2$ :

$$I_3 = I_2 \cup \{r(b)\}.$$

Iteratively continue this process until nothing is added any more:

$$I_4 = I_3 \cup \{r(c), q(b, a)\}$$

$$I_5 = I_4 \cup \{q(c, b)\}$$

$$I_6 = I_5 \cup \emptyset = I_5$$

We have already seen in Example 1.23 that  $I_5$  is indeed a Herbrand model.

We now develop this idea systematically.

### 1.26 Definition

Given a Datalog program  $P$  with underlying language  $L$ , let  $B_P$  be the set of all ground atoms over  $L$ , called the *Herbrand base* of  $P$ . Furthermore, let  $I_P = 2^{B_P}$  be the power set of  $B_P$ , i.e., the set of all subsets of  $B_P$ .

### 1.27 Example

For  $P$  as in Example 1.23, we have

$$B_P = \{p(a), p(b), p(c), r(a), r(b), r(c), \\ q(a, a), q(a, b), q(a, c), q(b, a), q(b, b), q(b, c), q(c, a), q(c, b), q(c, c)\}$$

which consists of 15 atoms. Correspondingly,  $I_P$  has  $2^{15} = 32,768$  elements.

**Exercise 15 (hand-in)** Give  $B_P$  for  $P$  as in Exercise 1.24. How many elements does  $I_P$  have?

### 1.28 Remark

$I_P$  is in fact the set of all Herbrand interpretations for  $P$ .

### 1.29 Definition

Given a Datalog program  $P$ , define a function  $T_P : I_P \rightarrow I_P$  by

$$T_P(I) = \{A \in B_P \mid (B_1 \wedge \cdots \wedge B_n \rightarrow A) \in \text{ground}(P) \text{ and } B_i \in I \text{ for all } i = 1, \dots, n\}.$$

This function is called the *single-step operator*, or *immediate consequence operator*, or simply the  $T_P$ -operator for  $P$ .

### 1.30 Example

Considering Example 1.25, we have

$$\begin{aligned} T_P(\emptyset) &= I_1 \\ T_P(I_1) &= I_2 \\ T_P(I_2) &= I_3 \\ T_P(I_3) &= I_4 \\ T_P(I_4) &= I_5 \\ T_P(I_5) &= I_5 \end{aligned}$$

To give some further examples, we also have

$$\begin{aligned} T_P(\{r(c), q(c, c)\}) &= \{p(a), q(a, b), q(b, c), r(c), q(c, c)\} \\ T_P(\{p(b)\}) &= \{p(a), q(a, b), q(b, c), r(b)\} \end{aligned}$$

### 1.31 Example

For  $P$  as in Example 1.24, we have

$$\begin{aligned} T_P(\{q(c)\}) &= \{p(a), q(b), q(c)\} \\ T_P(\{p(a), q(b), q(c)\}) &= \{p(a), q(b), q(c)\} \end{aligned}$$

**Exercise 16 (hand-in)** For the program  $P$  in Example 1.23, compute the following.

- (a)  $T_P(\{p(c), q(c, c)\})$
- (b)  $T_P(B_P)$

### 1.32 Definition

Given a Datalog program  $P$  and  $I \in I_P$ , we call  $I$  a *pre-fixed point* of  $T_P$  if  $T_P(I) \subseteq I$ . We call  $I$  a *fixed point* of  $T_P$  if  $T_P(I) = I$ .

9/17/13

### 1.33 Example

In Example 1.25, we have  $T_P(I_5) = I_5$ , hence  $I_5$  is a fixed point of  $T_P$ .

### 1.34 Example

For  $P$  as in Example 1.24, we have

$$T_P(\{p(a), q(b)\}) = \{p(a), q(b)\},$$

which therefore is a fixed point of  $T_P$ .

We also have

$$T_P(\{p(a), q(b), p(b)\}) = \{p(a), q(b)\} \subseteq \{p(a), q(b), p(b)\},$$

therefore  $\{p(a), q(b), p(b)\}$  is a pre-fixed point of  $T_P$ .

**Exercise 17 (hand-in)** With respect to Example 1.23, verify that  $B_P$  is a pre-fixed point of  $T_P$ .

**Exercise 18 (hand-in)** Give three pre-fixed points and one fixed point of the  $T_P$ -operator for  $P$  as in Exercise 14.

### 1.35 Theorem

Given any Datalog program  $P$ , the pre-fixed points of  $T_P$  are exactly the Herbrand models of  $P$ .

**Proof:** Let  $I$  be a pre-fixed point of  $T_P$ , i.e.,  $T_P(I) \subseteq I$ . Now let  $B_1 \wedge \dots \wedge B_n \rightarrow A$  be any rule in  $\text{ground}(P)$ . If  $\{B_1, \dots, B_n\} \subseteq I$ , then  $A \in T_P(I)$  by definition of  $T_P$ , and hence  $A \in I$  by the assumption that  $T_P(I) \subseteq I$ . This shows that  $I$  is a Herbrand model of  $P$ .

Conversely, let  $I$  be a Herbrand model of  $P$ . Now for any  $A \in T_P(I)$  there must be a rule  $B_1 \wedge \dots \wedge B_n \rightarrow A$  in  $\text{ground}(P)$  with  $\{B_1, \dots, B_n\} \subseteq I$ . Since  $I$  is a Herbrand model we obtain  $A \in I$ . This shows  $T_P(I) \subseteq I$ . ■

### 1.36 Lemma

Given any Datalog program  $P$  and  $I_1 \subseteq I_2 \in I_P$ , we have that

$$T_P(I_1) \subseteq T_P(I_2),$$

i.e., the  $T_P$ -operator is *monotonic*.

**Proof:** For any  $A \in T_P(I_1)$  there must be a rule  $B_1 \wedge \dots \wedge B_n \rightarrow A$  in  $\text{ground}(P)$  with  $\{B_1, \dots, B_n\} \subseteq I_1$ . Since  $I_1 \subseteq I_2$  we obtain  $\{B_1, \dots, B_n\} \subseteq I_2$ , and hence  $A \in T_P(I_2)$  as required. ■

### 1.37 Definition

Given any Datalog program  $P$ , we iteratively define the following.

$$\begin{aligned} T_P \uparrow 0 &= \emptyset \\ T_P \uparrow 1 &= T_P(T_P \uparrow 0) \\ &\vdots \\ T_P \uparrow (n+1) &= T_P(T_P \uparrow n) \end{aligned}$$

We furthermore define

$$T_P \uparrow \omega = \bigcup_{n \in \mathbb{N}} T_P \uparrow n.$$

The sets  $T_P \uparrow n$  are called *iterates* of the  $T_P$ -operator.

### 1.38 Example

Returning to Example 1.25, we have  $I_n = T_P \uparrow n$  for all  $n = 1, \dots, 6$  and  $T_P \uparrow \omega = I_5$ .

**Exercise 19 (hand-in)** Compute  $T_P \uparrow n$  for all  $n \in \mathbb{N}$  and  $T_P \uparrow \omega$  for  $P$  as in Example 1.24.

**Exercise 20 (hand-in)** Compute  $T_P \uparrow n$  for all  $n \in \mathbb{N}$  and  $T_P \uparrow \omega$  for  $P$  as in Exercise 14.

**Exercise 21 (hand-in)** Compute  $T_P \uparrow n$  for all  $n \in \mathbb{N}$  and  $T_P \uparrow \omega$  for  $P$  as in Exercise 9.

**Exercise 22** Compute  $T_P \uparrow n$  for all  $n \in \mathbb{N}$  and  $T_P \uparrow \omega$  for  $P$  as in Example 1.13.

**Exercise 23** Compute  $T_P \uparrow n$  for all  $n \in \mathbb{N}$  and  $T_P \uparrow \omega$  for  $P$  as in Example 1.12.

**Exercise 24 (if coding helps you with the material)** Write a computer program (you may choose your favorite language), which accepts as input graphs specified in the form of Example 1.5, and computes all  $T_P \uparrow n$ , where  $P$  consists of all the non-fact Datalog rules from Example 1.5, plus the input graph encoded as facts.

10/01/13

### 1.39 Theorem

For every Datalog program  $P$ , the following hold.

- (a)  $T_P \uparrow \omega$  is a fixed point of  $T_P$ .
- (b)  $T_P \uparrow \omega$  is a Herbrand model for  $P$ .
- (c) For every Herbrand model  $M$  for  $P$  we have that  $T_P \uparrow \omega \subseteq M$ .

Condition (c) states that  $T_P \uparrow \omega$  is the *least* Herbrand model of  $P$  (with respect to the set inclusion ordering).

**Proof:** First note that

$$T_P \uparrow 0 = \emptyset \subseteq T_P(\emptyset) = T_P \uparrow 1,$$

and due to monotonicity of  $T_P$ , we obtain

$$T_P \uparrow n \subseteq T_P \uparrow (n + 1)$$

for each  $n \in \mathbb{N}$ . Thus, the iterates of  $T_P$  form an increasing chain:

$$T_P \uparrow 0 \subseteq T_P \uparrow 1 \subseteq T_P \uparrow 2 \subseteq \dots T_P \uparrow n \subseteq T_P \uparrow (n + 1) \subseteq \dots \subseteq T_P \uparrow \omega.$$

Since furthermore  $T_P \uparrow \omega \subseteq B_P$ , and  $B_P$  is a finite set, there must be an  $n_P$  such that  $T_P \uparrow n_P = T_P \uparrow (n_P + 1) = T_P \uparrow \omega$ , i.e.,  $T_P \uparrow n_P = T_P \uparrow \omega$  must be a fixed point of  $T_P$ . This shows (a).

Every fixed point of  $T_P$  is a pre-fixed point of  $T_P$ . Hence  $T_P \uparrow \omega$  is a pre-fixed point of  $T_P$  and thus a Herbrand model for  $P$  by Theorem 1.35. This shows (b).

Now assume that  $M$  is another Herbrand model for  $P$ . Clearly,  $\emptyset \subseteq M$ , and thus  $T_P \uparrow 0 = T_P(\emptyset) \subseteq T_P(M) \subseteq M$  by monotonicity of  $T_P$  and since  $M$  is a pre-fixed point of  $T_P$ . By iteratively repeating this argument we obtain  $T_P \uparrow n \subseteq T_P(M) \subseteq M$  for all  $n \in \mathbb{N}$ , and thus  $T_P \uparrow \omega \subseteq M$ , which shows (c). ■



**Exercise 25 (hand-in)** Given a Datalog program  $P$ , an interpretation  $I \subseteq B_P$  is said to be *supported* if for every  $A \in I$  there exists a rule  $B_1 \wedge \dots \wedge B_n \rightarrow A$  in  $\text{ground}(P)$  with  $\{B_1, \dots, B_n\} \subseteq I$ .

Show the following.

- (a) An interpretation  $I \in I_P$  is supported if and only if  $I \subseteq T_P(I)$ .
- (b) The least Herbrand model of any program is supported.

#### 1.40 Definition

We say that a Datalog program  $P$  *Herbrand-entails* a ground atom  $A$ , written  $P \models_H A$ , if  $A \in T_P \uparrow \omega$ . In this case we also call  $A$  a *logical consequence of  $P$  under the Herbrand semantics*.

**Exercise 26 (hand-in)** Show that the Datalog program from Example 1.1 Herbrand-entails

grandmotherOf(ann, malia).

10/15/13

#### 1.41 Theorem

Given a Datalog program  $P$  and a ground atom  $A$ , we have  $P \models_H A$  if and only if  $A \in M$  for all Herbrand models  $M$  of  $P$ .

**Proof:** If  $P \models_H A$ , then  $A \in T_P \uparrow \omega$ . For any Herbrand model  $M$  of  $P$  we know that  $T_P \uparrow \omega \subseteq M$ , and we obtain  $A \in M$  as required.

If  $A \in M$  for all Herbrand models  $M$  of  $P$ , then we obtain  $A \in T_P \uparrow \omega$  because  $T_P \uparrow \omega$  is a Herbrand model of  $P$ . Thus,  $P \models_H A$ . ■

#### 1.42 Example

Consider Example 1.24, note that  $\{p(a), q(b)\}$  is the intersection of all the Herbrand models, and thus contains exactly the logical consequences of  $P$ .

## 2 Propositional Logic

9/24/13

### 2.A Syntax

[Schöning, 1989, Chapter 1.1]

Let  $\{A_1, A_2, \dots\}$  be an infinite set of *propositional variables*.

#### 2.1 Definition

An *atomic formula* is a propositional variable.

*Formulas* are defined by the following inductive process.

1. All atomic formulas are formulas.
2. For every formula  $F$ ,  $\neg F$  is a formula, called the *negation* of  $F$ .
3. For all formulas  $F$  and  $G$ , also  $(F \vee G)$  and  $(F \wedge G)$  are formulas, called the *disjunction* and the *conjunction* of  $F$  and  $G$ , respectively.

The symbols  $\neg$ ,  $\vee$ ,  $\wedge$  are called *connectives*.  $\neg$  is a *unary* connective, while  $\vee$  and  $\wedge$  are *binary* connectives.

If a formula  $F$  occurs in another formula  $G$ , then it is called a *subformula* of  $G$ . Note that every formula is a subformula of itself.

#### 2.2 Notation

We use the following abbreviations:

$A, B, C, \dots$  instead of  $A_1, A_2, \dots$  and other obvious variants.

[Be careful with the use of  $F$  and  $G$ !]

We sometimes omit brackets if it can be done safely. [Be careful with this!]

$(F \rightarrow G)$  instead of  $(\neg F \vee G)$

$(F \leftrightarrow G)$  instead of  $(F \rightarrow G) \wedge (G \rightarrow F)$

$\rightarrow$  and  $\leftrightarrow$  are also called connectives.

$(\bigvee_{i=1}^n F_i)$  instead of  $(F_1 \vee F_2 \vee \dots \vee F_n)$

$(\bigwedge_{i=1}^n F_i)$  instead of  $(F_1 \wedge F_2 \wedge \dots \wedge F_n)$

#### 2.3 Example

$(\neg B \rightarrow F)$  is  $(\neg \neg B \vee F)$ .

Some Subformulas:  $\neg \neg B$ ,  $\neg B$ .

#### 2.4 Example

$((I \vee \neg B) \rightarrow \neg F)$  is  $(\neg(I \vee \neg B) \vee \neg F)$ .

Some Subformulas:  $\neg(I \vee \neg B)$ ,  $I$ ,  $\neg B$ .

**Exercise 27** Determine all subformulas of  $((B \wedge F) \rightarrow \neg I)$ .

#### 2.5 Remark

Formulas can be represented in a unique way as *trees*. [Example 2.4 on whiteboard.]

**Exercise 28** Draw the formulas from Example 2.3 and Exercise 27 as trees.

## 2.B Semantics

[Schöning, 1989, Chapter 1.1 cont.]

### 2.6 Definition

$\mathbb{T} = \{0, 1\}$  – the set of *truth values*: *false*, and *true*, respectively.

An *assignment* is a function  $\mathcal{A} : \mathbf{D} \rightarrow \mathbb{T}$ , where  $\mathbf{D}$  is a set of atomic formulas.

Given such an assignment  $\mathcal{A}$ , we extend it to  $\mathcal{A}' : \mathbf{E} \rightarrow \mathbb{T}$ , where  $\mathbf{E}$  is the set of all formulas containing only elements from  $\mathbf{D}$  as atomic subformulas:

1.  $\mathcal{A}'(A_i) = \mathcal{A}(A_i)$  for each  $A_i \in \mathbf{D}$
2.  $\mathcal{A}'(F \wedge G) = \begin{cases} 1, & \text{if } \mathcal{A}'(F) = 1 \text{ and } \mathcal{A}'(G) = 1 \\ 0, & \text{otherwise} \end{cases}$
3.  $\mathcal{A}'(F \vee G) = \begin{cases} 1, & \text{if } \mathcal{A}'(F) = 1 \text{ or } \mathcal{A}'(G) = 1 \\ 0, & \text{otherwise} \end{cases}$
4.  $\mathcal{A}'(\neg F) = \begin{cases} 1, & \text{if } \mathcal{A}'(F) = 0 \\ 0, & \text{otherwise} \end{cases}$

[From now on, drop distinction between  $\mathcal{A}$  and  $\mathcal{A}'$ .]

### 2.7 Example

Let  $\mathcal{A}(B) = \mathcal{A}(F) = 1$  and  $\mathcal{A}(I) = 0$ .

$$\begin{aligned} \mathcal{A}(\neg(B \wedge F) \vee \neg I) &= \begin{cases} 1, & \text{if } \mathcal{A}(\neg(B \wedge F)) = 1 \text{ or } \mathcal{A}(\neg I) = 1 \\ 0, & \text{otherwise} \end{cases} \\ &= \begin{cases} 1, & \text{if } \mathcal{A}(B \wedge F) = 0 \text{ or } \mathcal{A}(I) = 0 \\ 0, & \text{otherwise} \end{cases} \\ &= \begin{cases} 1, & \text{if } \mathcal{A}(B) = 0 \text{ or } \mathcal{A}(F) = 0 \text{ or } \mathcal{A}(I) = 0 \\ 0, & \text{otherwise} \end{cases} \\ &= 1 \end{aligned}$$

**Exercise 29 (hand-in)** Do the calculation from Example 2.7 for the formula  $\neg(I \vee \neg B) \vee \neg F$  from Example 2.4 and the values  $\mathcal{A}(I) = 1$  and  $\mathcal{A}(B) = \mathcal{A}(F) = 0$ .

### 2.8 Remark

The same thing can be expressed via *truth tables*.

$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}(F \wedge G)$
0	0	0
0	1	0
1	0	0
1	1	1

$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}(F \vee G)$
0	0	0
0	1	1
1	0	1
1	1	1

$\mathcal{A}(F)$	$\mathcal{A}(\neg F)$
0	1
1	0

### 2.9 Example

Determining the truth values of formulas using truth tables:

[Use the tree structure of formulas.]

$\mathcal{A}(B)$	$\mathcal{A}(F)$	$\mathcal{A}(I)$	$\mathcal{A}(B \wedge F)$	$\mathcal{A}(\neg(B \wedge F))$	$\mathcal{A}(\neg I)$	$\mathcal{A}(\neg(B \wedge F) \vee \neg I)$
0	0	0	0	1	1	1
0	0	1	0	1	0	1
0	1	0	0	1	1	1
0	1	1	0	1	0	1
1	0	0	0	1	1	1
1	0	1	0	1	0	1
1	1	0	1	0	1	1
1	1	1	1	0	0	0

### 2.10 Remark

The truth value of a formula is uniquely determined by the truth values of the propositional variables it contains as subformulas.

**Exercise 30 (hand-in)** Make the truth table for the formula from Exercise 29.

### 2.11 Remark

$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}(F \rightarrow G)$
0	0	1
0	1	1
1	0	0
1	1	1

$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}(F \leftrightarrow G)$
0	0	1
0	1	0
1	0	0
1	1	1

### 2.12 Definition

$F$ , a formula,  $\mathcal{A}$ , an assignment.

$\mathcal{A}$  is *suitable* if it is defined for all atomic formulas occurring in  $F$ .

We write  $\mathcal{A} \models F$  if  $\mathcal{A}$  is suitable for  $F$  and  $\mathcal{A}(F) = 1$ . We say  $F$  *holds under*  $\mathcal{A}$  or  $\mathcal{A}$  *is a model for*  $F$ . Otherwise, we write  $\mathcal{A} \not\models F$ .

$F$  is *satisfiable* if  $F$  has at least one model. Otherwise, it is called *unsatisfiable* or *contradictory*.

A set  $\mathbf{M}$  of formulas is *satisfiable* if there is an assignment  $\mathcal{A}$  which is a model for each formula in  $\mathbf{M}$ . In this case,  $\mathcal{A}$  is called a *model* of  $\mathbf{M}$ , and we write  $\mathcal{A} \models \mathbf{M}$ . [Note the overloading of notation.]

$F$  is called *valid* or a *tautology* if every suitable assignment for  $F$  is a model for  $F$ . In this case we write  $\models F$ , and otherwise  $\not\models F$ .

### 2.13 Example

Examples of models for  $p \vee \neg q \vee \neg r$  are the following.  $\mathcal{A}_1$  with  $\mathcal{A}_1(p) = \mathcal{A}_1(q) = \mathcal{A}_1(r) = 1$ ;  $\mathcal{A}_2$  with  $\mathcal{A}_2(p) = 1$  and  $\mathcal{A}_2(q) = \mathcal{A}_2(r) = 0$ ;  $\mathcal{A}_3$  with  $\mathcal{A}_3(p) = \mathcal{A}_3(q) = 0$  and  $\mathcal{A}_3(r) = 1$ . You can find models by making the truth table for the formula: the assignments for which the truth value is 1 are models.

### 2.14 Example

For the formula  $(p \wedge \neg q) \vee \neg p$ , the assignment  $\mathcal{A}_1$  with  $\mathcal{A}_1(p) = \mathcal{A}_1(q) = 0$  is a model, because  $\mathcal{A}_1((p \wedge \neg q) \vee \neg p) = 1$ . The assignment  $\mathcal{A}_2$  with  $\mathcal{A}_2(p) = 0$  and  $\mathcal{A}_2(q) = 1$  is also a model, because  $\mathcal{A}_2((p \wedge \neg q) \vee \neg p) = 1$ . This can also be seen from the truth table for  $(p \wedge \neg q) \vee \neg p$ . The assignment  $\mathcal{A}_3$  which only assigns  $\mathcal{A}_3(p) = 0$  is not a model for the formula because it is not suitable for the formula.

**Exercise 31 (hand-in)** Give a model for  $\neg(p \wedge q) \vee \neg r$ .

### 2.15 Example

$A \vee \neg A$  is a tautology.

[This is established by the following truth table:

$\mathcal{A}(A)$	$\mathcal{A}(\neg A)$	$\mathcal{A}(A \vee \neg A)$
0	1	1
1	0	1

]

**Exercise 32 (hand-in)** Show the following.

1.  $A \wedge \neg A$  is unsatisfiable.
2.  $A \rightarrow \neg A$  is satisfiable.

### 2.16 Theorem

A formula  $F$  is a tautology if and only if  $\neg F$  is unsatisfiable.

**Proof:**  $F$  is a tautology

iff every suitable assignment for  $F$  is a model for  $F$

iff every suitable assignment for  $F$  (hence also for  $\neg F$ ) is not a model for  $\neg F$

iff  $\neg F$  does not have a model

iff  $\neg F$  is unsatisfiable

■

10/8/13

### 2.17 Definition

A formula  $G$  is a (*logical*) *consequence* of a set  $M = \{F_1, \dots, F_n\}$  of formulas if for every assignment  $\mathcal{A}$  which is suitable for  $G$  and for all elements of  $M$ , it follows that whenever  $\mathcal{A} \models F_i$  for all  $i = 1, \dots, n$ , then  $\mathcal{A} \models G$ .

If  $G$  is a logical consequence of  $M$ , we write  $M \models G$  and say  $M$  *entails*  $G$ . [Note the overloading of notation!]

### 2.18 Theorem

The following assertions are equivalent.

1.  $G$  is a consequence of  $\{F_1, \dots, F_n\}$ .
2.  $((\bigwedge_{i=1}^n F_i) \rightarrow G)$  is a tautology.
3.  $((\bigwedge_{i=1}^n F_i) \wedge \neg G)$  is unsatisfiable.

**Exercise 33 (hand-in)** Show that an assignment is a model for  $(\bigwedge_{i=1}^n F_i)$  if and only if it is a model for  $\{F_1, \dots, F_n\}$ .

**Exercise 34 (hand-in)** Prove that 1. and 2. of Theorem 2.18 are equivalent. [Hint: Use Exercise 33.]

### 2.19 Example

Using Theorem 2.18, we can determine logical consequences using truth tables.

E.g., *modus ponens*:  $\{P, P \rightarrow Q\} \models Q$ .

We have to show:  $(P \wedge (P \rightarrow Q)) \rightarrow Q$  is a tautology.

$\mathcal{A}(P)$	$\mathcal{A}(Q)$	$\mathcal{A}(P \rightarrow Q)$	$\mathcal{A}(P \wedge (P \rightarrow Q))$	$\mathcal{A}((P \wedge (P \rightarrow Q)) \rightarrow Q)$
0	0	1	0	1
0	1	1	0	1
1	0	0	0	1
1	1	1	1	1

**Exercise 35 (hand-in)** Express modus tollens, modus tollendo ponens, and modus ponendo tollens in propositional logic.

**Exercise 36 (hand-in)** Show, using truth tables, that the modi from Exercise 35 are valid.

## 2.C Datalog Revisited: Semantics By Grounding

We can relate Datalog and propositional logic as follows.

### 2.20 Definition

Given a Datalog language  $L$ , we can define a set of propositional variables as follows. For every ground atom  $p(v_1, \dots, v_n)$  over  $L$ , let  $p_{v_1, \dots, v_n}$  be a propositional variable. Furthermore, let  $v$  be the function from ground atoms to propositional variables defined as

$$v(p(v_1, \dots, v_n)) = p_{v_1, \dots, v_n}.$$

Given a ground Datalog rule  $B_1 \wedge \dots \wedge B_k \rightarrow A$ , furthermore define

$$v(B_1 \wedge \dots \wedge B_k \rightarrow A) = v(B_1) \wedge \dots \wedge v(B_k) \rightarrow v(A)$$

if  $k \geq 1$ , and

$$v(\rightarrow A) = v(A)$$

(for facts).

### 2.21 Example

For the ground Datalog rule

$$p(a, b) \wedge q(c) \rightarrow p(a, c)$$

we have

$$v(p(a, b) \wedge q(c) \rightarrow p(a, c)) = v(p(a, b)) \wedge v(q(c)) \rightarrow v(p(a, c)) = p_{a,b} \wedge q_c \rightarrow p_{a,c}.$$

For the ground fact  $p(b, c)$  we have  $v(p(b, c)) = p_{b,c}$ .

### 2.22 Definition

Given a Datalog program  $P$ , define the associated set  $v(P)$  of propositional formulas as

$$v(P) = \{v(r) \mid r \in \text{ground}(P)\}.$$

### 2.23 Example

For the program  $P$  in Examples 1.12 and 1.20,  $v(P)$  consists of the following formulas.

$$\begin{aligned}
& \text{mOf}_{c,c} \rightarrow \text{pOf}_{c,c} \\
& \text{mOf}_{c,m} \rightarrow \text{pOf}_{c,m} \\
& \text{mOf}_{c,n} \rightarrow \text{pOf}_{c,n} \\
& \text{mOf}_{m,c} \rightarrow \text{pOf}_{m,c} \\
& \text{mOf}_{m,m} \rightarrow \text{pOf}_{m,m} \\
& \text{mOf}_{m,n} \rightarrow \text{pOf}_{m,n} \\
& \text{mOf}_{n,c} \rightarrow \text{pOf}_{n,c} \\
& \text{mOf}_{n,m} \rightarrow \text{pOf}_{n,m} \\
& \text{mOf}_{n,n} \rightarrow \text{pOf}_{n,n} \\
& \text{bOf}_{c,c} \wedge \text{pOf}_{c,c} \rightarrow \text{uOf}_{c,c} \\
& \text{bOf}_{c,m} \wedge \text{pOf}_{m,n} \rightarrow \text{uOf}_{c,m} \\
& \text{bOf}_{c,n} \wedge \text{pOf}_{n,m} \rightarrow \text{uOf}_{c,n} \\
& \vdots \quad \text{overall 27 groundings of this rule} \\
& \text{bOf}_{c,m} \\
& \text{mOf}_{m,n}
\end{aligned}$$

**Exercise 37 (hand-in)** For  $P$  the Datalog program from Exercise 9, determine  $v(P)$ .

### 2.24 Theorem

Let  $P$  be a Datalog program and  $A \in B_P$ . Then  $P \models_H A$  if and only if  $v(P) \models v(A)$ .

**Proof:** For every Herbrand interpretation  $I$  define an assignment  $\mathcal{A}_I$  by setting, for every ground atom  $B$ ,

$$\mathcal{A}_I(v(B)) = \begin{cases} 1 & \text{if } B \in I \\ 0 & \text{if } B \notin I. \end{cases}$$

Clearly, if  $I$  is a Herbrand model for  $P$ , then  $\mathcal{A}_I$  is a model for  $v(P)$ .

Now assume  $v(P) \models v(A)$ , and let  $M$  be a Herbrand model of  $P$ . Then  $\mathcal{A}_M$  is a model of  $v(P)$  and hence  $\mathcal{A}_M(v(A)) = 1$ . By definition of  $\mathcal{A}_M$  we obtain  $A \in M$  as required.

Conversely, for every assignment  $\mathcal{A}$ , define a Herbrand interpretation  $I_{\mathcal{A}}$  as

$$I_{\mathcal{A}} = \{B \mid \mathcal{A}(v(B)) = 1\}.$$

Clearly, if  $\mathcal{A}$  is a model for  $v(P)$  then  $I_{\mathcal{A}}$  is a Herbrand model for  $P$ .

Now assume  $P \models_H A$ , and let  $\mathcal{M}$  be model for  $v(P)$ . Then  $I_{\mathcal{M}}$  is a Herbrand model for  $P$  and hence  $A \in I_{\mathcal{M}}$ . By definition of  $I_{\mathcal{M}}$  we obtain  $\mathcal{M}(v(A)) = 1$  as required. ■

### Remark:

Theorem 2.24 shows how the problem of determining logical consequences for Datalog can be *reduced* (transformed) to the problem of determining logical consequences for propositional logic.

However, the theorem does not work in the other direction. E.g., in propositional logic we have  $\{\neg p, p \vee q\} \models q$  and  $\{p \rightarrow q, q \rightarrow r\} \models p \rightarrow r$ , and neither of these can be transformed into a Datalog problem based on the theorem.

## 2.D Equivalence

[Schöning, 1989, Chapter 1.2]

### 2.25 Definition

Formulas  $F$  and  $G$  are (*semantically*) *equivalent* (written  $F \equiv G$ ) if for every assignment  $\mathcal{A}$  that is suitable for  $F$  and  $G$ ,  $\mathcal{A}(F) = \mathcal{A}(G)$ .

### 2.26 Example

$A \vee B \equiv B \vee A$ . (*commutativity* of  $\vee$ )

[

$\mathcal{A}(A)$	$\mathcal{A}(B)$	$\mathcal{A}(A \vee B)$	$\mathcal{A}(B \vee A)$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

]

$A \vee \neg A \equiv B \vee \neg B$ . [truth table]

### 2.27 Example

$F \equiv G$  iff  $\models (F \leftrightarrow G)$ . [truth table]



## 2.28 Theorem

The following hold for all formulas  $F$ ,  $G$ , and  $H$ .

$F \wedge F \equiv F$	$F \vee F \equiv F$	Idempotency
$F \wedge G \equiv G \wedge F$	$F \vee G \equiv G \vee F$	Commutativity
$(F \wedge G) \wedge H \equiv F \wedge (G \wedge H)$	$(F \vee G) \vee H \equiv F \vee (G \vee H)$	Associativity
$F \wedge (G \vee H) \equiv (F \wedge G) \vee (F \wedge H)$	$F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$	Distributivity
$\neg\neg F \equiv F$	Double Negation	(also Involution)
$\neg(F \wedge G) \equiv \neg F \vee \neg G$	$\neg(F \vee G) \equiv \neg F \wedge \neg G$	de Morgan's Laws

**Proof:** Straightforward using truth tables. ■

**Exercise 38 (hand-in)** Prove that 2. and 3. of Theorem 2.18 are equivalent.

**Exercise 39 (hand-in)** Translate the “secrets” of the centenarian (slide 15 of slideset 1) into formulas, where  $B$  stands for *beer for dinner*,  $F$  for *fish for dinner* and  $I$  for *ice cream for dinner*.

**Exercise 40 (hand-in)** Show that the claim on slide 15 of slideset 1 holds.

## 2.29 Remark

Disjunction is dispensable. [ $F \vee G \equiv \neg(\neg F \wedge \neg G)$ ]

Alternatively, conjunction is dispensable. [ $F \wedge G \equiv \neg(\neg F \vee \neg G)$ ]

## 2.30 Remark

Let  $F \uparrow G = \neg(F \wedge G)$ .

$\neg F \equiv \neg(F \wedge F) \equiv F \uparrow F$ .

$F \vee G \equiv \neg(\neg F \wedge \neg G) \equiv \neg F \uparrow \neg G \equiv (F \uparrow F) \uparrow (G \uparrow G)$

$F \wedge G \equiv \neg\neg(F \wedge G) \equiv \neg(F \uparrow G) \equiv (F \uparrow G) \uparrow (F \uparrow G)$ .

## 2.31 Remark (The contraposition principle)

$\{F\} \models G$  iff  $\{\neg G\} \models \neg F$ .

$\{F\} \models G$  iff  $F \rightarrow G$  is a tautology (Theorem 2.18).

$F \rightarrow G \equiv \neg F \vee G \equiv \neg(\neg G) \vee (\neg F) \equiv (\neg G) \rightarrow (\neg F)$ .

$(\neg G) \rightarrow (\neg F)$  is a tautology iff  $\{\neg G\} \models \neg F$  (Theorem 2.18)]

10/10/13

## 2.E Normal Forms

[Schöning, 1989, Chapter 1.2 cont.]

### 2.32 Definition

A *literal* is an atomic formula (a *positive literal*) or the negation of an atomic formula (a *negative literal*).

A formula  $F$  is in *negation normal form* (NNF) if it is made up only of literals,  $\vee$ , and  $\wedge$  (and brackets).

### 2.33 Theorem

For every formula  $F$ , there is a formula  $G \equiv F$  which is in NNF.

**Proof:** The proof of Theorem 2.36 below shows this as well. ■

### 2.34 Example

$$(\neg(I \vee \neg B) \vee \neg F) \equiv (\neg I \wedge B) \vee \neg F$$

### 2.35 Definition

A formula  $F$  is in *conjunctive normal form* (CNF) if it is a conjunction of disjunctions of literals, i.e., if

$$F = \left( \bigwedge_{i=1}^n \left( \bigvee_{j=1}^{m_i} L_{i,j} \right) \right),$$

where the  $L_{i,j}$  are literals.

A formula  $F$  is in *disjunctive normal form* (DNF) if it is a disjunction of conjunctions of literals, i.e., if

$$F = \left( \bigvee_{i=1}^n \left( \bigwedge_{j=1}^{m_i} L_{i,j} \right) \right),$$

where the  $L_{i,j}$  are literals.

### 2.36 Theorem

For every formula  $F$  there is a formula  $F_1 \equiv F$  in CNF and a formula  $F_2 \equiv F$  in DNF.

**Proof:** Proof by structural induction.

*Induction base:* If  $F$  is atomic, then it is already in CNF and in DNF.

*Induction hypothesis:*  $G$  has CNF  $G_1$  and DNF  $G_2$ ,  $H$  has CNF  $H_1$  and DNF  $H_2$ . *Induction step:* We have 3 cases.

Case 1:  $F$  has the form  $F = \neg G$ .

Then

$$F \equiv \neg G_1 \equiv \neg \left( \bigwedge_{i=1}^n \left( \bigvee_{j=1}^{m_i} L_{i,j} \right) \right) \equiv \left( \bigvee_{i=1}^n \left( \bigwedge_{j=1}^{m_i} \neg L_{i,j} \right) \right) \equiv \left( \bigvee_{i=1}^n \left( \bigwedge_{j=1}^{m_i} \overline{L_{i,j}} \right) \right),$$

where

$$\overline{L_{i,j}} = \begin{cases} A & \text{if } L_{i,j} = \neg A \\ \neg A & \text{if } L_{i,j} = A \end{cases}$$

and the latter formula is in DNF as required. Analogously, we can obtain from  $G_2$  a CNF formula equivalent to  $F$ .

Case 2:  $F$  has the form  $F = G \vee H$ .

Then  $F \equiv G_2 \vee H_2$ , which is in DNF.

Further,

$$F \equiv G_1 \vee H_1 \equiv \left( \bigwedge_{i=1}^n \left( \bigvee_{j=1}^{m_i} K_{i,j} \right) \right) \vee \left( \bigwedge_{k=1}^o \left( \bigvee_{l=1}^{p_k} L_{k,l} \right) \right) \equiv \left( \bigwedge_{i=1}^n \left( \bigwedge_{k=1}^o \left( \bigvee_{j=1}^{m_i} K_{i,j} \vee \bigvee_{l=1}^{p_k} L_{k,l} \right) \right) \right),$$

which is in CNF.

Case 3:  $F$  has the form  $F = G \wedge H$ .

This case is analogous to Case 2. ■

**Exercise 41 (hand-in)** Show by structural induction: For any formula  $F$  (with all brackets written), we have  $b(F) \leq c(F)$ , where  $b(F)$  is the number of all opening brackets in  $F$ , and  $c(F)$  is the number of all connectives in  $F$ .

### 2.37 Remark

Structural induction is a fundamental proof technique, comparable with natural induction.

**Exercise 42 (hand-in)** Transform  $\neg((A \vee B) \wedge (C \vee D) \wedge (E \vee F))$  into CNF.

10/15/13

### 2.38 Remark

DNF via truth table.

If, e.g.,

$\mathcal{A}(A)$	$\mathcal{A}(B)$	$\mathcal{A}(C)$	$\mathcal{A}(F)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

then a DNF for  $F$  is  $(\neg A \wedge \neg B \wedge \neg C) \vee (A \wedge \neg B \wedge \neg C) \vee (A \wedge \neg B \wedge C)$ .

**Exercise 43 (hand-in)** Give a CNF for the formula  $F$  in Remark 2.38.

### 2.39 Definition

Two formulas  $F$  and  $G$  are *equisatisfiable* if the following holds:  $F$  has a model if and only if  $G$  has a model.

**Exercise 44 (hand-in)** Show the following: For all formulas  $F_i$  ( $i = 1, 2, 3$ ),  $F_1 \vee (F_2 \wedge F_3)$  and  $(F_1 \vee E) \wedge (E \leftrightarrow (F_2 \wedge F_3))$  are equisatisfiable ( $E$  is a propositional variable not occurring in  $F_1, F_2, F_3$ ).

## 2.F Tableaux Algorithm

[Ben-Ari, 1993, Chapter 2.6, strongly modified]

Translating truth tables directly into an algorithm is very expensive.

We take the following approach:

For showing  $F_1, \dots, F_n \models G$ , it suffices to show that  $F = F_1 \wedge \dots \wedge F_n \wedge \neg G$  is unsatisfiable (Theorem 2.18).

We attempt to construct a model for  $F$  in such a way that, if and only if the construction fails, we know that  $F$  is unsatisfiable.

### 2.40 Definition

Let  $F$  be a formula in NNF. A *tableau branch* for  $F$  is a set of formulas, defined inductively as follows.

- $\{F\}$  is a tableau branch for  $F$ .
- If  $T$  is a tableau branch for  $F$  and  $G \wedge H \in T$ , then  $T \cup \{G, H\}$  is a tableau branch for  $F$ .
- If  $T$  is a tableau branch for  $F$  and  $G \vee H \in T$ , then  $T \cup \{G\}$  is a tableau branch for  $F$  and  $T \cup \{H\}$  is a tableau branch for  $F$ .

A *tableau* for  $F$  is a set of tableau branches for  $F$ .

A tableau branch is *closed* if it contains an atomic formula  $A$  and the literal  $\neg A$ . Otherwise, it is *open*.

A tableau branch  $T$  is called *complete* if it satisfies the following conditions.

- $T$  is open.
- If  $G \wedge H \in T$ , then  $\{G, H\} \subseteq T$ .
- If  $G \vee H \in T$ , then  $G \in T$  or  $H \in T$ .

A tableau  $M$  for  $F$  is called *complete* if it satisfies the following conditions.

- If  $G \vee H \in T \in M$ , and  $T$  is open, then there are branches  $S_1 \in M$  and  $S_2 \in M$  with  $\{G\} \cup T \subseteq S_1$  and  $\{H\} \cup T \subseteq S_2$ .
- All branches of  $M$  are complete or closed.

A tableau is *closed* if it is complete and all its branches are closed.

10/17/13

If  $F$  is not in NNF, then a tableau (resp., tableau branch) for  $F$  is a tableau (resp. tableau branch) for an NNF of  $F$ .

### 2.41 Example

Consider  $(\neg I \wedge B) \vee \neg F$ , for which a complete (but not closed) tableau is  $\{ \{(\neg I \wedge B) \vee \neg F, \neg I \wedge B, \neg I, B\}, \{(\neg I \wedge B) \vee \neg F, \neg F\} \}$ .

**Exercise 45 (hand-in)** Give a complete tableau (as set of sets of formulas) for  $(\neg A \wedge \neg B \wedge \neg C) \vee (A \wedge \neg B \wedge \neg C)$ .

### 2.42 Remark

Tableaux can be represented graphically (blackboard).

### 2.43 Theorem (Soundness)

A formula  $F$  is satisfiable if there is a complete tableau branch for  $F$ .

### 2.44 Theorem (Completeness)

If a formula  $F$  is satisfiable, then there is a complete tableau branch for  $F$ .

### 2.45 Theorem

A formula  $F$  is

1. unsatisfiable if and only if there is a closed tableau for  $F$ ,

2. a tautology if and only if there is a closed tableau for  $\neg F$ .

### 2.46 Example

Modus Ponens holds if  $(P \wedge (P \rightarrow Q)) \rightarrow Q$  is a tautology. We construct a complete tableau (blackboard) for  $\neg((P \wedge (P \rightarrow Q)) \rightarrow Q)$ , which turns out to be closed.

**Exercise 46 (hand-in)** Do the same as in Example 2.46 for Modus Tollens.

**Exercise 47 (hand-in)** Show  $\{A \rightarrow (B \rightarrow C)\} \models (A \rightarrow B) \rightarrow (A \rightarrow C)$  using the tableaux algorithm.

### 2.47 Lemma

Let  $F$  be a formula,  $T$  be a complete tableau branch for  $F$ , and  $L_1, \dots, L_n$  be all the literals contained in  $T$ . Then any assignment  $\mathcal{A}$  with  $\mathcal{A}(L_1 \wedge \dots \wedge L_n) = 1$  is a model for  $F$ .

**Proof:** We show by structural induction, that  $\mathcal{A}$  is a model for each formula  $F'$  in  $T$ .

Induction Base: Let  $F' = L$  be a literal. Then by definition  $\mathcal{A}(F') = 1$ .

Induction Hypothesis:  $\mathcal{A}(G) = \mathcal{A}(H) = 1$  for  $G, H \in T$ .

Induction Step: (1) Let  $F' = G \wedge H \in T$ . Then  $G \in T$  and  $H \in T$ . By IH,  $\mathcal{A}(F') = \mathcal{A}(G \wedge H) = 1$ . (2) Let  $F' = G \vee H$ . Then  $G \in T$  or  $H \in T$ . By IH,  $\mathcal{A}(G) = 1$  or  $\mathcal{A}(H) = 1$ , hence  $\mathcal{A}(F') = 1$ . (3) The case  $F' = \neg G \in T$  cannot happen since all formulas are in NNF, and the literal case was dealt with in the induction base. ■

**Proof of Theorem 2.43:** By Lemma 2.47, we obtain that  $F$  has a model, hence it is satisfiable. ■

### 2.48 Example

Is the following formula valid? satisfiable? unsatisfiable?

$$(((A \rightarrow B) \rightarrow A) \rightarrow A)$$

(done on whiteboard)

### 2.49 Example

Is the following formula valid? satisfiable? unsatisfiable?

$$(A \rightarrow (B \rightarrow C)) \rightarrow ((A \wedge B) \rightarrow C)$$

(done on whiteboard)

**Proof of Theorem 2.44:** First note the following, for any assignment  $M$  and all formulas  $G$  and  $H$ :

- If  $M \models G \wedge H$ , then  $M \models G$  and  $M \models H$ .
- if  $M \models G \vee H$ , then  $M \models G$  or  $M \models H$ .

Since  $F$  is satisfiable, it has a model  $M$ . Construct a tableau branch  $T$  for  $F$  recursively as follows.

- If  $G \wedge H \in T$ , set  $T := T \cup \{G, H\}$ .

- If  $G \vee H \in T$  with  $M \models G$ , set  $T := T \cup \{G\}$ , otherwise set  $T := T \cup \{H\}$ .

The recursion terminates since only subformulas of  $F$  are added and sets cannot contain duplicate elements. The resulting  $T$  is a complete tableau branch, and  $M \models T$ , by definition. ■

**Proof of Theorem 2.45:**

We prove Statement 1. Statement 2 is shown in Exercise 48.

Let  $A$  be the statement “ $F$  is unsatisfiable”, and let  $B$  be the statement “ $F$  has a closed tableau”.

We need to show:  $A \equiv B$ , for which it suffices to show that  $A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A)$  is valid.

By the contraposition principle, it therefore suffices to show that  $(\neg B \rightarrow \neg A) \wedge (\neg A \rightarrow \neg B) \equiv (\neg B \leftrightarrow \neg A)$  is valid, i.e., that  $\neg A \equiv \neg B$ .

$\neg A$  is the statement “ $F$  is not unsatisfiable”, i.e. “ $F$  is satisfiable”.

$\neg B$  is the statment “ $F$  does not have a closed tableau”. Since, every formula has a complete tableau, this is equivalent to the statement “ $F$  has a complete tableau branch”.

It thus remains to show:  *$F$  is satisfiable if and only if  $F$  has a complete tableau branch.* This was shown in Theorems 2.43 and 2.44. ■

**2.50 Remark**

In short, Statement 1 of Theorem 2.45 holds because it expresses the contrapositions of Theorem 2.43 and 2.44.

**Exercise 48 (hand-in)** Show Theorem 2.45 part 2.

**Exercise 49 (hand-in)** For any formula  $F$ , let  $F'$  be the formula obtained from  $F$  by replacing all  $\vee$  by  $\wedge$ , and by replacing all  $\wedge$  by  $\vee$ . Furthermore, let  $\overline{F}$  be obtained from  $F$  by replacing each occurrence of an atomic formula  $A$  in  $F$  by  $\neg A$ .

Example: For  $F = (A \wedge B) \vee \neg C$ , we have  $F' = (A \vee B) \wedge \neg C$  and  $\overline{F} = (\neg A \wedge \neg B) \vee \neg \neg C$ ; and  $\overline{F'} = (\neg A \vee \neg B) \wedge \neg \neg C$ .

Show by structural induction:  $F \equiv \neg \overline{F'}$  for each formula  $F$ .

skipped

**Exercise 50 (no hand-in)** Is the following true or false?

*Let  $M, N$  be sets of formulas. If  $\{F \mid M \models F\} \subseteq \{F \mid N \models F\}$  then  $M \subseteq N$ .*

Prove that your answer is correct.

**Exercise 51 (no hand-in)** Show: A set  $M$  of formulas is unsatisfiable if and only if some finite subset of it is unsatisfiable.

**Exercise 52 (no hand-in)** Show:  $M \models F$  if and only if  $M \cup \{\neg F\}$  is unsatisfiable. (Proof by contradiction principle.)

**Exercise 53 (no hand-in)** Let  $\{F_1, F_2, F_3, \dots\}$  be a (countably) infinite set. Give an algorithm which enumerates all its finite subsets.

## 3 First-order Predicate Logic

### 3.1 Example

Difficult/impossible to model in propositional logic:

- For all  $n \in \mathbb{N}$ ,  $n! \geq n$ .

### 3.2 Example

Difficult/impossible to model in propositional logic:

1. Healthy beings are not dead.
2. Every cat is alive or dead.
3. If somebody owns something, (s)he cares for it.
4. A happy cat owner owns a cat and all beings he cares for are healthy.
5. Schrödinger is a happy cat owner.

## 3.A Syntax

[Schöning, 1989, Chapter 2.1]

### 3.3 Definition

- *Variables*:  $x_1, x_2, \dots$  (also  $y, z, \dots$ ).
- *Function symbols*:  $f_1, f_2, \dots$  (also  $g, h, \dots$ ), each with an *arity* ( $\in \mathbb{N}$ ) (number of parameters).  
*Constants* are function symbols with arity 0.
- *Predicate symbols*:  $P_1, P_2, \dots$  (also  $Q, R, \dots$ ), each with an *arity* ( $\in \mathbb{N}$ ) (number of parameters).

*Terms* are inductively defined:

- Each variable is a term.
- If  $f$  is a function symbol of arity  $k$ , and if  $t_1, \dots, t_k$  are terms, then  $f(t_1, \dots, t_k)$  is a term.

*Formulas* are inductively defined:

- If  $P$  is a predicate symbol of arity  $k$ , and if  $t_1, \dots, t_k$  are terms, then  $P(t_1, \dots, t_k)$  is a formula (called *atomic*).
- For each formula  $F$ ,  $\neg F$  is a formula.
- For all formulas  $F$  and  $G$ ,  $(F \wedge G)$  and  $(F \vee G)$  are formulas.
- If  $x$  is a variable and  $F$  is a formula, then  $\exists xF$  and  $\forall xF$  are formulas.

### 3.4 Definition

$F \rightarrow G$  (respectively,  $F \leftrightarrow G$ ) is shorthand for  $\neg F \vee G$  (respectively,  $(F \rightarrow G) \wedge (G \rightarrow F)$ ). We also use other notational variants from propositional logic freely.

### 3.5 Example

The following are formulas ( $s$  is a constant).

1.  $\forall x(H(x) \rightarrow \neg D(x))$
2.  $\forall x(C(x) \rightarrow (A(x) \vee D(x)))$
3.  $\forall x\forall y(O(x, y) \rightarrow R(x, y))$
4.  $\forall x(P(x) \rightarrow (\exists y(O(x, y) \wedge C(y)) \wedge (\forall y(R(x, y) \rightarrow H(y))))))$
5.  $P(s)$

In 1, predicate symbols are  $D$  and  $H$ , and  $x$  is a term.

**Exercise 54 (hand-in)** Identify all predicate symbols and all terms in Example 3.5 3.

### 3.6 Example

Example 3.1 could be written as

$$\forall n(n \in \mathbb{N} \rightarrow n! \geq n),$$

where (with abuse of our introduced formal notation), “ $\in \mathbb{N}$ ” is a unary predicate symbol, “ $\geq$ ” is a binary predicate symbol, and “!” is a unary function symbol, written postfix.

**Exercise 55 (hand-in)** Determine all predicate symbols and all function symbols, with arities, of the formula

$$\forall \varepsilon \exists \delta \forall x((\varepsilon > 0 \wedge \delta > 0) \rightarrow (|x - 2| < \delta \rightarrow |x^3 - 2^3| < \varepsilon)).$$

### 3.7 Definition

If a formula  $F$  is part of a formula  $G$ , then it is called a *subformula* of  $G$ .

An occurrence of a variable  $x$  in a formula  $F$  is *bound* if it occurs within a subformula of  $F$  of the form  $\exists xG$  or  $\forall xG$ . Otherwise it is *free*.

A formula without free variables is *closed*. A formula with free variables is *open*.

$\exists, \forall$  are *quantifiers*,  $\vee, \wedge, \neg, \rightarrow, \leftrightarrow$  are *connectives*.

### 3.8 Example

All subformulas of  $\forall x(C(x) \rightarrow (A(x) \vee D(x)))$ :

$C(x), A(x), D(x), A(x) \vee D(x), C(x) \rightarrow (A(x) \vee D(x)), \forall x(C(x) \rightarrow (A(x) \vee D(x)))$ .

### 3.9 Example

In the formula  $P(x) \wedge \forall x(P(x) \rightarrow Q(f(x)))$ , the first occurrence of  $x$  is free, the others are bound.

**Exercise 56 (hand-in)** Give all subformulas of Example 3.5 4. Which of them are closed? Which of them are open?

## 3.B Semantics

[Schöning, 1989, Chapter 2.1 cont.]

### 3.10 Definition

A *structure* is a pair  $\mathcal{A} = (U_{\mathcal{A}}, I_{\mathcal{A}})$ , with  $U_{\mathcal{A}} \neq \emptyset$  a set (*ground set* or *universe*) and  $I_{\mathcal{A}}$  a mapping which maps



- each  $k$ -ary predicate symbol  $P$  to a  $k$ -ary predicate (relation) on  $U_{\mathcal{A}}$  (if  $I_{\mathcal{A}}$  is defined for  $P$ )
- each  $k$ -ary function symbol  $f$  to a  $k$ -ary function on  $U_{\mathcal{A}}$  (if  $I_{\mathcal{A}}$  is defined for  $f$ )
- each variable  $x$  to an element of  $U_{\mathcal{A}}$  (if  $I_{\mathcal{A}}$  is defined for  $x$ ).

Write  $P^{\mathcal{A}}$  for  $I_{\mathcal{A}}(P)$  etc.  $\mathcal{A}$  is *suitable* for a formula  $F$  if  $I_{\mathcal{A}}$  is defined for all predicate and function symbols in  $F$  and for all free variables in  $F$ .

### 3.11 Example

$$F = \forall x \forall y (P(a) \wedge (P(x) \rightarrow (P(s(x)) \wedge Q(x, x) \wedge ((P(y) \wedge Q(x, y)) \rightarrow Q(x, s(y))))))$$

Structure  $(U_{\mathcal{A}}, I_{\mathcal{A}})$ :

$$\begin{aligned} U_{\mathcal{A}} &= \mathbb{N} \\ a^{\mathcal{A}} &= 0 (\in \mathbb{N}) \\ s^{\mathcal{A}} &: n \mapsto n + 1 \\ P^{\mathcal{A}} &= \mathbb{N} \quad (= U_{\mathcal{A}}) \\ Q^{\mathcal{A}} &= \{(n, k) \mid n \leq k\} \end{aligned}$$

Another structure  $(U_{\mathcal{B}}, I_{\mathcal{B}})$ :

$$\begin{aligned} U_{\mathcal{B}} &= \{\ominus, \odot\} \\ a^{\mathcal{B}} &= \ominus \\ s^{\mathcal{B}} &: \ominus \mapsto \odot; \odot \mapsto \odot \\ P^{\mathcal{B}} &= U_{\mathcal{B}} \\ Q^{\mathcal{B}} &= \{(\ominus, \odot)\} \end{aligned}$$

**Exercise 57 (hand-in)** Give a structure for the formula

$$\forall x \forall y (Q(x, y) \rightarrow Q(y, x)).$$

10/31/13

### 3.12 Definition

$F$  a formula.  $\mathcal{A} = (U_{\mathcal{A}}, I_{\mathcal{A}})$  a suitable structure for  $F$ .

Define for each term  $t$  in  $F$  its *value*  $t^{\mathcal{A}}$ :

1. If  $t = x$  is a variable,  $t^{\mathcal{A}} = x^{\mathcal{A}}$ .
2. If  $t = f(t_1, \dots, t_k)$ , then  $t^{\mathcal{A}} = f^{\mathcal{A}}(t_1^{\mathcal{A}}, \dots, t_k^{\mathcal{A}})$ .

Define for  $F$  its *truth value*  $\mathcal{A}(F)$  as follows, where  $\mathcal{A}_{[x/u]}$  is identical to  $\mathcal{A}$  except  $x^{\mathcal{A}_{[x/u]}} = u$ .

1.  $\mathcal{A}(P(t_1, \dots, t_k)) = \begin{cases} 1, & \text{if } (\mathcal{A}(t_1), \dots, \mathcal{A}(t_k)) \in P^{\mathcal{A}} \\ 0, & \text{otherwise} \end{cases}$
2.  $\mathcal{A}(H \wedge G) = \begin{cases} 1, & \text{if } \mathcal{A}(H) = 1 \text{ and } \mathcal{A}(G) = 1 \\ 0, & \text{otherwise} \end{cases}$

3.  $\mathcal{A}(H \vee G) = \begin{cases} 1, & \text{if } \mathcal{A}(H) = 1 \text{ or } \mathcal{A}(G) = 1 \\ 0, & \text{otherwise} \end{cases}$
4.  $\mathcal{A}(\neg G) = \begin{cases} 1, & \text{if } \mathcal{A}(G) = 0 \\ 0, & \text{otherwise} \end{cases}$
5.  $\mathcal{A}(\forall x G) = \begin{cases} 1, & \text{if for all } u \in U_{\mathcal{A}}, \mathcal{A}_{[x/u]}(G) = 1 \\ 0, & \text{otherwise} \end{cases}$
6.  $\mathcal{A}(\exists x G) = \begin{cases} 1, & \text{if there exists some } u \in U_{\mathcal{A}} \text{ s.t. } \mathcal{A}_{[x/u]}(G) = 1 \\ 0, & \text{otherwise} \end{cases}$

If  $\mathcal{A}(F) = 1$ , we write  $\mathcal{A} \models F$  and say  $F$  is true in  $\mathcal{A}$  or  $\mathcal{A}$  is a model for  $F$ .  $F$  is *valid* (or a *tautology*, written  $\models F$ ) if  $\mathcal{A} \models F$  for every suitable structure  $\mathcal{A}$  for  $F$ .  $F$  is *satisfiable* if there is  $\mathcal{A}$  with  $\mathcal{A} \models F$ , and otherwise it is *unsatisfiable*.

### 3.13 Remark

Many notions and results carry over directly from propositional logic: *logical consequence*, *equivalence of formulas*, Theorem 2.18, Theorem 2.28, etc. See Remark 3.19.

11/05/13

### 3.14 Example

Consider the sentences

James Potter is the parent of Harry Potter.  
 Harry Potter is an orphan.  
 Any parent of any orphan is dead.

They can be represented formally as follows.

$$\text{parentOf}(\text{jamespotter}, \text{harrypotter}) \tag{22}$$

$$\wedge \text{orphan}(\text{harrypotter}) \tag{23}$$

$$\wedge \forall x \forall y (\text{orphan}(x) \wedge \text{parentOf}(y, x) \rightarrow \text{dead}(y)) \tag{24}$$

This has

$$\text{dead}(\text{jamespotter})$$

as logical consequence.

Proof sketch: From lines (22) and (23) we can conclude by the rule in (24) with  $x = \text{harrypotter}$  and  $y = \text{jamespotter}$  that  $\text{dead}(\text{harrypotter})$ .

Before we go for a formal proof, let's first give some examples for signatures—see Table 1.

Now for a formal proof: Let  $\mathcal{A}$  be any model for the formula in (22)-(24). From (22) we then obtain

$$(\text{jamespotter}^{\mathcal{A}}, \text{harrypotter}^{\mathcal{A}}) \in \text{parentOf}^{\mathcal{A}}.$$

From (23) we obtain

$$\text{harrypotter}^{\mathcal{A}} \in \text{orphan}^{\mathcal{A}}.$$

$\mathcal{U}_{\mathcal{A}}$	$\{j, h\}$	$\mathbb{N}$	$\mathbb{N}$	$\{a\}$	$\{j, h\}$	$\{j, h\}$
$\text{harrypotter}^{\mathcal{A}}$	$h$	1	1	$a$	$h$	$h$
$\text{jamespotter}^{\mathcal{A}}$	$j$	2	2	$a$	$j$	$j$
$\text{orphan}^{\mathcal{A}}$	$\{h\}$	$\{1, 3, 4\}$	$\{3, 4, 5\}$	$\{a\}$	$\{h\}$	$\{h\}$
$\text{parentOf}^{\mathcal{A}}$	$\{(j, h)\}$	$\{(2, 1)\}$	$\{(1, 2), (3, 1)\}$	$\{(a, a)\}$	$\{(h, j)\}$	$\{(j, h)\}$
$\text{dead}^{\mathcal{A}}$	$\{j\}$	$\{1, 2\}$	$\{1, 3, 4\}$	$\{a\}$	$\emptyset$	$\{h\}$
	model	model	no model	model	no model	no model

Table 1: Signatures for Example 3.14.

From (24) we obtain that, whenever

$$u \in \text{orphan}^{\mathcal{A}} \quad \text{and} \quad (v, u) \in \text{parentOf}^{\mathcal{A}},$$

then

$$v \in \text{dead}^{\mathcal{A}}.$$

So consequently

$$\text{jamespotter}^{\mathcal{A}} \in \text{dead}^{\mathcal{A}}.$$

Since this argument holds for all models  $\mathcal{A}$ , we have that

$$\text{dead}(\text{harrypotter})$$

is indeed a logical consequence.

### 3.15 Example

$$\begin{aligned} & \text{parentOf}(\text{fatherOf}(\text{harrypotter}), \text{harrypotter}) \\ & \wedge \text{orphan}(\text{harrypotter}) \\ & \wedge \forall x \forall y (\text{orphan}(x) \wedge \text{parentOf}(y, x) \rightarrow \text{dead}(y)) \end{aligned}$$

has

$$\text{dead}(\text{fatherOf}(\text{harrypotter}))$$

as logical consequence.

### 3.16 Example

$$\begin{aligned} & \text{human}(\text{harrypotter}) \wedge \text{orphan}(\text{harrypotter}) \\ & \wedge \forall x (\text{human}(x) \rightarrow \text{parentOf}(\text{fatherOf}(x), x)) \\ & \wedge \forall x \forall y (\text{orphan}(x) \wedge \text{parentOf}(y, x) \rightarrow \text{dead}(y)) \end{aligned}$$

has

$$\text{dead}(\text{fatherOf}(\text{harrypotter}))$$

as logical consequence.

### 3.17 Example

$$\begin{aligned} &\forall x(\text{human}(x) \rightarrow \text{parentOf}(\text{fatherOf}(x), x)) \\ &\quad \wedge \forall x \forall y(\text{orphan}(x) \wedge \text{parentOf}(y, x) \rightarrow \text{dead}(y)) \end{aligned}$$

has

$$\forall x(\text{human}(x) \wedge \text{orphan}(x) \rightarrow \text{dead}(\text{fatherOf}(x)))$$

as logical consequence.

**Exercise 58 (hand-in)** Give two structures for the first formula in Example 3.17, one of which is a model for the formula, and one of which is not a model for the formula.

### 3.18 Example

Consider the formula  $F = \exists x \forall y Q(x, y)$  under the structure  $\mathcal{A} = (U_{\mathcal{A}}, I_{\mathcal{A}})$  from Example 3.11. We show  $\mathcal{A}(F) = 1$ .

First note that  $0 \leq n$  for all  $n \in \mathbb{N}$ , i.e.  $\mathcal{A}_{[x/0][y/n]}(Q(x, y)) = 1$  for all  $n \in \mathbb{N} = U_{\mathcal{A}}$ . Thus,  $\mathcal{A}_{[x/0]}(\forall y Q(x, y)) = 1$  and therefore  $\mathcal{A}(\exists x \forall y Q(x, y)) = 1$  as desired.

**Exercise 59 (hand-in)** Show that  $(U_{\mathcal{B}}, I_{\mathcal{B}})$  as in Example 3.11 is a model for

$$\forall x \exists y (P(x) \wedge Q(s(x), y)).$$

### 3.19 Remark

Predicate logic “degenerates” to propositional logic if either all predicate symbols have arity 0, or if no variables are used. For the latter, a formula like  $(Q(a) \wedge \neg R(f(b), c)) \wedge P(a, b)$  can be written as the propositional formula  $(A \wedge \neg B) \wedge C$  with  $A$  for  $Q(a)$ ,  $B$  for  $R(f(b), c)$ , and  $C$  for  $P(a, b)$ .

### 3.20 Remark

We deal with *first-order* predicate logic. Second-order predicate logic also allows to quantify over predicate symbols.

**Exercise 60 (hand-in)** Sentence 1 of Example 3.2 can be written as.

$$\forall x(\text{Healthy}(x) \rightarrow \neg \text{Dead}(x)).$$

Translate all other sentences from Example 3.2. Use *schroedinger* as a constant symbol and use only the following predicate symbols:

unary: *Healthy*, *Dead*, *Cat*, *Alive*, *HappyCatOwner*

binary: *owns*, *cares*

**Exercise 61 (hand-in)** Sketch, how you would formally prove, using Exercise 60, that Schrödinger’s cat is alive.

### 3.C Datalog Revisited

#### 3.21 Definition

Given a Datalog rule  $r = B_1 \wedge \dots \wedge B_k \rightarrow A$ , let  $\pi(r)$  be the formula

$$\forall x_1 \dots \forall x_n (B_1 \wedge \dots \wedge B_k \rightarrow A)$$

where  $x_1, \dots, x_n$  are (all) the variables occurring in  $r$ .

Given a Datalog program  $P$ , let  $\pi(P) = \{\pi(r) \mid r \in P\}$ .

#### 3.22 Theorem

Let  $P$  be any Datalog program and let  $A \in B_P$ . Then  $P \models_H A$  if and only if  $\pi(P) \models A$ .

**Exercise 62 (hand-in: Due Nov 14)** Prove Theorem 3.22. (Hint: seek guidance from the proof of Theorem 2.24.)

11/14/13

### 3.D Equivalence

[Schöning, 1989, Chapter 2.2]

#### 3.23 Theorem

The following hold for arbitrary formulas  $F$  and  $G$ .

$$\begin{array}{ll} \neg \forall x F \equiv \exists x \neg F & \neg \exists x F \equiv \forall x \neg F \\ \forall x F \wedge \forall x G \equiv \forall x (F \wedge G) & \exists x F \vee \exists x G \equiv \exists x (F \vee G) \\ \forall x \forall y F \equiv \forall y \forall x F & \exists x \exists y F \equiv \exists y \exists x F \end{array}$$

If  $x$  does not occur free in  $G$ , then

$$\begin{array}{ll} \forall x F \wedge G \equiv \forall x (F \wedge G) & \forall x F \vee G \equiv \forall x (F \vee G) \\ \exists x F \wedge G \equiv \exists x (F \wedge G) & \exists x F \vee G \equiv \exists x (F \vee G) \end{array}$$

**Proof:** We show only  $\forall x F \wedge \forall x G \equiv \forall x (F \wedge G)$ :

$$\mathcal{A}(\forall x F \wedge \forall x G) = 1$$

$$\text{iff } \mathcal{A}(\forall x F) = 1 \text{ and } \mathcal{A}(\forall x G) = 1$$

$$\text{iff for all } u \in U_{\mathcal{A}}, \mathcal{A}_{[x/u]}(F) = 1 \text{ and for all } v \in U_{\mathcal{A}}, \mathcal{A}_{[x/v]}(G) = 1$$

$$\text{iff for all } u \in U_{\mathcal{A}}, \mathcal{A}_{[x/u]}(F) = 1 \text{ and } \mathcal{A}_{[x/u]}(G) = 1$$

$$\text{iff } \mathcal{A}(\forall x (F \wedge G)) = 1 \quad \blacksquare$$

**Exercise 63 (hand-in)** Show, without using any of the statements in Theorem 3.23, that the first statement,  $\neg \forall x F \equiv \exists x \neg F$ , holds.

**Exercise 64 (hand-in)** Show, that  $\forall x \exists y P(x, y) \not\equiv \exists u \forall v P(v, u)$ .

**Exercise 65 (hand-in)** Show, using the statements from Theorem 3.23, that  $\forall x \exists y (P(x) \wedge Q(y)) \equiv \exists y \forall x (P(x) \wedge Q(y))$ .

**Exercise 66 (hand-in)** Show by using the statements from of Theorem 3.23, that

$$\forall x (P(x) \rightarrow (\exists y (O(x, y) \wedge C(y)) \wedge (\forall z (R(x, z) \rightarrow H(z))))))$$

and

$$\forall z \forall x \exists y ((P(x) \rightarrow (O(x, y) \wedge C(y))) \wedge ((P(x) \wedge R(x, z)) \rightarrow H(z)))$$

are equivalent.

### 3.24 Definition

A *substitution*  $[x/t]$ , where  $x$  is a variable and  $t$  a term, is a mapping which maps each formula  $G$  to the formula  $G[x/t]$ , which is obtained from  $G$  by replacing all free occurrences of  $x$  by  $t$ .

### 3.25 Example

$$(P(x, y) \wedge \forall y Q(x, y))[x/a][y/f(x)] = P(a, f(x)) \wedge \forall y Q(a, y)$$

**Exercise 67 (hand-in)** What is  $(\forall x (Q(x, y, z)[y/a])[x/b] \wedge \forall x (P(x, y)[y/x][x/a]))[z/x]$ ?

**Exercise 68 (hand-in)** Show, that, for any formula  $F$  in which  $y$  does not occur as free variable,  $\forall x F \equiv \forall y F[x/y]$ .

## 3.E Normal Forms

[Schöning, 1989, Chapter 2.2 cont.]

### 3.26 Definition

A *literal* is an atomic formula (a *positive literal*) or the negation of an atomic formula (a *negative literal*).

A formula  $F$  is in *negation normal form* (NNF) if the negation symbol  $\neg$  occurs only in literals (and  $\rightarrow$ ,  $\leftrightarrow$  don't appear in it).

### 3.27 Theorem

For every formula  $F$ , there is a formula  $G \equiv F$  which is in NNF.

**Proof:** Apply de Morgan, double negation, and  $\neg \forall x F \equiv \exists x \neg F$  and  $\neg \exists x F \equiv \forall x \neg F$  exhaustively. ■

### 3.28 Example

$$\begin{aligned} & \neg(\exists x P(x, y) \vee \forall z Q(z)) \wedge \neg \exists w P(f(a, w)) \\ & \equiv (\neg \exists x P(x, y) \wedge \neg \forall z Q(z)) \wedge \forall w \neg P(f(a, w)) \\ & \equiv (\forall x \neg P(x, y) \wedge \exists z \neg Q(z)) \wedge \forall w \neg P(f(a, w)) \end{aligned}$$

**Exercise 69 (hand-in)** Transform all formulas from Example 3.5 into NNF.

## 3.F Tableaux Algorithm

[Ben-Ari, 1993, Chapter 5.5, strongly modified]

### 3.29 Definition

Let  $F$  be a formula in NNF. A *tableau branch* for  $F$  is a set of formulas, defined inductively as follows.

- $\{F\}$  is a tableau branch for  $F$ .
- If  $T$  is a tableau branch for  $F$  and  $G \wedge H \in T$ , then  $T \cup \{G, H\}$  is a tableau branch for  $F$ .
- If  $T$  is a tableau branch for  $F$  and  $G \vee H \in T$ , then  $T \cup \{G\}$  is a tableau branch for  $F$  and  $T \cup \{H\}$  is a tableau branch for  $F$ .
- If  $T$  is a tableau branch for  $F$  and  $\forall xG \in T$ , then  $T \cup \{G[x/t]\}$  is a tableau branch for  $F$ , where  $t$  is any term.
- If  $T$  is a tableau branch for  $F$  and  $\exists xG \in T$ , then  $T \cup \{G[x/a]\}$  is a tableau branch for  $F$ , where  $a$  is a constant symbol which does not occur in  $T$  (or in the tableau currently constructed).

A *tableau* for  $F$  is a set of tableau branches for  $F$ .

A tableau branch is *closed* if it contains an atomic formula  $A$  and its negation  $\neg A$ . Otherwise, it is *open*.

A tableau  $M$  for  $F$  is called *closed* if for each  $T \in M$  there is a closed  $T' \in M$  with  $T \subseteq T'$ .

If  $F$  is not in NNF, then a tableau (resp., tableau branch) for  $F$  is a tableau (resp. tableau branch) for an NNF of  $F$ .

11/19/13

### 3.30 Theorem (Soundness)

If a closed formula  $F$  has a closed tableau, then  $F$  is unsatisfiable.

### 3.31 Theorem (Completeness)

If a closed formula  $F$  is unsatisfiable, then there is a closed tableau for  $F$ .

### 3.32 Example

We show  $\exists u \forall v P(v, u) \models \forall x \exists y P(x, y)$ . I.e. we make a tableau for

$$\exists u \forall v P(v, u) \wedge \exists x \forall y \neg P(x, y),$$

see Figure 2 (left).

### 3.33 Example

We show, that

$$\exists x \exists y (P(x) \vee Q(y)) \models \exists x (P(x) \vee Q(x)).$$

[done on whiteboard]

### 3.34 Example

We show, that

$$\forall x \exists y (P(x) \wedge Q(y)) \equiv \exists y \forall x (P(x) \wedge Q(y)).$$

[done on whiteboard]

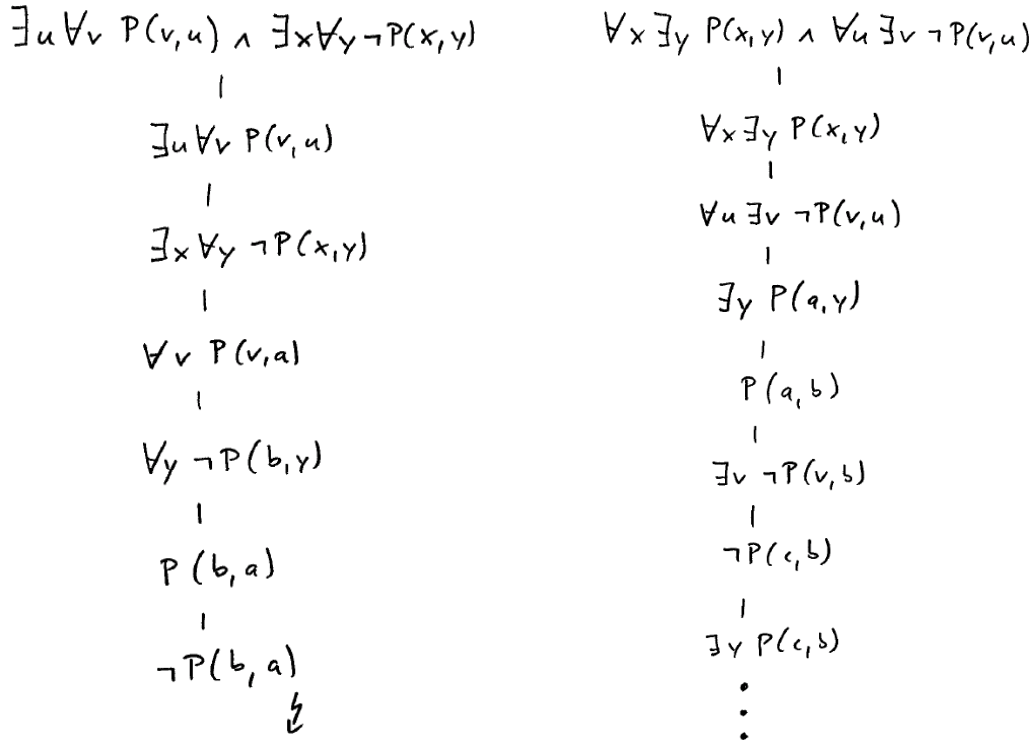


Figure 2: Tableaux for Example 3.32 (left) and Remark 3.36 (right).

### 3.35 Example

We show, that

$$\forall x(P(x) \rightarrow (\exists y(O(x, y) \wedge C(y)) \wedge (\forall z(R(x, z) \rightarrow H(z))))))$$

has

$$\forall z \forall x \exists y ((P(x) \rightarrow (O(x, y) \wedge C(y))) \wedge ((P(x) \wedge R(x, z)) \rightarrow H(z)))$$

as logical consequence.

[done on whiteboard]

11/21/13

### 3.36 Remark

The (predicate logic) tableaux algorithm does *not* in general provide a means to find out if a formula is satisfiable or falsifiable.

Consider  $\forall x \exists y P(x, y) \stackrel{?}{\models} \exists u \forall v P(v, u)$ . If we attempt to make a tableau for

$$\forall x \exists y P(x, y) \wedge \forall u \exists v \neg P(v, u),$$

see for example Figure 2, then the search for closing the tableau does not stop. The reason for this is that the tableau cannot close, but the occurrence of the quantifiers in the formula prompts the algorithm to ever explore new terms for the bound variables.



**Exercise 70 (hand-in, due 11/26)** Show, using a tableau, that  $\exists x(P(x) \wedge Q(x)) \models \exists xP(x) \wedge \exists yQ(y)$ .

**Exercise 71 (hand-in, due 11/26)** Show, using a tableau, that  $\exists x(O(s, x) \wedge A(x))$  is a logical consequence of the formulas in Example 3.5.

**Exercise 72 (hand-in, due 11/26)** Show, using a tableau, that  $Q(a) \wedge Q(b) \wedge \forall x(P(x) \wedge (Q(x) \rightarrow \neg P(x)))$  is unsatisfiable.

### 3.37 Remark

While the propositional tableaux algorithm always terminates, this is not the case for the predicate logic tableaux algorithm.

## 3.G Theoretical Aspects

[Schöning, 1989, Part of Chapters 1.4 and 2.3 plus additional material]

### 3.38 Theorem (monotonicity)

Let  $M, N$  be sets of formulas. If  $M \subseteq N$  then  $\{F \mid M \models F\} \subseteq \{F \mid N \models F\}$ .

**Proof:** Let  $F$  be such that  $M \models F$ .

Let  $\mathcal{A}$  be a model for  $N$ . Then all formulas in  $N$ , and hence all formulas in  $M$ , are true under  $\mathcal{A}$ . Hence  $\mathcal{A} \models F$ . This holds for all models of  $N$ , and hence  $N \models F$ . ■

### 3.39 Definition

A problem with a yes/no answer is called a *decision problem*. A decision problem is called *decidable* if there exists an algorithm which terminates on any allowed input of the problem and, upon termination, outputs the correct answer.

### 3.40 Example

“Is  $n$  an even number?” is decidable (allowed input: any  $n \in \mathbb{N}$ ).

[

1. If  $n=1$ , terminate with output 'No'.
2. If  $n=0$ , terminate with output 'Yes'.
3. Set  $n := n-2$ .
4. Go to 1.

]

### 3.41 Theorem (decidability of finite propositional entailment)

The problem of deciding whether a finite set  $M$  of propositional formulas entails some other propositional formula  $F$  is decidable.

**Proof:**  $M$  contains only a finite number of propositional variables. Use truth tables to check whether all models of  $M$  are models of  $F$ . ■

**3.42 Theorem (complexity of propositional satisfiability)**

The problem of deciding whether a finite set of propositional formulas is satisfiable, is NP-complete.

**Proof:** See any book or lecture on computational complexity theory. ■

**3.43 Theorem (complexity of propositional entailment)**

The problem of deciding whether a finite set of propositional formulas entails some other formula is NP-complete.

**3.44 Theorem (decidability of Datalog entailment)**

The problem of deciding whether a finite Datalog program  $P$  Herbrand-entails some  $A \in B_P$  is decidable.

**Proof:** The set of propositional formulas  $v(P)$  as defined in Definition 2.22 is finite. Theorems 2.24 and 3.41 then complete the proof. ■

**3.45 Theorem (undecidability of predicate logic)**

The problem “*Given a formula  $F$ , is  $F$  valid?*” is undecidable.

**Proof:** Out of scope for this lecture. Usually by reduction of the Halting Problem. ■

**Exercise 73 (no hand-in)** Show, that the problem “*Given a formula  $F$  and a finite set of formulas  $M$ , is  $M \models F$ ?*” is undecidable. [use Theorem 3.45]

**3.46 Definition**

A decision problem is *semi-decidable* if there exists an algorithm which, on any allowed input of the problem, terminates if the answer is ‘yes’ and outputs the correct answer. Note that, in this case, the algorithm does not necessarily terminate if the correct answer is ‘no’.

**3.47 Theorem (semi-decidability of predicate logic)**

The problem “*Given a formula  $F$ , is  $F$  valid?*” is semi-decidable.

**Proof:** We have, e.g., the tableaux calculus for this. ■

**3.48 Remark**

The formula

$$F = \forall x \forall y \forall u \forall v \forall w (P(x, f(x)) \wedge \neg P(y, y) \wedge ((P(u, v) \wedge P(v, w)) \rightarrow P(u, w))$$

is satisfiable but has no finite model (with  $U_{\mathcal{A}}$  finite).

$\mathcal{A} = (U_{\mathcal{A}}, I_{\mathcal{A}})$  is a model, where

$$\begin{aligned} U_{\mathcal{A}} &= \mathbb{N} \\ P^{\mathcal{A}} &= \{(m, n) \mid m < n\} \\ f^{\mathcal{A}}(n) &= n + 1 \end{aligned}$$

Assume  $B = (U_B, I_B)$  is a finite model for  $F$ . Let  $u_0 \in U_B$  and consider the sequence  $(u_i)_{i \in \mathbb{N}}$  with  $u_{i+1} = f^B(u_i)$ . Since  $U_B$  is finite, there exist  $i < j$  with  $u_i = u_j$ .  $F$  enforces transitivity of  $F$ , hence  $(u_i, u_j) \in P^B$ . But since  $u_i = u_j$  this contradicts  $\forall y \neg P(y, y)$ .

### 3.49 Theorem (Löwenheim-Skolem)

If a (finite or) countable set of formulas is satisfiable, then it is satisfiable in a countable domain.

### 3.50 Remark

According to Theorem 3.49, it is impossible to axiomatize the real numbers in first-order predicate logic.

## 4 Application: Knowledge Representation for the World Wide Web

[See [Hitzler et al., 2009] for further reading.]

[Slideset 2]