MATHEMATICAL LOGIC FOR COMPUTER SCIENCE

A.Y. 20/21, HANDOUT N. 3

Abstract. General problems based on $\models$. Logical equivalence, logical laws, normal forms.

1. Comments on truth, general problems

We introduced the following three relations:

$$\mathfrak{A} \models F[\alpha].$$

$$\mathfrak{A} \models F.$$

$$\models F.$$

Recall that by definition $\mathfrak{A} \models F$ where F is a formula means that for all α, $\mathfrak{A} \models F[\alpha]$. If $F(x_1, \ldots, x_n)$ is a formula and α is an assignment we sometimes write $\mathfrak{A} \models F[\binom{x_1,\ldots,x_n}{a_1,\ldots,a_n}]$ or just $\mathfrak{A} \models F[a_1, \ldots, a_n]$, where $\alpha(x_i) = a_i$. As we observed, only the free variables in a formula matter with respect to the fact that $\mathfrak{A} \models F[\alpha]$ or not.

Note that $\mathfrak{A} \not\models F[\alpha]$ is the negation of $\mathfrak{A} \models F[\alpha]$ and holds iff $\mathfrak{A} \models \neg F[\alpha]$.

Note instead that $\mathfrak{A} \not\models F$ is the negation of $\mathfrak{A} \models F$ so it means: for some α, $\mathfrak{A} \not\models F[\alpha]$. On the other hand $\mathfrak{A} \models \neg F$ means for all α, $\mathfrak{A} \models \neg F[\alpha]$.

We already observed that **for sentences** we have the following dichotomy:

$$\mathfrak{A} \not\models F \text{ iff } \mathfrak{A} \models \neg F.$$

Using the notions introduced so far we can isolate a number of natural and interesting general problems.

1.1. Decision Problem(s).

Fix a language $\mathcal{L}$. The following is called the **decision problem** for $\mathcal{L}$:

> Given a sentence S in $\mathcal{L}$, decide whether $\models S$ or not.

This problem has no algorithmic solutions for almost all non-trivial first-order languages, while it has algorithmic solutions for some restricted languages.

1.2. Truth Evaluation/Model-Checking.

Fix a language $\mathcal{L}$. The following is called the **truth-evaluation** or **model-checking** problem for $\mathcal{L}$:

> Given a structure $\mathfrak{A}$, a formula F in $\mathcal{L}$ and an assignment α in $\mathfrak{A}$, decide whether $\mathfrak{A} \models F[\alpha]$ or not.

The version for sentences is the following:

> Given a structure $\mathfrak{A}$ and a sentence S in $\mathcal{L}$, decide whether $\mathfrak{A} \models S$ or not.

Notes by Lorenzo Carlucci, carlucci@di.uniroma1.it.

This fundamental problem can be asked for $\mathfrak{A}$ in a particular class $\mathcal{C}$ rather than arbitrary structures, as well as for particular languages.

A typical case of interest is that of **finite** structures. For these, it can be observed that the very definition of the relation $\mathfrak{A} \models F[\alpha]$ can be turned into a recursive algorithm. A straightforward analysis of this algorithm shows that its time complexity is polynomial in the size of the structure and exponential in the size of the formula; while the space complexity is logarithmic in the size of the structure and polynomial in the size of the formula. In fact, Grädel proved that Model-Checking for first-order logic is **PSPACE**-complete.

1.3. **Query Evaluation.** It is natural to consider formulas with free variables as defining subsets of a structure, by looking at the set of assignments that satisfy the formula. From the perspective of Database Theory, this means reading a formula as a query; from a more mathematical perspective this means investigating which subsets of a structure can be defined by formulas in the language (e.g., can we define the set prime numbers using the language of arithmetic over the structure of the natural numbers?).

> Given a structure $\mathfrak{A}$ and a formula $F(x_1, \dots, x_n)$ in $\mathcal{L}$, consider the set
> $$\{(a_1, \dots, a_n) \in A^n \ : \ \mathfrak{A} \models F[a_1, \dots, a_n]\}.$$
> Is there a decision algorithm for this set?

1.4. **Query complexity/Decision of Theories.** Some structures/databases are more interesting than others. It is therefore natural to fix one such structure and investigate the set of sentences that are true in it. Fix a structure $\mathfrak{A}$. We consider:

> $Th(\mathfrak{A}) = \{S \text{ a sentence } \ : \mathfrak{A} \models S\}$. Is there an algorithm to decide this set?

$Th(\mathfrak{A})$ is called **the theory of** $\mathfrak{A}$. The decision problem for theories of particular structures is of fundamental interest in Logic both from the perspective of database theory as from a more general mathematical perspective. In general, depending on the structure $\mathfrak{A}$ and on the language considered, we can have very different answers, ranging between the following:

(1) The problem is algorithmically undecidable.
(2) The problem is decidable but unfeasible.
(3) The problem is decidable and feasible.

For the particular case of finite structures the problem is always decidable and we can inquiry in its computational complexity. In this case we speak about **expression complexity** or **query complexity**. For first-order logic languages the query complexity can be proved to **PSPACE**.

1.5. **Data complexity/Decision of Models.** It is interesting in some cases to fix one particularly interesting sentence S and investigate the set of structures that make it true. Fix a sentence S. We consider:

> $Mod(S) = \{\mathfrak{A} \ : \ \mathfrak{A} \models S\}$. Is there an algorithm to decide this set?

$Mod(S)$ is called the set of **models of** S. For general possibly infinite structures it might not make sense to ask for an algorithm to decide membership in $Mod(S)$, while for finite structures such an algorithm always exists and we can inquiry in its computational complexity. In this case we speak about **structure** or **data complexity**. For first-order logic languages the data complexity can be proved to **LOGSPACE**.

It is sometimes interesting to consider the problem only for structures of a certain type, i.e., belonging to a certain class $\mathcal{C}$. Also, as we will see, it might make sense to consider the models of more than a single sentence. For finitely many sentences $S_1, \dots, S_n$, this is equivalent to investigating the models of their conjunction $S_1 \wedge \cdots \wedge S_n$. For infinitely many sentences we are asking about which structures satisfy them jointly: is $\mathcal{S} = \{S_1, S_2, \dots\}$ is a set of sentences, we can define $Mod(\mathcal{S})$ as the set of all structures $\mathfrak{A}$ such that for all $i \in \mathbf{N}$ we have $\mathfrak{A} \models S_i$.

2. Logical Equivalence, Logic Laws, Normal Forms

Definition 2.1 (Logical Equivalence)**.** Two formulas F and G are called **logically equivalent** if

$$\models F \leftrightarrow G.$$

We write $F \equiv G$ in this case.

Recall that, if F and G have free variables and these are contained in $\{x_1, \ldots, x_n\}$, then the above condition is equivalent to

$$\models \forall x_1 \ldots \forall x_n (F \leftrightarrow G).$$

Logical equivalence is an equivalence relation on formulas.

(1) $A \equiv A$
(2) If $A \equiv B$ and $B \equiv C$ then $A \equiv C$
(3) If $A \equiv B$ then $B \equiv A$.

The quotient of the set of formulas under $\equiv$ is called Lindenbaum Algebra.

Two logically equivalent formulas behave exactly in the same way with respect to truth/falsity in all structures under all assignments. It is reasonable to expect that they can be substituted for one another in all contexts in which only truth/falsity matter.

This is indeed the case.

We can then justify a logical equivalence by a sequence of logical equations using the below *substitution theorems*

Example We show $\models (A \to (B \to C)) \leftrightarrow ((A \wedge B) \to C)$.

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

Example We show $\models (A \to B) \leftrightarrow (\neg B \to \neg A)$.

$$\neg B \to \neg A \equiv \neg\neg B \vee \neg A \equiv B \vee \neg A \equiv \neg A \vee B \equiv A \to B$$

The following give a rigorous justification to the above procedure.

(1) If A and B are equivalent, and I substitute in A and in B the same (common) subformula with any formula, I obtain two logically equivalent formulas.
(2) If in a formula A I substitute the same subformula with two logically equivalent formulas, I obtain two logically equivalent formulas.

The notion of logical equivalence can be used to illustrate a number of noteworthy equivalences which have the dignity of being called fundamental truths or laws of logic.

2.1. Algebraic Laws.

(1) Associativity
  (a) $(A \vee B) \vee C \equiv A \vee (B \vee C)$
  (b) $(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$
(2) Commutativity
  (a) $(A \vee B) \equiv (B \vee A)$
  (b) $(A \wedge B) \equiv (B \wedge A)$
(3) Distributivity
  (a) $(A \vee (B \wedge C)) \equiv (A \vee B) \wedge (A \vee C)$
  (b) $(A \wedge (B \vee C)) \equiv (A \wedge B) \vee (A \wedge C)$
(4) De Morgan Laws
  (a) $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$
  (b) $\neg(A \wedge B) \equiv (\neg A \vee \neg B)$
(5) Double Negation Law $\neg\neg A \equiv A$

(6) Idempotency
  (a) $(A \vee A) \equiv A$
  (b) $(A \wedge A) \equiv A$

2.2. **Inter-definability of Logical Symbols.**

(1) $(A \leftrightarrow B) \equiv ((A \to B) \wedge (B \to A))$
(2) $(A \to B) \equiv (\neg A \vee B)$
(3) $(A \vee B) \equiv (\neg A \to B)$
(4) $(A \vee B) \equiv \neg(\neg A \wedge \neg B)$
(5) $(A \wedge B) \equiv \neg(\neg A \vee \neg B)$

2.3. **Other Algebraic Laws.**

(1) Absorption
  (a) $(A \vee \bot) \equiv A$
  (b) $(A \wedge \top) \equiv A$
(2) Contradiction, Excluded Middle
  (a) $(A \vee \neg A) \equiv \top$
  (b) $(A \wedge \neg A) \equiv \bot$

2.4. **Quantifiers Laws:**

(1) $\exists x(F \vee G) \equiv \exists x F \vee \exists x G$
(2) $\forall x(F \wedge G) \equiv \forall x F \wedge \forall x G$
(3) $\exists x F \equiv \neg \forall x \neg F$
(4) $\forall x F \equiv \neg \exists x \neg F$
(5) $\forall x \forall y F \equiv \forall y \forall x F$
(6) $\exists x \exists y F \equiv \exists y \exists x F$
(7) If x does not appear in F, then
  (a) $F \vee \exists x G \equiv \exists x(F \vee G), \quad F \wedge \exists x G \equiv \exists x(F \wedge G)$
  (b) $F \wedge \forall x G \equiv \forall x(F \wedge G), \quad F \vee \forall x G \equiv \forall x(F \vee G)$

Using the notion of logical equivalence we can establish useful normal form results in the spirit of Algebra.

First of all observe that we can always get rid of the connectives $\leftrightarrow$ and $\to$ using equivalences $(A \leftrightarrow B) \equiv (A \to B) \wedge (B \to A)$ and $(A \to B) \equiv (\neg A \vee B)$.

2.5. **Negation Normal Form.** A formula is in **negation normal form** if the negation symbol $\neg$ only appears in front of atomic formulas in the formula. A formula can be algorithmically transformed in a logically equivalent formula in Negation Normal Form by applying the De Morgan and the Double Negation laws.

2.6. **Conjunctive and Disjunctive Normal Forms.** A formula is in **Conjunctive Normal Form** is it is a conjunction of disjunctions of atomic formulas or negated atomic formulas (for short the latter two types of formulas are called literals). A formula is in **Disjunctive Normal Form** if it is a disjunction of conjunctions of atomic formulas or negated atomic formulas.

Every quantifier-free formula can be algorithmically transformed into a logically equivalent quantifier-free formula in CNF and into a logically equivalent formula in DNF.

For example, applying the following steps allows conversion in DNF: put the formula in Negation Normal Form, then apply distributivity laws.

2.7. **Prenex Normal Form.** A formula is in **prenex normal form** if it starts with a prefix of quantifiers followed by a formula with no quantifier (quantifier-free).

Remark 2.2 (Renaming of bound variables)**.** If y does not appear in $\forall x G$, then $\forall y G(y)$ is equivalent to $\forall x G$, where $G(y)$ is the formula obtained from G by substituting all occurrences of x by y.

Proposition 2.3 (Prenex Normal Form)**.** *For each formula F there exists a formula F' in prenex normal form logically equivalent to F. There is an algorithm to get F' from F.*

Proof. Structural induction on F. The atomic case is trivial.

Let F be of the form $\neg G$. Let G be equivalent to $Q_1 x_1 \dots Q_n x_n G'$ with quantifiers Q_i and G' quantifier-free. Then

$$F \equiv \neg Q_1 x_1 \dots Q_n x_n G' \equiv Q_1^- x_1 \dots Q_n^- x_n \neg G',$$

where Q_i^- is $\forall$ if Q_i is $\exists$ and viceversa.

Let F be of form $G \wedge H$. Let G be equivalent to $Q_1 x_1 \dots Q_n x_n G'$ and H to $\tilde{Q}_1 y_1 \dots \tilde{Q}_k y_k H'$ where we can assume (by renaming of bound variables) that the x and the y are pairwise distinct and distinct from the free variables in G and H. Then

$$F \equiv Q_1 x_1 \dots Q_n x_n G' \wedge \tilde{Q}_1 y_1 \dots \tilde{Q}_k y_k H' \equiv Q_1 x_1 \dots Q_n x_n \tilde{Q}_1 y_1 \dots \tilde{Q}_k y_k (G' \wedge H').$$

Let F be of form $\forall v G$. Let G be equivalent to $Q_1 x_1 \dots Q_n x_n G'$. We can assume that the bound variables in $Q_1 x_1 \dots Q_n x_n G'$ are different from v. Then

$$F \equiv \forall v G \equiv \forall v Q_1 x_1 \dots Q_n x_n G'.$$

□