

Introduction to Logic

Logic in Logic

Michael Genesereth
Computer Science Department
Stanford University

Whole Numbers

Object Constant: 0

Unary Function Constant: s

Binary Relation Constants:

same - the first and second arguments are identical

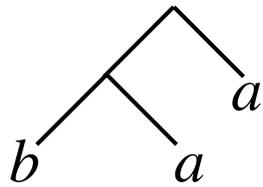
succ - the first argument immediately precedes second

less - the first argument less than or equal to second

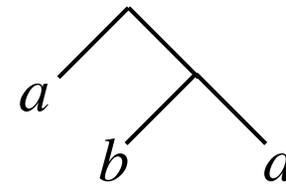
Trees

Object constants: a, b

Unary function constants: $cons$



$cons(cons(b, a), a)$



$cons(a, cons(b, a))$

Unary relation constants: $symmetric, uniform, \dots$

Binary relation constant: $subtree, congruent, mirror, \dots$

Lists

Object Constants: a, b, c, d, nil

Binary Function Constant: $cons$

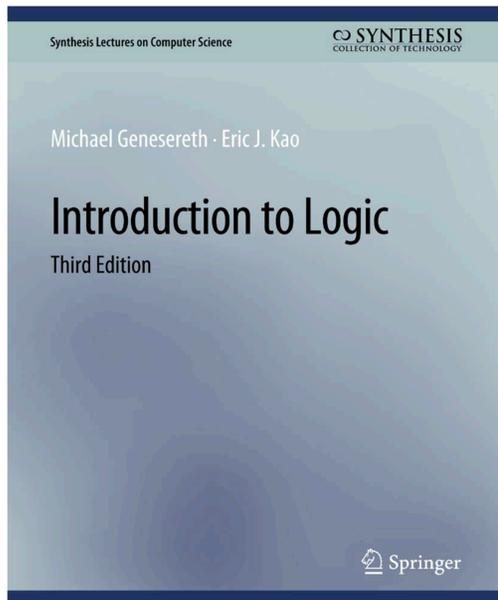
Binary Relation Constant: $member$

Ternary Relation Constant: $append$

$member(b, [a, b, c])$
 $append([a, b], [c, d], [a, b, c, d])$

Metalevel Logic

Metalevel Logic



proposition(p)
proposition(q)
proposition(r)

negation(not(x)) \Leftrightarrow *sentence(x)*

conjunction(and(x,y)) \Leftrightarrow *sentence(x)* \wedge *sentence(y)*

disjunction(or(x,y)) \Leftrightarrow *sentence(x)* \vee *sentence(y)*

implication(if(x,y)) \Leftrightarrow *sentence(x)* \Rightarrow *sentence(y)*

biconditional(iff(x,y)) \Leftrightarrow *sentence(x)* \Leftrightarrow *sentence(y)*

sentence(x) \Leftrightarrow

proposition(x) \vee *negation(x)* \vee *conjunction(x)* \vee

disjunction(x) \vee *implication(x)* \vee *biconditional(x)*

Propositional Logic in Term Logic

CHAPTER 2
Propositional Logic

2.1 Introduction

Propositional Logic is concerned with propositions and their interrelationships. The notion of a proposition here cannot be defined precisely. Roughly speaking, a proposition is a possible condition of the world that is either true or false, e.g. the possibility that it is raining. The possibility that it is cloudy, and so forth. The condition need not be true in order for it to be a proposition. In fact, we might want to say that it is false or that it is true if some other proposition is true.

In this chapter, we first look at the syntactic rules that define the language of Propositional Logic. We then introduce the notion of a truth assignment and use it to define the meaning of Propositional Logic sentences. After that, we present a mechanical method for evaluating sentences for given truth assignments, and we present a mechanical method for finding truth assignments that satisfy sentences. We conclude with some examples of Propositional Logic in formalizing Natural Language and Digital Circuits.

2.2 Syntax

In Propositional Logic, there are two types of sentences – simple sentences and compound sentences. Simple sentences express simple facts about the world. Compound sentences express logical relationships between the simpler sentences of which they are composed.

Simple sentences in Propositional Logic are often called *proposition constants* or, sometimes, *logical constants*. In what follows, we write proposition constants as strings of letters, digits, and underscores “_”, where the first character is a lower case letter. For example, *rain* is a proposition constant, as are *RAINING*, *IT_RAINING*, and *RAINING_OR_SNOWING*. *Rain* is not a proposition constant because it begins with an upper case character. *IT_RAINING* fails because it begins with a number, *rain* or *snowing* fails because it contains hyphens (instead of underscores).

Compound sentences are formed from simpler sentences and express relationships among the constituent sentences. There are five types of compound sentences, viz. negations, conjunctions, disjunctions, implications, and biconditionals.

A negation consists of the negation operator “~” and an arbitrary sentence, called the *target*. For example, given the sentence *p*, we can form the negation of *p* as shown below.

$$\neg p$$

A conjunction is a sequence of sentences separated by occurrences of the \wedge operator and enclosed in parentheses, as shown below. The constituent sentences are called *conjuncts*. For example, we can form the conjunction of *p* and *q* as follows.

CHAPTER 3
Propositional Analysis

3.1 Introduction

Satisfiability is a relationship between specific sentences and specific truth assignments. In Logic, we are usually more interested in properties and relationships of sentences that hold across all truth assignments. We begin this chapter with a look at logical properties of individual sentences (as opposed to relationships among sentences) – validity, contingency, and unsatisfiability. We then look at three types of logical relationships between sentences – logical entailment, logical equivalence, and logical consistency. We conclude with a discussion of the connections between the logical properties of individual sentences and logical relationships between sentences.

3.2 Logical Properties

In the preceding chapter, we saw that some sentences are true in some truth assignments and false in others. However, this is not always the case. There are sentences that are always true and sentences that are always false as well as sentences that are sometimes true and sometimes false. This leads to a partition of sentences into three disjoint categories.

A sentence is *valid* if and only if it is satisfied by every truth assignment. For example, the sentence $(p \vee \neg p)$ is valid. If a truth assignment makes *p* true, then the first disjunct is true and the disjunction as a whole is true. If a truth assignment makes *p* false, then the second disjunct is true and the disjunction as a whole is true.

A sentence is *unsatisfiable* if and only if it is not satisfied by any truth assignment. For example, the sentence $(p \wedge \neg p)$ is unsatisfiable. No matter what truth assignment we take, the sentence is always false. The argument is analogous to the argument in the preceding paragraph.

Finally, a sentence is *contingent* if and only if there is some truth assignment that satisfies it and some truth assignment that falsifies it. For example, the sentence $(p \wedge q)$ is contingent. If *p* and *q* are both true, it is true. If *p* and *q* are both false, it is false.

In one sense, valid sentences and unsatisfiable sentences are useless. Valid sentences do not rule out any possible truth assignments, and unsatisfiable sentences rule out all truth assignments. Thus, they tell us nothing about the real world. In this regard, contingent sentences are the most useful. On the other hand, from a logical perspective, valid and unsatisfiable sentences are useful in that, as we shall see, they serve as the basis for legal transformations that we can perform on other logical sentences.

For many purposes, it is useful to group validity, contingency, and unsatisfiability into two groups. We say that a sentence is *satisfiable* if and only if it is valid or contingent. In other words, the sentence is satisfied by at least one truth assignment. We say that a sentence is *falsifiable* if and



proposition(p)
proposition(q)
proposition(r)

negation(not(x)) \Leftrightarrow *sentence(x)*

conjunction(and(x,y)) \Leftrightarrow *sentence(x)* \wedge *sentence(y)*

disjunction(or(x,y)) \Leftrightarrow *sentence(x)* \vee *sentence(y)*

implication(if(x,y)) \Leftrightarrow *sentence(x)* \Rightarrow *sentence(y)*

biconditional(iff(x,y)) \Leftrightarrow *sentence(x)* \Leftrightarrow *sentence(y)*

sentence(x) \Leftrightarrow
proposition(x) \vee *negation(x)* \vee *conjunction(x)* \vee
disjunction(x) \vee *implication(x)* \vee *biconditional(x)*

CHAPTER 5
Natural Deduction

5.1 Introduction

Direct deduction has the merit of being simple to understand. Unfortunately, as we have seen, the proofs can easily become unwieldy. The deduction theorem helps. It asserts that, if we have a proof of a conclusion from premises, then a proof of the corresponding implication. However, that assertion is not itself a proof. *Natural deduction* cures this deficiency through the use of conditional proofs.

We begin this lesson with a discussion of conditional proofs. We then show how they are combined in the popular *Fitch proof system*. We discuss soundness and completeness of the system. And we finish by providing some tips for finding proofs using the Fitch system.

5.2 Conditional Proofs

Conditional proofs are similar to direct proofs in that they are sequences of reasoning steps. However, they differ from direct proofs in that they have more structure. In particular, sentences can be grouped into subproofs nested within outer subproofs.

As an example, consider the conditional proof shown below. It resembles a direct proof except that we have grouped the sentences on lines 3 through 5 into a subproof within our overall proof.

1. $p \Rightarrow q$ Premise
2. $p \Rightarrow r$ Premise
3. p Assumption
4. q Implication Elimination: 3, 1
5. r Implication Elimination: 4, 2
6. $p \Rightarrow r$ Implication Introduction: 3, 5

The main benefit of conditional proofs is that they allow us to prove things that cannot be proved using only ordinary rules of inference. In conditional proofs, we can make assumptions within subproofs; we can prove conclusions from these assumptions; and, from these derivations, we can derive implications outside of those subproofs, with our assumptions as antecedents and our conclusions as consequents.

The conditional proof above illustrates this. On line 3, we begin a subproof with the assumption that *p* is true. Note that *p* is not a premise in the overall problem. In a subproof, we can make

CHAPTER 4
Direct Proofs

4.1 Introduction

Checking logical entailment with truth tables has the merit of being conceptually simple. However, it is not always the most practical method. The number of truth assignments of a language grows exponentially with the number of logical constants. When the number of logical constants in a propositional language is large, it may be impossible to process its truth table.

Proof methods provide an alternative way of checking logical entailment that addresses this problem. In many cases, it is possible to create a proof of a conclusion from a set of premises that is much smaller than the truth table for the language; moreover, it is often possible to find such proofs with less work than is necessary to check the entire truth table.

We begin this lesson by defining some basic concepts – axiom schemas, rules of inference, and direct proofs. We then look at a couple of proof systems, with emphasis on one particular proof system, viz. the *Fitch system*. After that, we look at the properties of soundness and completeness – the standards by which proof systems are judged. Finally, we look at hierarchical proofs and some more heuristics about proofs.

4.2 Axiom Schemas

An axiom schema (or schema) is an expression satisfying the grammatical rules of our language except for the occurrence of metavariables (written here as Greek letters) in place of various subparts of the expression. For example, the following expression is a schema with metavariables ϕ and ψ .

$$\phi \Rightarrow (\psi \Rightarrow \phi)$$

An instance of an axiom schema is the sentence obtained by consistently substituting sentences for the metavariables in the rule. For example, the following are all instances of the schema above.

$$p \Rightarrow (q \Rightarrow p)$$

$$p \Rightarrow (p \Rightarrow p)$$

$$\neg p \Rightarrow (p \Rightarrow \neg p)$$

$$(p \Rightarrow q) \Rightarrow (q \Rightarrow p)$$

$$(p \Rightarrow q) \Rightarrow (q \Rightarrow p)$$

An axiom schema is *valid* if and only if every instance of the schema is valid. The schema above is valid, as are the schemas shown below.

Reflexivity $\phi \Rightarrow \phi$

CHAPTER 6
Resolution Proofs

6.1 Introduction

Propositional Resolution is a powerful rule of inference for Propositional Logic. Using Propositional Resolution (without axiom schemas or other rules of inference), it is possible to build a theorem prover that is sound and complete for all of Propositional Logic. What makes the search space using Propositional Resolution is much smaller than for standard Propositional Logic.

This chapter is devoted entirely to Propositional Resolution. We start with a look at clause form, a variation of the language of Propositional Logic. We then examine the resolution rule itself. We close with some examples.

6.2 Clause Form

Propositional Resolution works only on expressions in clause form. Before the rule can be applied, the premises and conclusions must be converted to this form. Fortunately, as we shall see, there is a simple procedure for making this conversion.

A *literal* is either an atomic sentence or a negation of an atomic sentence. For example, if *p* is a logical constant, the following sentences are both literals.

$$p$$

$$\neg p$$

A *clause sentence* is either a literal or a disjunction of literals. If *p* and *q* are logical constants, then the following are clause sentences.

$$p$$

$$\neg p$$

$$\neg p \vee q$$

A clause is the set of literals in a clause sentence. For example, the following sets are the clauses corresponding to the clause sentences above.

$$\{p\}$$

$$\{\neg p\}$$

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

Note that the empty set $\{\}$ is also a clause. It is equivalent to an empty disjunction and, therefore, is unsatisfiable. As we shall see, it is a particularly important special case.

A *resolvent clause*, there is a simple procedure for computing an arbitrary set of Propositional

Basic Idea

(1) Represent Propositional Logic *sentences* as *terms* in Term Logic.

$p \wedge \neg q$ represented as $and(p, not(q))$

(2) Write Term Logic sentences to define the syntax and semantics of Propositional Logic.

$conjunction(and(p, not(q)))$

(3) Create Term Logic proofs of Propositional Logic metatheorems (e.g. soundness, completeness, deduction theorem, and so forth).

$\forall x. \forall y. (entails(x, y) \Rightarrow proves(x, y))$

Syntactic Metavocabulary

Object Constants (representing *propositions*):

p, q, r

Syntactic Metavocabulary

Object Constants (representing propositions):

p, q, r

Function constants (representing logical operators):

$not(x)$

$if(x,y)$

$and(x,y)$

$iff(x,y)$

$or(x,y)$

These are terms!

Not sentences!

Syntactic Metavocabulary

Object Constants (representing propositions):

p, q, r

Function constants (representing logical operators):

$not(x)$

$if(x,y)$

$and(x,y)$

$iff(x,y)$

$or(x,y)$

These are terms!

Not sentences!

Unary Relation Constants (properties of sentences):

$proposition(x)$

$implication(x)$

$negation(x)$

$biconditional(x)$

$conjunction(x)$

$sentence(x)$

$disjunction(x)$

These are sentences.

Syntactic Metadefinitions

proposition(p)

proposition(q)

proposition(r)

negation(not(x)) \Leftrightarrow *sentence(x)*

conjunction(and(x,y)) \Leftrightarrow *sentence(x)* \wedge *sentence(y)*

disjunction(or(x,y)) \Leftrightarrow *sentence(x)* \wedge *sentence(y)*

implication(if(x,y)) \Leftrightarrow *sentence(x)* \wedge *sentence(y)*

biconditional(iff(x,y)) \Leftrightarrow *sentence(x)* \wedge *sentence(y)*

sentence(x) \Leftrightarrow

proposition(x) \vee *negation(x)* \vee *conjunction(x)* \vee

disjunction(x) \vee *implication(x)* \vee *biconditional(x)*

Semantic Metavocabulary

Unary Relation Constants (properties of sentences):

valid(x) - validity

contingent(x) - contingency

unsatisfiable(x) - unsatisfiability

Binary Relation Constants (relations among sentences):

equivalent(x,y) - logical equivalence

entails(x,y) - logical entailment

consistent(x,y) - consistency

We also need to talk about truth assignments in order to define these notions. Doable but messy; skipping here.

Semantic Metatheorems

Validity of Axiom Schemata:

$$\mathit{valid}(\mathit{or}(x,\mathit{not}(x))) \Leftrightarrow \mathit{sentence}(x)$$

Equivalence and Entailment:

$$\mathit{equivalent}(x,y) \Leftrightarrow \mathit{entails}(x,y) \wedge \mathit{entails}(y,x)$$

Deduction Theorem:

$$\mathit{entails}(\mathit{and}(x,y),z) \Leftrightarrow \mathit{entails}(x,\mathit{if}(y,z))$$

Rules of Inference

And Introduction:

$$\forall x. \forall y. (\textit{sentence}(x) \wedge \textit{sentence}(y) \Leftrightarrow \textit{ai}(x, y, \textit{and}(x, y)))$$

And Elimination:

$$\forall x. \forall y. (\textit{sentence}(x) \wedge \textit{sentence}(y) \Leftrightarrow \textit{ae}(\textit{and}(x, y), x))$$

$$\forall x. \forall y. (\textit{sentence}(x) \wedge \textit{sentence}(y) \Leftrightarrow \textit{ae}(\textit{and}(x, y), y))$$

More Metatheorems

Soundness:

$$\forall x. \forall y. (\textit{proves}(x,y) \Rightarrow \textit{entails}(x,y))$$

Completeness:

$$\forall x. \forall y. (\textit{entails}(x,y) \Rightarrow \textit{proves}(x,y))$$

Term Logic in Term Logic

Can we define Term Logic in Term Logic?

Basic idea: represent Term Logic expressions as terms in Term Logic, write sentences to define syntax and semantics, prove metatheorems.

Syntactic Metavocabulary

NB: We need terms to represent *functional terms* and *relational sentences*.

$$p(a, f(a)) \quad \text{relsent}(p, a, \text{funterm}(f, a))$$

NB: We need *constants* in our language to refer to *variables* in the language we are describing.

$$\forall y. p(y, f(y)) \quad \text{forall}(ny, \text{relsent}(p, ny, \text{funterm}(f, ny)))$$

Syntactic Metadeclarations

obconst(a)

funconst(f)

relconst(r)

variable(nx)

functionalterm(funterm(w,x)) \Leftrightarrow *funconst(w)* \wedge *term(x)*

relationalsentence(relsent(w,x)) \Leftrightarrow *relconst(w)* \wedge *term(x)*

negation(not(x)) \Leftrightarrow *sentence(x)*

conjunction(and(x,y)) \Leftrightarrow *sentence(x)* \wedge *sentence(y)*

disjunction(or(x,y)) \Leftrightarrow *sentence(x)* \wedge *sentence(y)*

implication(if(x,y)) \Leftrightarrow *sentence(x)* \wedge *sentence(y)*

biconditional(iff(x,y)) \Leftrightarrow *sentence(x)* \wedge *sentence(y)*

universal(forall(v,x)) \Leftrightarrow *variable(v)* \wedge *sentence(x)*

universal(exists(v,x)) \Leftrightarrow *variable(v)* \wedge *sentence(x)*

Cardinality Problem

In formalizing Propositional Logic, we *can* talk about truth assignments. The Herbrand base is always finite, and so there are only finitely many truth assignments.

In formalizing Term Logic, things are more difficult. The Herbrand base can be infinite (though it is always *countable*). However, the number of truth assignments can be *uncountable*. Unfortunately, we have only countably many terms!

Term Logic in Another Logic

*Can we define the semantics of Term Logic in some **other** logic?*

Good News / Bad News: First-Order Logic (FOL) allows for uncountable universes and so in principle can be used. Unfortunately, FOL theories with infinite universes have *nonstandard models* (unintended models that *cannot be excluded*).

NB: FOL is *weaker* than Term Logic. Some notions that can be defined exactly in Term Logic cannot be defined in FOL without allowing nonstandard models, e.g. Peano Arithmetic, transitive closure.

Term Logic in Another Logic

Can we define the semantics of Term Logic in some other logic?

Good News / Bad News: First-Order Logic (FOL) allows for uncountable universes and so in principle can be used. Unfortunately, FOL theories with infinite universes have *nonstandard models* (unintended models that *cannot be excluded*).

Good News / Bad News: Second-Order Logic (SOL) allows us to eliminate these nonstandard models, but it is more complicated and there is no complete proof procedure.

Self-Referential Logic

Can we use this "metalevel" approach to relate the truth of sentences described in a metalanguage to sentences describing those sentences?

Truth Predicate

Can we use this "metalevel" approach to relate the truth of sentences described in a metalanguage to sentences describing those sentences?

Example: If so, can we define a *truth predicate* that allows us to say whether or not a sentence is true?

$$\forall x. \forall y. (\text{true}(\text{relsent}(p, x, y)) \Leftrightarrow p(x, y))$$

Beliefs

Can we use this "metalevel" approach to relate the truth of sentences described in a metalanguage to sentences describing those sentences?

Example: Can we use our truth predicate to formalize the truth of people's beliefs, beliefs about those beliefs, etc.?

$$\forall x.(\text{believes}(\text{john}, x) \Leftrightarrow \text{true}(x))$$

Disinformation

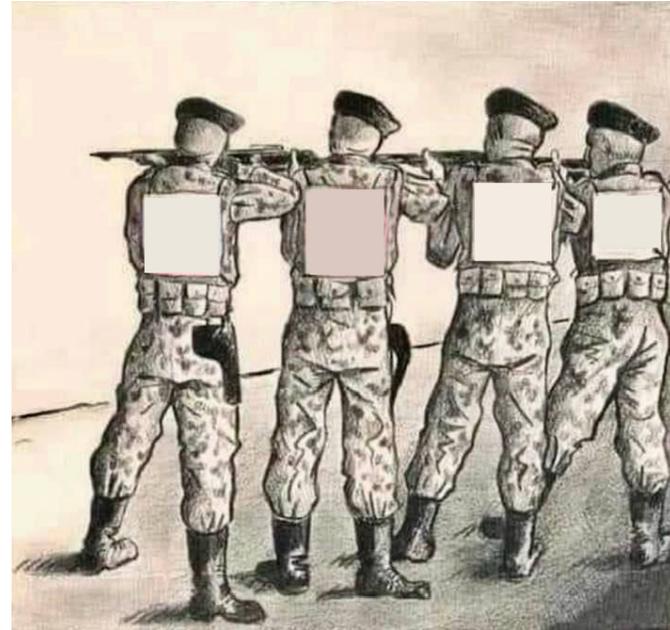
Can we use this "metalevel" approach to relate the truth of sentences described in a metalanguage to sentences describing those sentences?

Example: Can we use our truth predicate to formalize the truth or falsehood of people's statements?

$$\forall x.(says(john,x) \Rightarrow true(x))$$

Puzzle

You are taken prisoner by a drug cartel and told: *If you tell a lie, we will hang you. If you tell the truth, we will shoot you.* What do you say?



You say: *You will hang me.*

Result: They hang you *and* shoot you!

Suggestion: You should have asked if they meant *if and only if*.

Paradoxes

Unfortunately, trying to use a logic to define a truth predicate is problematic.

We run the risk of *paradoxes* (sentences that are both true and false / neither true nor false).

This sentence is false.

Also *nonsense terms* (terms that do not refer to anything).

The set of all sets that do not contain themselves

Results

We *can* completely formalize Propositional Logic in Term Logic.

(1) We can formalize *some details* of Term Logic in Term Logic but not everything. (2) We can formalize *more* of Term Logic in FOL, but we end up with *nonstandard models*. (3) We can eliminate nonstandard models using SOL, but it is complicated and there is no complete proof procedure.

We can axiomatize a metalevel truth predicate; but, unless we are very, very careful, this can lead to unpleasant complications, e.g. paradoxes.

