

Introduction to Logic

Functional Logic

Michael Genesereth
Computer Science Department
Stanford University

Motivation

Finite Worlds

n rows \times n columns in Friends, Goldrush, Minefinder

Finite Graphs

University Students

Population of a state or country

Countable Worlds

Integers - 1, 2, 3, 4, ...

Strings - "adbyug78377bh", ...

Sequences - [], [a], [b], [a,a], [a,b], [b,a], [b,b], [a,a,a], ...

Sets - {}, {a}, {b}, {a,b}, {{a},{b}}, {{a},{a,b}}, ...

Possibilities

Infinite Relational Logic - Infinite Vocabulary

a_1, a_2, a_3, \dots

Functional Logic - Structured Terms

$0, s(0), s(s(0)), s(s(s(0))), \dots$

$a, b, \text{pair}(a,a), \text{pair}(b,a), \text{pair}(b,b), \text{pair}(a, \text{pair}(a,b)), \dots$

$a, b, \text{set}(), \text{set}(a), \text{set}(b), \text{set}(a,b), \text{set}(\text{set}(a), \text{set}(a,b)), \dots$

Programme

Today

Syntax

Semantics

Properties and Relationships

Examples

Next Time

Fitch Proofs with Induction

After Thanksgiving

Equality

Review

Syntax

Words

Words are strings of letters, digits, and occurrences of the underscore character.

Variables begin with characters from the end of the alphabet (from *u* through *z*).

u, v, w, x, y, z

Constants begin with digits or letters from the beginning of the alphabet (from *a* through *t*).

a, b, c, 123, father, mother, comp225, barack_obama

Constants

Object constants (symbols) represent objects.

joe, stanford, france, 2345

Function constants (constructors) represent functions.

successor, pair, set

Relation constants (predicates) represent relations.

knows, loves

Arity

The *arity* of a function constant or a relation constant is the number of arguments it takes.

Unary function or relation constant - 1 argument

Binary function or relation constant - 2 arguments

Ternary function or relation constant - 3 arguments

n-ary function or relation constant - *n* arguments

Signatures

A *signature* consist of a set of object constants, a set of function constants, and a set of relation constants together with a specification of arity for the function constants and relation constants.

Object Constants: a, b

Unary Function Constant: f

Binary Function Constant: g

Unary Relation Constant: p

Binary Relation Constant: q

Terms

A *term* is either a variable, an object constant, or a **functional term** (defined shortly).

Terms represent objects.

Terms are analogous to noun phrases in natural language (e.g. *France*, *the set of 2 and 3*)

Functional Terms

A *functional term* is an expression consisting of an n -ary function constant and n terms enclosed in parentheses and separated by commas.

$$f(a)$$

$$f(x)$$

$$g(a, y)$$

Functional terms are terms and so *can* be nested*.

$$g(f(a), g(y, a))$$

* *unlike relational sentences*

Sentences

Three types of sentences in Functional Logic:

Relational sentences - analogous to the simple sentences in natural language

Logical sentences - analogous to the logical sentences in natural language

Quantified sentences - sentences that express the significance of variables

Relational Sentences

A *relational sentence* is an expression formed from an n -ary relation constant and n terms enclosed in parentheses and separated by commas.

$$q(a, f(a))$$

Reminder: Relational sentences are *not* terms and *cannot* be nested inside terms or relational sentences.

No! $q(a, q(a, y))$ **No!**

Logical Sentences

Logical sentences in Functional Logic are analogous to those in Propositional Logic (except with functional terms).

$$(\neg q(a, f(a)))$$

$$(p(a) \wedge p(f(a)))$$

$$(p(a) \vee p(f(a)))$$

$$(q(x, f(a)) \Rightarrow q(f(a), x))$$

$$(q(x, f(a)) \Leftrightarrow q(f(a), x))$$

Quantified Sentences

Universal sentences assert facts about all objects.

$$(\forall x.(p(x) \Rightarrow q(x, f(x))))$$

Existential sentence assert the existence of objects with given properties.

$$(\exists x.(p(x) \wedge q(x, f(x))))$$

Quantified sentences can be nested within other sentences.

$$\begin{aligned} &(\forall x.p(x)) \vee (\exists x.q(x, f(x))) \\ &(\forall x.(\exists y.q(f(x), y))) \end{aligned}$$

Parentheses

Parentheses can be removed when precedence allows us to reconstruct sentences correctly.

Precedence relations same as in Propositional Logic with quantifiers being of *higher* precedence than logical operators.

$$\begin{aligned}\forall x.p(x) \Rightarrow q(x,x) &\rightarrow (\forall x.p(x)) \Rightarrow q(x,x) \\ \exists x.p(x) \wedge q(x,x) &\rightarrow (\exists x.p(x)) \wedge q(x,x)\end{aligned}$$

Semantics

Herbrand Universe and Herbrand Base

The *Herbrand universe* for a Functional language is the set of all *ground terms* that can be formed from the vocabulary of the language.

The *Herbrand base* for a Functional language is the set of all *ground relational sentences* that can be formed from the vocabulary of the language.

Example Without Functions

Object Constants: a, b

Unary Relation Constant: p

Binary Relation Constant: q

Herbrand Universe:

$$\{a, b\}$$

Herbrand Base:

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

Example With Functions

Object Constants: a

Unary Function Constant: f

Unary Relation Constant: p

Herbrand Universe:

$\{a, f(a), f(f(a)), \dots\}$

Infinite!!!

Herbrand Base:

$\{p(a), p(f(a)), p(f(f(a))), \dots\}$

Infinite!!!

Truth Assignments

A *truth assignment* is an association between ground atomic sentences and the truth values *true* or *false*. As with Propositional Logic, we use 1 as a synonym for *true* and 0 as a synonym for *false*.

$$p(a)^i = 1$$

$$p(b)^i = 0$$

$$p(f(a))^i = 1$$

$$p(f(b))^i = 0$$

$$p(f(f(a)))^i = 0$$

$$p(f(f(b)))^i = 0$$

...

$$q(a,a)^i = 1$$

$$q(a,b)^i = 0$$

$$q(a,f(a))^i = 0$$

$$q(a,f(b))^i = 1$$

$$q(b,f(a))^i = 0$$

$$q(b,f(b))^i = 1$$

...

Everything Else

All other notions are defined the same as in Relational Logic.

The main difference is that now we have truth assignments that are *infinitely large* and there are *infinitely many* of them.

Bad News: It is no longer possible in general to determine logical entailment and other properties with truth tables.

Good News: In many cases, logical entailment can be established with finite proofs.

Example - Whole Numbers

Whole Numbers

Entities (natural numbers together with 0):

$0, 1, 2, 3, 4, \dots$

Successor:

$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow \dots$

Less Than (transitive closure of successor):

$0 < 1$	$1 < 2$	\dots
$0 < 2$	$1 < 3$	\dots
$0 < 3$	$1 < 4$	\dots
\dots	\dots	\dots

Possible Representations

Object Constants: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ...

Ground Terms: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ...

Possible Representations

Object Constants: $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, \dots$

Ground Terms: $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, \dots$

Object Constant: 0

Unary Function Constant: s

Ground Terms: $0, s(0), s(s(0)), \dots$

NB: spelling matters in our standard notation for numbers

We do not write as a, b, c, d, \dots

We write as $0, 1, 2, \dots, 9, [1,0], [1,1], [1,2], \dots, [1,0,0], \dots$

Arithmetic operations take advantage of this

Signature

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

Axiomatization

Enumerating ground relational data impossible

$same(0,0)$	$\neg succ(0,0)$	$\neg less(0,0)$
$\neg same(0,s(0))$	$succ(0,s(0))$	$less(0,s(0))$
$\neg same(0,s(s(0)))$	$\neg succ(0,s(s(0)))$	$less(0,s(s(0)))$
...

Solution - write logical and quantified sentences

Same

Definition:

$$\forall x. \text{same}(x, x)$$

$$\forall x. (\neg \text{same}(0, s(x)) \wedge \neg \text{same}(s(x), 0))$$

$$\forall x. \forall y. (\neg \text{same}(x, y) \Rightarrow \neg \text{same}(s(x), s(y)))$$

Same

Definition:

$$\forall x. \text{same}(x, x)$$

$$\forall x. (\neg \text{same}(0, s(x)) \wedge \neg \text{same}(s(x), 0))$$

$$\forall x. \forall y. (\neg \text{same}(x, y) \Rightarrow \neg \text{same}(s(x), s(y)))$$

Examples:

$$\text{same}(0, 0)$$

$$\text{same}(s(0), s(0))$$

$$\text{same}(s(s(0)), s(s(0)))$$

...

Same

Definition:

$$\forall x. \text{same}(x, x)$$

$$\forall x. (\neg \text{same}(0, s(x)) \wedge \neg \text{same}(s(x), 0))$$

$$\forall x. \forall y. (\neg \text{same}(x, y) \Rightarrow \neg \text{same}(s(x), s(y)))$$

Examples:

same(0,0)

$\neg \text{same}(0, s(0))$

$\neg \text{same}(s(0), 0)$

same(*s*(0),*s*(0))

$\neg \text{same}(0, s(s(0)))$

$\neg \text{same}(s(s(0)), 0)$

same(*s*(*s*(0)),*s*(*s*(0)))

...

...

...

Same

Definition:

$$\forall x. \text{same}(x, x)$$

$$\forall x. (\neg \text{same}(0, s(x)) \wedge \neg \text{same}(s(x), 0))$$

$$\forall x. \forall y. (\neg \text{same}(x, y) \Rightarrow \neg \text{same}(s(x), s(y)))$$

Examples:

$\text{same}(0, 0)$

$\neg \text{same}(0, s(0))$

$\neg \text{same}(s(0), 0)$

$\text{same}(s(0), s(0))$

$\neg \text{same}(0, s(s(0)))$

$\neg \text{same}(s(s(0)), 0)$

$\text{same}(s(s(0)), s(s(0)))$

...

...

...

$\neg \text{same}(s(0), s(s(0)))$

$\neg \text{same}(s(s(0)), s(0))$

$\neg \text{same}(s(0), s(s(s(0))))$

$\neg \text{same}(s(s(s(0))), s(0))$

...

...

Successor

Positives:

$$\forall y. succ(x, s(x))$$

Functionality:

$$\forall x. \forall y. \forall z. (succ(x, y) \wedge succ(x, z) \Rightarrow same(y, z))$$

or

$$\forall x. \forall y. \forall z. (succ(x, y) \wedge \neg same(y, z) \Rightarrow \neg succ(x, z))$$

Less Than

Successor:

$$\forall x. \forall y. (\text{succ}(x, y) \Rightarrow \text{less}(x, y))$$

Transitivity:

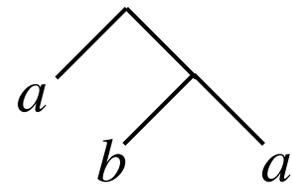
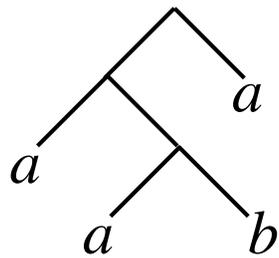
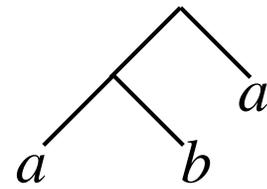
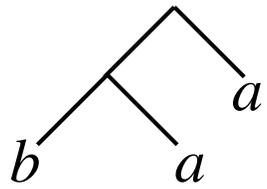
$$\forall x. \forall y. \forall z. (\text{less}(x, y) \wedge \text{less}(y, z) \Rightarrow \text{less}(x, z))$$

Irreflexivity:

$$\forall x. \neg \text{less}(x, x)$$

Example - Trees

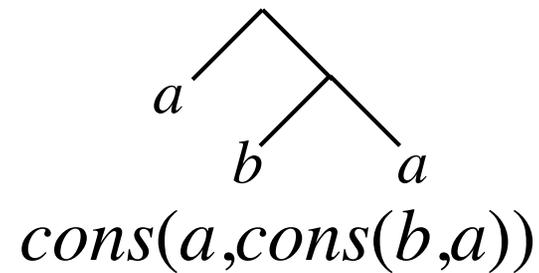
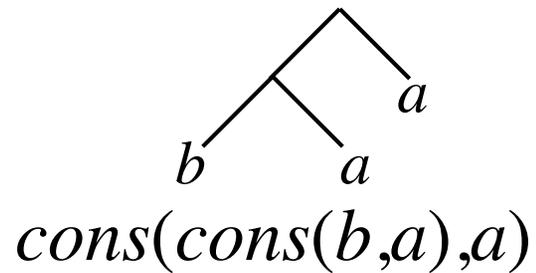
Trees



Tree Vocabulary

Object constants: a, b

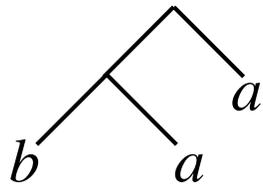
Binary function constants: $cons$



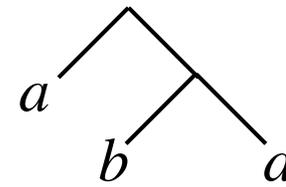
Tree Vocabulary

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: $subtree, congruent, mirror, \dots$

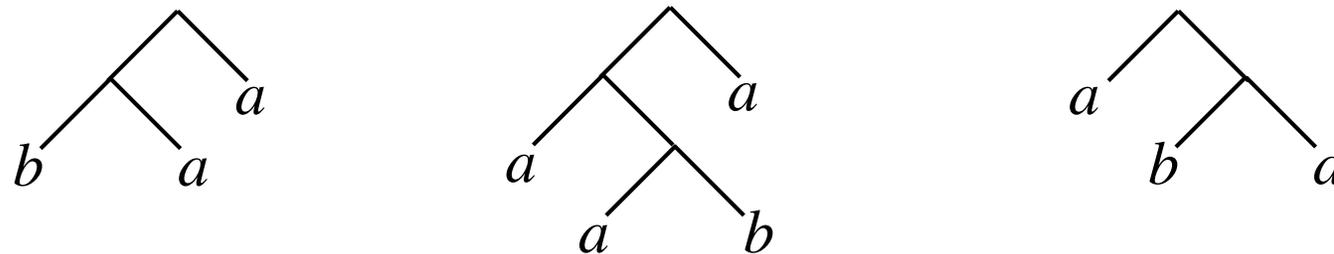
Congruence

Two trees are *congruent* if and only if they have the *same shape*. (Labels on leaf nodes irrelevant.)

Examples:



Non-Examples:



Definition

Congruence of atomic trees

congruent(a, a)

congruent(a, b)

congruent(b, a)

congruent(b, b)

Congruence of compound trees:

$$\forall u. \forall v. \forall x. \forall y. (\text{congruent}(\text{cons}(u, v), \text{cons}(x, y)) \Leftrightarrow \text{congruent}(u, x) \wedge \text{congruent}(v, y))$$

Non-Congruence of mixed trees:

$$\forall x. \forall y. (\neg \text{congruent}(a, \text{cons}(x, y)) \wedge \neg \text{congruent}(\text{cons}(x, y), a))$$

$$\forall x. \forall y. (\neg \text{congruent}(b, \text{cons}(x, y)) \wedge \neg \text{congruent}(\text{cons}(x, y), b))$$

Example - Linked Lists

Linked Lists

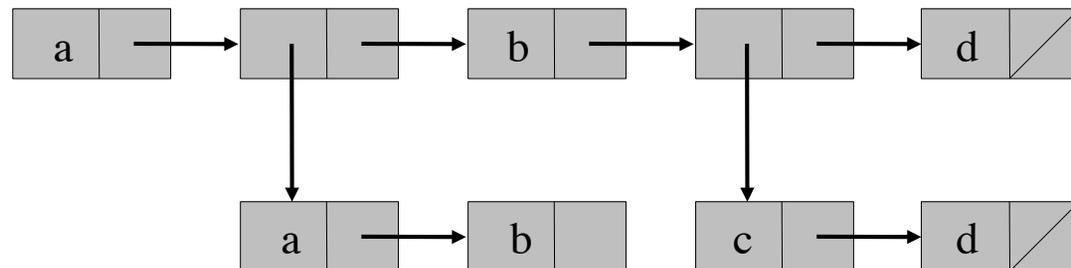
Flat Lists:

$[a, b, c, d]$

Nested Lists:

$[a, [a, b], b, [c, d], d]$

Linked List:



Representation

Example:



Representation as a functional term:

cons(a,cons(b,cons(c,cons(d,nil))))

Signature

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

Binary Function Constant: $cons$

Binary Relation Constant: $member$

Ternary Relation Constant: $append$

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

Membership

Example: $member(b, [a, b, c])$

$member(b, cons(a, cons(b, cons(c, nil))))$

Definition:

$\forall x. \forall y. member(x, cons(x, y))$

$\forall x. \forall y. \forall z. (member(x, z) \Rightarrow member(x, cons(y, z)))$

What else do we need?

Concatenation

Example: $append([a, b], [c, d], [a, b, c, d])$

$$append(cons(a, cons(b, nil)), \\ cons(c, cons(d, nil)), \\ cons(a, cons(b, cons(c, cons(d, nil))))))$$

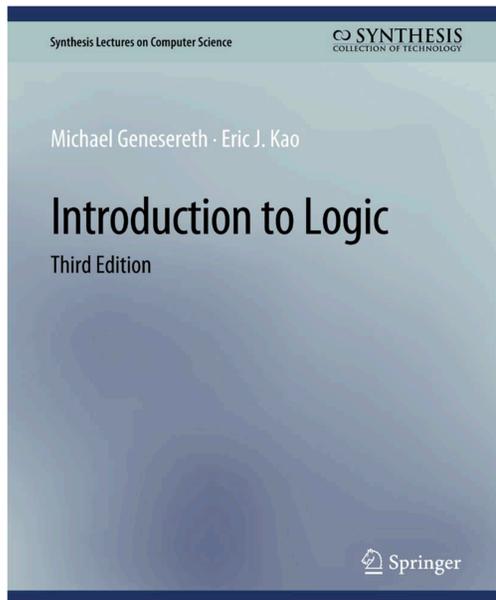
Definition :

$$\forall y. append(nil, y, y)$$
$$\forall x. \forall y. \forall z. \forall w. (append(y, z, w)$$
$$\Rightarrow append(cons(x, y), z, cons(x, w)))$$

What else do we need?

Example - 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 Functional 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, `raining` is a proposition constant, as are `RAINING`, `!RAINING`, and `raining_or_snowing`. `Raining` is not a proposition constant because it begins with an upper case character. `!RAINING` fails because it begins with a number, `raining_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.

(¬p)

A conjunction is a sequence of sentences separated by occurrences of the `^` 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 & !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 & !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 & q)` is contingent. If `p` and `q` are both true, it is true. If any of `p` and `q` is 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 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



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 is 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.

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

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

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

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

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's more, 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*, then, is a clause produced for combining an arbitrary set of Propositional

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)*

Basic Idea

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

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

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

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

(3) Create Functional 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):

$not(x)$

$if(x,y)$

$and(x,y)$

$iff(x,y)$

These are terms!!

$or(x,y)$

Syntactic Metavocabulary

Object Constants (representing propositions):

p, q, r

Function constants (representing logical operators):

$not(x)$

$if(x,y)$

$and(x,y)$

$iff(x,y)$

These are terms!!

$or(x,y)$

Unary Relation Constants (properties of sentences):

$proposition(x)$

$implication(x)$

$negation(x)$

$biconditional(x)$

$conjunction(x)$

$sentence(x)$

$disjunction(x)$

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:

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

Functional Logic in Functional Logic

Can we define Functional Logic in Functional Logic?

Basic idea: represent Functional Logic expressions as terms in Functional 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 Metadefinitions

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 Functional 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!

Functional Logic in Another Logic

Can we define the semantics of Functional 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 Functional Logic. Some notions that can be defined exactly in Functional Logic cannot be defined in FOL without allowing nonstandard models, e.g. Peano Arithmetic, transitive closure.

Functional Logic in Another Logic

Can we define the semantics of Functional 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

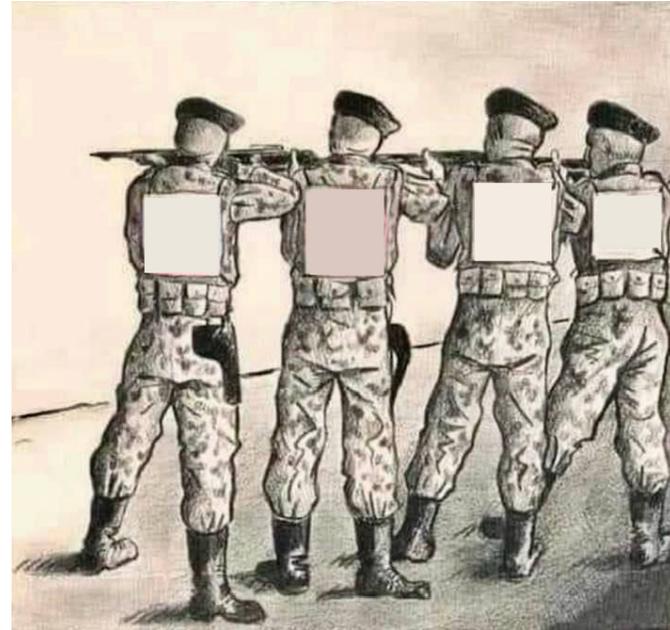
Can we use this "metalevel" approach to relate the truth of sentences described in a metalanguage to sentences describing those sentences?

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 Functional Logic.

(1) We can formalize *some details* of Functional Logic in Functional Logic but not everything. (2) We can formalize *more* of Functional 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.

