

Introduction to Logic

Term Logic

Michael Genesereth
Computer Science Department
Stanford University

Motivation

Finite Worlds

n rows $\times$ n columns in Friends, Olympics, Minefinder
Finite Graphs
University Students
Population of a state or country

Countably Infinite 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}}, ...

Relational Logic, as defined in the preceding unit, allows us to axiomatize worlds with varying numbers of objects. The main restriction is that the worlds must be finite (since we have only finitely many constants to refer to these objects).

Often, we want to describe worlds with infinitely many objects. For example, it would be nice to axiomatize arithmetic over the integers or to talk about sequences of objects of varying lengths. Unfortunately, this is not possible due to the finiteness restriction of Relational Logic.

Possibilities

Infinite Relational Logic - Infinite Vocabulary
 $a1, a2, a3, \dots$

Term Logic - Structured Terms

$0, s(0), s(s(0)), s(s(s(0))), \dots$
 $a, b, \text{pair}(a,a), \text{pair}(b,a), \text{pair}(b,b), \text{pair}(a, \text{pair}(a,b)), \dots$
 $a, b, \text{set}(), \text{set}(a), \text{set}(b), \text{set}(a,b), \text{set}(\text{set}(a), \text{set}(a,b)), \dots$

One way to get infinitely many terms is to allow our vocabulary to have infinitely many object constants. While there is nothing wrong with this in principle, it makes the job of axiomatizing things effectively impossible, as we would have to write out infinitely many sentences in many cases.

In this section, we explore an alternative to Relational Logic, called Term Logic, in which we can name infinitely many objects with a finite vocabulary. The trick is to expand our language to include not just object constants but also complex terms that can be built from object constants in infinitely many ways. By constructing terms in this way, we can get infinitely many names for objects; and, because our vocabulary is still finite, we can finitely axiomatize some things in a way that would not be possible with infinitely many object constants.

Programme

Today
Syntax
Semantics
Properties and Relationships
Examples
Fitch Proofs

Next Time
Induction

Next Week
Logic in Logic
Equality

In this chapter, we proceed through the same stages as in the introduction to Relational Logic. We start with syntax and semantics. We then discuss evaluation and satisfaction. We look at some examples. And we conclude with a discussion of some of the properties of Term Logic.

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

In Propositional Logic, sentences are constructed from a basic vocabulary of propositional constants. In Herbrand Logic, there are no propositional constants; instead we have object variables and constants.

In our examples here, we write both variables and constants as strings of letters, digits, and a few non-alphanumeric characters (such as "_"). By convention, variables begin with letters from the end of the alphabet. Examples include *u, v, w, x, y, z*. By convention, all constants begin with either letters (other than *u, v, w, x, y, z*) or digits. Examples include *a, b, c, 123, comp225, and 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

There are three types of constants in Term Logic - object constants, function constants, and relation constants. Object constants represent objects, e.g. joe, stanford, usa, 2345. Function constants represent functions on objects, e.g. father, mother, the age of a person, plus, times. Relation constants represent relations on objects, e.g. knows and likes.

Note that there is no distinction in spelling between object constants, function constants, and relation constants. The type of each such word is determined by its usage or, in some cases, in an explicit specification.

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

As we shall see, function constants and relation constants are used in forming complex expressions by combining them with an appropriate number of arguments. Accordingly, each function constant and relation constant has an associated arity, i.e. the number of arguments with which that function constant or relation constant can be combined. A function constant or relation constant that can be combined with a single argument is said to be unary; one that can be combined with two arguments is said to be binary; one that can be combined with three arguments is said to be ternary; more generally, a function or relation constant that can be combined with *n* arguments is said to be *n*-ary.

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*

A signature consists of a non-empty set of object constants, a possibly empty set of function constants, a non-empty set of relation constants and an assignment of arities for each of the function constants and relation constants. For example, we might the signature consisting of object constants *a* and *b*, the unary function constant *f*, the binary function constant *g*, the unary relation constant *p*, and the 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*)

A term is defined to be a variable, an object constant, or a functional expression (as defined in a minute). Terms typically represent objects presumed or hypothesized to exist in the world; and, as such, they are analogous to noun phrases in natural language, e.g. Joe or the car's left front wheel.

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.

$$\begin{array}{l} f(a) \\ f(x) \\ g(a, y) \end{array}$$

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

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

* *unlike relational sentences*

A functional term is an expression formed from an n -ary function constant and n terms enclosed in parentheses and separated by commas. For example, if g is a binary function constant and if a and y are terms, then $g(a, a)$ is a functional term, as are $g(a, y)$ and $g(y, y)$.

Note that functional terms can be nested inside of other functional terms. For example, we can apply the function constant g to the functional term $g(a, y)$ and the functional term $g(y, a)$.

Sentences

Three types of sentences in Term 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

There are three types of sentences in Term Logic. There are relational sentences, which are analogous to propositions in Propositional Logic. There are logical sentences, which are analogous to the logical sentences in Propositional Logic. And there are quantified sentences, which 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!

A relational sentence is an expression formed from an n -ary relation constant and n terms. For example, if q is a relation constant with arity 2 and if a and y are terms, then the expression shown below is a syntactically legal relational sentence.

Note that, unlike functional terms, relational sentences may not be nested within other relational sentences.

Logical Sentences

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

$$\begin{aligned} &(\neg q(a, f(a))) \\ &(p(a) \wedge p(f(a))) \\ &(p(a) \vee p(f(a))) \\ &(q(x, f(a)) \Rightarrow q(f(a), x)) \\ &(q(x, f(a)) \Leftrightarrow q(f(a), x)) \end{aligned}$$

Logical sentences in Term Logic are defined the same as in Propositional Logic. There are negations, conjunctions, disjunctions, implications, and equivalences. The syntax is exactly the same, except that the elementary components are relational sentences and equations rather than proposition constants.

Quantified Sentences

Universal sentences assert facts about all objects.

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

Existential sentences 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.

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

There are two types of quantified sentences - universally quantified sentences and existentially quantified sentences.

A universally quantified sentence is used to assert that all objects have a certain property. For example, the top expression here is a universally quantified sentence asserting that p holds of an object, then q holds of that object and itself. The universal quantifier is notated with this upside down A.

An existentially quantified sentence is used to assert that there are objects having a certain property. For example, the middle expression is an existentially quantified sentence asserting that there is an object that satisfies p and, when paired with itself satisfies q as well. The existential quantifier is notated by an E rotated through 180 degrees.

Note that quantified sentences can be nested within other sentences. For example, in the first sentence at the bottom, we have quantified sentences inside of a disjunction. In the second sentence at the bottom, we have a quantified sentence nested inside of another quantified sentence.

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.

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

As with Propositional Logic and Relational Logic, we can drop unneeded parentheses in Term Logic, relying on precedence to disambiguate the structure of unparenthesized sentences. In Term Logic, the precedence relations of the logical operators are the same as in Propositional Logic, and quantifiers have higher precedence than logical operators.

The examples here show how to parenthesize sentences with both quantifiers and logical operators. The sentences on the right are partially parenthesized versions of the sentences on the left. (To be fully parenthesized, we would need to add parentheses around each of the sentences as a whole.) Notice that in each case the quantifier applies only to the sentences to its right, even though the other sentence also contains occurrences of the quantified variable.

Semantics

Herbrand Universe and Herbrand Base

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

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

The Herbrand base for a Functional language is the set of all ground atomic sentences that can be formed from the vocabulary of the language. Let's look at some examples.

Example Without Functions

Object Constants: a, b
Unary Relation Constant: p
Binary Relation Constant: q

Herbrand Universe:

$\{a, b\}$

Herbrand Base:

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

For a schema with object constants a and b , no function constants, and relation constants p and q where p has arity 1 and q has arity 2, the Herbrand base is shown here. We have one p sentence for each object constant, and we have one q sentence for each of the four possible pairs of object constants.

It is worthwhile to note that, for a given relation constant and a finite set of terms, there is an upper bound on the number of possible ground relational sentences that can be formed using that relation constant. In particular, for a domain of size b , there are b^n distinct n -tuples of object constants; and hence there are b^n ground relational sentences for each n -ary relation constant. Since the number of relation constants in a signature is finite, this means that the Herbrand base is also finite.

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!!!*

Of course, not all Herbrand bases are finite. In the presence of function constants, the number of ground terms is infinite; and so the Herbrand base is infinite. For example, in a language a single object constant a and a single unary function constant f and a single unary relation constant p , the Herbrand base consists of the sentences $p(a)$, $p(f(a))$, $p(f(f(a)))$, and so forth.

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*.

$p(a)^i = 1$	$q(a,a)^i = 1$
$p(b)^i = 0$	$q(a,b)^i = 0$
$p(f(a))^i = 1$	$q(a,f(a))^i = 0$
$p(f(b))^i = 0$	$q(a,f(b))^i = 1$
$p(f(f(a)))^i = 0$	$q(b,f(a))^i = 0$
$p(f(f(b)))^i = 0$	$q(b,f(b))^i = 1$
...	...

Infinite!!!

A truth assignment for a Term Logic language is a function that maps each ground relational sentence in the Herbrand base to a truth value. As in Propositional Logic, we use the digit 1 as a synonym for true and 0 as a synonym for false; and we refer to the value assigned to a ground atom by writing the atom with the name of the truth assignment as a superscript. The truth assignment i defined here is an example for the case of the language mentioned earlier.

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.

A truth assignment for a Term Logic language is a function that maps each ground relational sentence in the Herbrand base to a truth value. As in Propositional Logic, we use the digit 1 as a synonym for true and 0 as a synonym for false; and we refer to the value assigned to a ground atom by writing the atom with the name of the truth assignment as a superscript. The truth assignment i defined here is an example for the case of the language mentioned earlier.

Example - Whole Numbers

Whole Numbers

Entities (natural numbers together with 0):

0, 1, 2, 3, 4, ...

Successor:

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

Less Than (transitive closure of successor):

0 < 1	1 < 2	...
0 < 2	1 < 3	...
0 < 3	1 < 4	...
...	...	...

Now, let's look at the problem of encoding arithmetic for all of the natural numbers. Since there are infinitely many natural numbers, we need infinitely many terms.

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, ...

One way to get infinitely many terms is to have a signature with infinitely many object constants. These constants are the only ground terms in our language. There is nothing wrong with this in principle. However, it makes the job of axiomatizing arithmetic effectively impossible, as we would have to write out infinitely many literals to capture our successor relation.

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, ...

Object Constant: 0

Unary Function Constant: s

Ground Terms: 0, $s(0)$, $s(s(0))$, ...

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, ..., 9, [1,0], [1,1], [1,2], ..., [1,0,0], ...

Arithmetic operations take advantage of this

An alternative approach is to represent numbers using a single object constant (e.g. 0) and a single unary function constant (e.g. s). We can then represent every number n by applying the function constant to 0 exactly n times. In this encoding, $s(0)$ represents 1; $s(s(0))$ represents 2; and so forth. With this encoding, we automatically get an infinite universe of terms, and we can axiomatize arithmetic with finitely many sentences.

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

In what follows, we use the second approach. Here is our signature. We have one object constant 0 and the successor function s . We have one binary relation constant $same$ which holds of two terms if and only if they are identical. And we have two ternary relation constants, one for plus and one for times.

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

Unfortunately, axiomatizing Peano Arithmetic is more challenging than axiomatizing Modular Arithmetic. We cannot just write out ground relational sentences to characterize our relations, because there are infinitely many cases to consider. For Peano Arithmetic, we must rely on logical sentences and quantified sentences, not just because they are more economical but because they are the only way we can characterize our relations in finite space.

Same

Definition:

$$\forall x. same(x,x)$$

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

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

Let's look at the $same$ relation first. The axioms shown here define the $same$ relation in terms of 0 and the s function. For all x , $same(x,x)$. For all x , 0 is not the same as $s(x)$ and $s(x)$ is not the same as 0. For all x and for all y , if x is not the same as y , then $s(x)$ is not the same as $s(y)$.

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:

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

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

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

...

It is easy to see that these axioms completely characterize the same relation. By the first axiom, the same relation holds of every term and itself. The other two axioms tell us what is not true.

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:

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

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

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

...

The second axiom tells us that 0 is not the same as any composite term. The same holds true with the arguments reversed.

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:

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

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

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

...

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

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

...

...

The third axiom builds on these results to show that non-identical composite terms of arbitrary complexity do not satisfy the same relation. Viewed the other way around, to see that two non-identical terms are not the same, we just strip away occurrences of s from each term till one of the two terms becomes 0 and the other one is not 0. By the second axiom, these are not the same, and so the original terms are not the same.

Successor

Positives:

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

Functionality:

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

or

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

Once we have the same relation, we can define the other relations in our arithmetic. The axioms shown here define the plus relation in terms of 0, s, and same. Adding 0 to any number results in that number. If adding a number x to a number y produces a number z, then adding the successor of x to y produces the successor of z. Finally, using same, we write a functionality axiom for plus.

Less Than

Successor:

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

Transitivity:

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

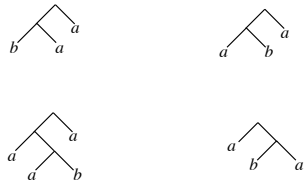
Irreflexivity:

$$\forall x. \neg less(x, x)$$

Same as the way we defined above in the Blocks World.

Example - Trees

Trees

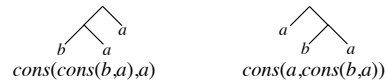


Trees have many properties - e.g. symmetry, uniformity. Also many relationships - e.g. congruence.

Tree Vocabulary

Object constants: a, b

Binary function constants: $cons$



Tree Vocabulary

Object constants: a, b

Binary function constants: $cons$



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

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

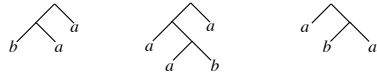
Congruence

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

$\text{congruent}(a, a)$
 $\text{congruent}(a, b)$
 $\text{congruent}(b, a)$
 $\text{congruent}(b, b)$

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:

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

We can similarly define symmetry, uniformity, mirror image, and so forth.
 We can show that every uniform tree is symmetric.
 We can show that every symmetric tree is congruent to its mirror image.

Example - Linked Lists

Linked Lists

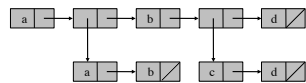
Flat Lists:

$[a, b, c, d]$

Nested Lists:

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

Linked List:



A list is finite sequence of objects. Lists can be flat, like the first list shown here. Lists can also be nested within other lists, as in our second example. A linked list is a way of representing nested lists of variable length. Each element is represented by a cell containing two items. The first is either an object or a nested list. The second item is either empty or is the remainder of the list. For example, the structure shown here is a linked list representation of our nested list.

Our goal in this example is to formalize linked lists and define some useful relations on such structures.

Representation

Example:



Representation as a functional term:

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

The first step is representation. To talk about linked lists, we use the binary function constant `cons` and we use the object constant `nil` to refer to the empty list. With this function constant, we can encode the list $[a, b, c, d]$ as shown here. Cons of `a` onto cons of `b` onto cons of `c` onto cons of `d` onto `nil`.

Signature

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

Binary Function Constant: *cons*

Binary Relation Constant: *member*

Ternary Relation Constant: *append*

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

Our signature for this example consists of some objects constants - `a`, `b`, `c`, `d` - to denote atomic elements and a single constant `nil` to represent the empty list. We have a single binary function constant `cons`. And we will define three relation constants, `member`, `among`, and `append`.

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?

We start with the member relation, which holds of an object and a list if the object is a top-level member of the list. For example, b is a member of the list consisting of a and b and c.

Using the function constant cons, we can define the member relation as shown here. The first sentence states that an object is a member of a list if it is the first member. The second sentence states that an object is a member of a list if it is a member of the rest of the list.

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?

Next, let's look at concatenation.. The result of appending the lists [a,b] and [c,d] would be [a,b,c,d]. Once again, our definition has consists of two sentences. The first sentence says that appending an empty list to any list results in that list. The second sentence says that appending a list with x as first element and y as remainder to a list z results in the list with x as first element with the remainder being the result of appending the y to z.

It is worth noting that there is a similarity between this definition and the definition of plus in Peano arithmetic. Here, we have an identity nil instead of 0, and here we have cons instead of s. However, the definitions are very similar.

Fitch Proofs

Propositional Logic

$$\{m \Rightarrow p \vee q, p \Rightarrow q\} \models m \Rightarrow q?$$

m	p	q	$m \Rightarrow p \vee q$	$p \Rightarrow q$	$m \Rightarrow q$
1	1	1	1	1	1
1	1	0	1	0	0
1	0	1	1	1	1
1	0	0	0	1	0
0	1	1	1	1	1
0	1	0	1	0	1
0	0	1	1	1	1
0	0	0	1	1	1

48

As in Propositional Logic, it is in principle possible to build a truth table for any set of sentences in Relational Logic. This truth table can then be used to determine validity, satisfiability, and so forth or to determine logical entailment and logical equivalence.

As an example, let us assume we have a language with just two object constants a and b and two relation constants p and q . Now consider the sentences shown below, and assume we want to know whether these sentences logically entail $\exists x.q(x)$.

$$p(a) \vee p(b)$$

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

A truth table for this problem is shown below. Each of the first four columns represents one of the elements of the Herbrand base for this language. The two middle columns represent our premises, and the final column represents the conclusion.

Looking at the table, we see that there are 12 truth assignments that make the first premise true and nine that make the second premise true and five that make them both true (rows 1, 5, 6, 9, and 11). Note that every truth assignment that makes both premises true also makes the conclusion true. Hence, the premises logically entail the conclusion.

Relational Logic

$$\{p(a) \vee p(b), \forall x.(p(x) \Rightarrow q(x))\} \models \exists x.q(x)?$$

$p(a)$	$p(b)$	$q(a)$	$q(b)$	$p(a) \vee p(b)$	$\forall x.(p(x) \Rightarrow q(x))$	$\exists x.q(x)$
1	1	1	1	1	1	1
1	1	1	0	1	0	1
1	1	0	1	1	0	1
1	1	0	0	1	0	0
1	0	1	1	1	1	1
1	0	1	0	1	1	1
1	0	0	1	1	0	1
1	0	0	0	1	0	0
0	1	1	1	1	1	1
0	1	1	0	1	0	1
0	1	0	1	1	1	1
0	1	0	0	1	0	0
0	0	1	1	0	1	1
0	0	1	0	0	1	1
0	0	0	1	0	1	1
0	0	0	0	0	1	0

As in Propositional Logic, it is in principle possible to build a truth table for any set of sentences in Relational Logic. This truth table can then be used to determine validity, satisfiability, and so forth or to determine logical entailment and logical equivalence.

As an example, let us assume we have a language with just two object constants a and b and two relation constants p and q . Now consider the sentences shown below, and assume we want to know whether these sentences logically entail $\exists x.q(x)$.

$$p(a) \vee p(b)$$

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

A truth table for this problem is shown below. Each of the first four columns represents one

of the elements of the Herbrand base for this language. The two middle columns represent our premises, and the final column represents the conclusion.

Looking at the table, we see that there are 12 truth assignments that make the first premise true and nine that make the second premise true and five that make them both true (rows 1, 5, 6, 9, and 11). Note that every truth assignment that makes both premises true also makes the conclusion true. Hence, the premises logically entail the conclusion.

Functional Logic

Object constants: n
 Unary relation constants: k
 Factoids in Herbrand Base: *infinite*
 Interpretations: *infinite*

50

Logical Rules of Inference

Negation Introduction
 Negation Elimination

And Introduction
 And Elimination

Or Introduction
 Or Elimination

Assumption
 Implication Elimination

Biconditional Introduction
 Biconditional Elimination

In the Fitch system for Relational Logic, we include all ten of the rules of inference for Propositional Logic. We have introduction and elimination rules for conjunctions, disjunctions, implications, and biconditionals.

Rules of Inference for Quantifiers

Universal Introduction
Universal Elimination

Existential Introduction
Existential Elimination

Domain Closure

Induction (Linear, Tree, Structural) *New!!*

To these rules of inference for logical operators, we add five new rules for quantified sentences - introduction and elimination rules for both universal and existential sentences and domain closure. Induction is new and is described in next lecture.

Example - Spin

Problem

On transition, a particle goes from spin zero to spin positive or spin negative and then goes to spin zero on second transition. Show that if it has spin zero, it will have spin zero after two transitions.

Givens:

$\forall x: (zero(x) \Rightarrow pos(f(x)) \vee neg(f(x)))$

$\forall x: (pos(x) \Rightarrow zero(f(x)))$

$\forall x: (neg(x) \Rightarrow zero(f(x)))$

Goal:

$\forall x: (zero(x) \Rightarrow zero(f(f(x))))$

As an illustration of these concepts, consider the following problem. Suppose we believe that everybody loves somebody. And suppose we believe that everyone loves a lover. Our job is to prove that Jack loves Jill.

Proof

1. $\forall x.(zero(x) \Rightarrow pos(f(x)) \vee neg(f(x)))$ Premise
2. $\forall x.(pos(x) \Rightarrow zero(f(x)))$ Premise
3. $\forall x.(neg(x) \Rightarrow zero(f(x)))$ Premise

Our premises appear on lines 1 and 2. The sentence on line 3 is the result of applying Universal Elimination to the sentence on line 1, substituting jill for y. The sentence on line 4 is the result of applying Universal Elimination to the sentence on line 2, substituting jack for x. The sentence on line 5 is the result of applying Universal Elimination to the sentence on line 4, substituting jill for y. Finally, we apply Implication Elimination to produce our conclusion on line 6.

Proof

1. $\forall x.(zero(x) \Rightarrow pos(f(x)) \vee neg(f(x)))$ Premise
2. $\forall x.(pos(x) \Rightarrow zero(f(x)))$ Premise
3. $\forall x.(neg(x) \Rightarrow zero(f(x)))$ Premise
4. $| zero(c)$ Assumption

Our premises appear on lines 1 and 2. The sentence on line 3 is the result of applying Universal Elimination to the sentence on line 1, substituting jill for y. The sentence on line 4 is the result of applying Universal Elimination to the sentence on line 2, substituting jack for x. The sentence on line 5 is the result of applying Universal Elimination to the sentence on line 4, substituting jill for y. Finally, we apply Implication Elimination to produce our conclusion on line 6.

Proof

1. $\forall x.(zero(x) \Rightarrow pos(f(x)) \vee neg(f(x)))$ Premise
2. $\forall x.(pos(x) \Rightarrow zero(f(x)))$ Premise
3. $\forall x.(neg(x) \Rightarrow zero(f(x)))$ Premise
4. $| zero(c)$ Assumption
5. $| zero(c) \Rightarrow pos(f(c)) \vee neg(f(c))$ UE: 1

Our premises appear on lines 1 and 2. The sentence on line 3 is the result of applying Universal Elimination to the sentence on line 1, substituting jill for y. The sentence on line 4 is the result of applying Universal Elimination to the sentence on line 2, substituting jack for x. The sentence on line 5 is the result of applying Universal Elimination to the sentence on line 4, substituting jill for y. Finally, we apply Implication Elimination to produce our conclusion on line 6.

Proof

- | | |
|---|------------|
| 1. $\forall x.(zero(x) \Rightarrow pos(f(x)) \vee neg(f(x)))$ | Premise |
| 2. $\forall x.(pos(x) \Rightarrow zero(f(x)))$ | Premise |
| 3. $\forall x.(neg(x) \Rightarrow zero(f(x)))$ | Premise |
| 4. $zero(c)$ | Assumption |
| 5. $zero(c) \Rightarrow pos(f(c)) \vee neg(f(c))$ | UE: 1 |
| 6. $pos(f(c)) \vee neg(f(c))$ | IE: 5, 4 |

Our premises appear on lines 1 and 2. The sentence on line 3 is the result of applying Universal Elimination to the sentence on line 1, substituting jill for y. The sentence on line 4 is the result of applying Universal Elimination to the sentence on line 2, substituting jack for x. The sentence on line 5 is the result of applying Universal Elimination to the sentence on line 4, substituting jill for y. Finally, we apply Implication Elimination to produce our conclusion on line 6.

Proof

- | | |
|---|------------|
| 1. $\forall x.(zero(x) \Rightarrow pos(f(x)) \vee neg(f(x)))$ | Premise |
| 2. $\forall x.(pos(x) \Rightarrow zero(f(x)))$ | Premise |
| 3. $\forall x.(neg(x) \Rightarrow zero(f(x)))$ | Premise |
| 4. $zero(c)$ | Assumption |
| 5. $zero(c) \Rightarrow pos(f(c)) \vee neg(f(c))$ | UE: 1 |
| 6. $pos(f(c)) \vee neg(f(c))$ | IE: 5, 4 |
| 7. $pos(f(c)) \Rightarrow zero(f(f(c)))$ | UE: 2 |
| 8. $neg(f(c)) \Rightarrow zero(f(f(c)))$ | UE: 3 |

Our premises appear on lines 1 and 2. The sentence on line 3 is the result of applying Universal Elimination to the sentence on line 1, substituting jill for y. The sentence on line 4 is the result of applying Universal Elimination to the sentence on line 2, substituting jack for x. The sentence on line 5 is the result of applying Universal Elimination to the sentence on line 4, substituting jill for y. Finally, we apply Implication Elimination to produce our conclusion on line 6.

Proof

- | | |
|---|-------------|
| 1. $\forall x.(zero(x) \Rightarrow pos(f(x)) \vee neg(f(x)))$ | Premise |
| 2. $\forall x.(pos(x) \Rightarrow zero(f(x)))$ | Premise |
| 3. $\forall x.(neg(x) \Rightarrow zero(f(x)))$ | Premise |
| 4. $zero(c)$ | Assumption |
| 5. $zero(c) \Rightarrow pos(f(c)) \vee neg(f(c))$ | UE: 1 |
| 6. $pos(f(c)) \vee neg(f(c))$ | IE: 5, 4 |
| 7. $pos(f(c)) \Rightarrow zero(f(f(c)))$ | UE: 2 |
| 8. $neg(f(c)) \Rightarrow zero(f(f(c)))$ | UE: 3 |
| 9. $zero(f(f(c)))$ | OE: 6, 7, 8 |

Our premises appear on lines 1 and 2. The sentence on line 3 is the result of applying Universal Elimination to the sentence on line 1, substituting jill for y. The sentence on line 4 is the result of applying Universal Elimination to the sentence on line 2, substituting jack for x. The sentence on line 5 is the result of applying Universal Elimination to the sentence on line 4, substituting jill for y. Finally, we apply Implication Elimination to produce our conclusion on line 6.

Proof

1.	$\forall x.(zero(x) \Rightarrow pos(f(x)) \vee neg(f(x)))$	Premise
2.	$\forall x.(pos(x) \Rightarrow zero(f(x)))$	Premise
3.	$\forall x.(neg(x) \Rightarrow zero(f(x)))$	Premise
4.	$zero(c)$	Assumption
5.	$zero(c) \Rightarrow pos(f(c)) \vee neg(f(c))$	UE: 1
6.	$pos(f(c)) \vee neg(f(c))$	IE: 5, 4
7.	$pos(f(c)) \Rightarrow zero(f(f(c)))$	UE: 2
8.	$neg(f(c)) \Rightarrow zero(f(f(c)))$	UE: 3
9.	$zero(f(f(c)))$	OE: 6, 7, 8
10.	$zero(c) \Rightarrow zero(f(f(c)))$	II: 4, 9

Our premises appear on lines 1 and 2. The sentence on line 3 is the result of applying Universal Elimination to the sentence on line 1, substituting jill for y. The sentence on line 4 is the result of applying Universal Elimination to the sentence on line 2, substituting jack for x. The sentence on line 5 is the result of applying Universal Elimination to the sentence on line 4, substituting jill for y. Finally, we apply Implication Elimination to produce our conclusion on line 6.

Proof

1.	$\forall x.(zero(x) \Rightarrow pos(f(x)) \vee neg(f(x)))$	Premise
2.	$\forall x.(pos(x) \Rightarrow zero(f(x)))$	Premise
3.	$\forall x.(neg(x) \Rightarrow zero(f(x)))$	Premise
4.	$zero(c)$	Assumption
5.	$zero(c) \Rightarrow pos(f(c)) \vee neg(f(c))$	UE: 1
6.	$pos(f(c)) \vee neg(f(c))$	IE: 5, 4
7.	$pos(f(c)) \Rightarrow zero(f(f(c)))$	UE: 2
8.	$neg(f(c)) \Rightarrow zero(f(f(c)))$	UE: 3
9.	$zero(f(f(c)))$	OE: 6, 7, 8
10.	$zero(c) \Rightarrow zero(f(f(c)))$	II: 4, 9
11.	$\forall x.(zero(x) \Rightarrow zero(f(f(x))))$	UI: 10

Our premises appear on lines 1 and 2. The sentence on line 3 is the result of applying Universal Elimination to the sentence on line 1, substituting jill for y. The sentence on line 4 is the result of applying Universal Elimination to the sentence on line 2, substituting jack for x. The sentence on line 5 is the result of applying Universal Elimination to the sentence on line 4, substituting jill for y. Finally, we apply Implication Elimination to produce our conclusion on line 6.

Analysis

As we have seen Relational Logic is very well-behaved. All questions are decidable. What happens when we relax our restrictions and look at Term Logic (TL)?

Logical Entailment and Provability

A set of premises Δ *logically entails* a conclusion ϕ ($\Delta \models \phi$) if and only if every truth assignment that satisfies Δ also satisfies ϕ .

If there exists a *finite* proof of a sentence ϕ from a set Δ of premises using the rules of inference in R , we say that ϕ is *provable* from Δ using R (written $\Delta \vdash_R \phi$).

A reminder that, in talking about Logic, we have two notions - logical entailment and provability. A set of premises logically entails a conclusion if and only if every truth assignment that satisfies the premises also satisfies the conclusion. A sentence is provable from a set of premises if and only if there is a finite proof of the conclusion from the premises. The concepts are quite different. One is based on truth assignments; the other is based on symbolic manipulation of expressions.

Soundness and Completeness

A proof system is *sound* if and only if every provable conclusion is logically entailed.

If $\Delta \vdash \phi$, then $\Delta \models \phi$.

A proof system is *complete* if and only if every logical conclusion is provable.

If $\Delta \models \phi$, then $\Delta \vdash \phi$.

A proof system is sound if and only if every provable conclusion is logically entailed. In other words, if $\Delta \vdash \phi$, then $\Delta \models \phi$. We say that a proof system is complete if and only if every logical conclusion is provable. In other words, if $\Delta \models \phi$, then $\Delta \vdash \phi$.

Fitch Proof System

Theorem: Fitch is sound and complete for **Relational Logic**.

$\Delta \models \phi$ if and only if $\Delta \vdash_{\text{Fitch}} \phi$.

Theorem: Fitch is *sound* for **Term Logic** but it is *not complete*.

if $\Delta \vdash_{\text{Fitch}} \phi$, then $\Delta \models \phi$.

Bad News

Theorem: Satisfiability / logical entailment for TL are not decidable.

Theorem: There is *no* sound and complete proof procedure for Term Logic.

The bad news is that the question of unsatisfiability for FL is not even semidecidable. Explaining this in detail is beyond the scope of this course. However, we can give a line of argument that suggests why it is true. The argument reduces the question of decidability of satisfiability / logical entailment for Functional Logic to a question that is broadly accepted to be undecidable. If our logic **were** decidable, then this other question would be decidable; and, since it is known to be undecidable, Logical entailment must not be decidable either.

Diophantine Equations or halting problem

Diophantine equation:

$$3x^2 = 1$$

Diophantine equation in Peano Arithmetic:

$$\exists x. \exists y. (\text{times}(x, x, y) \wedge \text{times}(s(s(s(0))), y, s(0)))$$

Solvability Question:

Axioms of Arithmetic $\cup$

$$\{\exists x. \exists y. (\text{times}(x, x, y) \wedge \text{times}(s(s(s(0))), y, s(0)))\}$$

It is known that the question of solvability of Diophantine equations is *not* decidable.

As we know Diophantine equations can be readily expressed as sentences in FL. For example, we can represent the solvability of Diophantine equation $3x^2=1$ with the sentence $\exists x, y, ix \text{ times } x \text{ is } y \text{ and } 3 \text{ times } y \text{ is } 1$. We can represent every Diophantine in an analogous way. We can express the unsolvability of a Diophantine equation by negating the corresponding sentence. We can then ask the question of whether the axioms of arithmetic logically entail this negation or, equivalently, whether the axioms of Arithmetic together with the unnegated sentence are unsatisfiable.

The problem is that it is well known that determining whether Natural Diophantine equations are unsolvable is not semidecidable. If we could determine the unsatisfiability of our FL encoding of a Diophantine equation, we could decide whether it is unsolvable, contradicting the non-semidecidability of that problem.

Two Types of Completeness

A *proof procedure* is complete if and only if everything that is logically entailed is provable.

A *set of sentences* Δ is complete if and only if for every sentence ϕ , either $\Delta \models \phi$ or $\Delta \models \neg \phi$.

Explanation

Given a finite set of sentences and a complete proof procedure, logical entailment is semidecidable.

If a set of sentences is complete, then logical entailment is decidable. (We try to prove the sentence and its negation, and one of the two will terminate if complete.)

Our axiomatization of Peano Arithmetic in Term Logic is finite and complete. **Assume** we had a sound and complete proof procedure. Then we would be able to decide the satisfiability of any Diophantine equation. **Contradiction.**
Upshot: there is no sound and complete proof procedure for Term Logic.

Compactness

A logic is *compact* if and only if every unsatisfiable set of sentences has a finite subset that is unsatisfiable.

Theorem: Propositional Logic is compact.

Theorem: Relational Logic is compact.

First of all, there is compactness. A logic is compact if and only if every unsatisfiable set of sentences has a finite subset that is unsatisfiable.

Relational Logic with Tarskian semantics turns out to be compact. The upshot is that it is possible to demonstrate unsatisfiability in finite space; alternatively, all proofs are finite.

By contrast, Herbrand Logic is not compact - there are infinite sets of sentences that are unsatisfiable while every finite subset is satisfiable. Consider the set of sentences shown here. It is clearly unsatisfiable under Herbrand semantics; but, if we remove any one sentence, it becomes satisfiable.

The upshot of this is that there are infinite sets of sentences where we cannot demonstrate unsatisfiability with a finite argument within the language itself. Fortunately, this does not cause any practical difficulties, since in all cases of practical interest we are working with finite sets of premises.

But...

A logic is *compact* if and only if every unsatisfiable set of sentences has a finite subset that is unsatisfiable.

Theorem: Propositional Logic is compact.

Theorem: Relational Logic is compact.

Theorem: Term Logic is *not* compact.

$$\{p(0), p(s(0)), p(s(s(0))), \dots, \exists x. \neg p(x)\}$$

Significance: *Some conclusions in Term Logic have only infinite proofs.*

First of all, there is compactness. A logic is compact if and only if every unsatisfiable set of sentences has a finite subset that is unsatisfiable.

Relational Logic with Tarskian semantics turns out to be compact. The upshot is that it is possible to demonstrate unsatisfiability in finite space; alternatively, all proofs are finite.

By contrast, Herbrand Logic is not compact - there are infinite sets of sentences that are unsatisfiable while every finite subset is satisfiable. Consider the set of sentences shown here. It is clearly unsatisfiable under Herbrand semantics; but, if we remove any one sentence, it becomes satisfiable.

The upshot of this is that there are infinite sets of sentences where we cannot demonstrate unsatisfiability with a finite argument within the language itself. Fortunately, this does not cause any practical difficulties, since in all cases of practical interest we are working with finite sets of premises.

First-Order Logic (FOL)

First-Order Logic is logic with the same syntax as Term Logic but with a different semantics.

FOL is compact.

FOL has a sound and complete proof procedure!

FOL is (almost exclusively) taught in courses on Logic.

However, this is not as bad as it seems. It turns out that everything that is true under Tarskian semantics is also true under Herbrand semantics, so we can use the same rules of inference. The upshot here is that we lost nothing by switching to Herbrand semantics. In fact, we can add some additional rules of inference. It is not that Herbrand logic is weaker. In fact, it is stronger. There are more things that are true. We cannot prove them all, but we can prove everything we could prove before.

Some may be disturbed by the fact that Herbrand entailment is not semi-decidable. But a similar argument could be leveled against Tarskian semantics. Semi-decidability is not that great either, as a proof procedure might still run forever if the proposed conclusion does not follow from the premises.

But...

There are things we can express in Term Logic that cannot be expressed in FOL, e.g. Peano Arithmetic, transitive closure.

Even though there is no complete proof procedure, Term Logic is *not* inferentially weaker than First-Order Logic. In fact, it is *stronger* than First-Order Logic.

If we can prove a conclusion in FOL, we can prove the result in Term Logic. In fact, using induction, we can prove *more* things in Term Logic than in FOL.

The issue is that there are *more* things that are true. We just cannot prove them all.

However, this is not as bad as it seems. It turns out that everything that is true under Tarskian semantics is also true under Herbrand semantics, so we can use the same rules of inference. The upshot here is that we lost nothing by switching to Herbrand semantics. In fact, we can add some additional rules of inference. It is not that Herbrand logic is weaker. In fact, it is stronger. There are more things that are true. We cannot prove them all, but we can prove everything we could prove before.

Some may be disturbed by the fact that Herbrand entailment is not semi-decidable. But a similar argument could be leveled against Tarskian semantics. Semi-decidability is not that great either, as a proof procedure might still run forever if the proposed conclusion does not follow from the premises.

