

Introduction to Logic

Induction

Michael Genesereth
Computer Science Department
Stanford University

Truth Table for 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.

Relational Logic Sizes

Object constants: n
Binary relation constants: k
Factoids in Herbrand Base: $k * n^2$
Truth Assignments: $2^{k * n^2}$

Object constants: 4
Binary relation constants: 4
Factoids in Herbrand Base: 64
Truth Assignments: $2^{64} = 18,446,744,073,709,551,616$

*Upshot: Truth tables impractical.
Proofs more practical.*

Term Logic Sizes

Object constants: $n \geq 1$
Functions constants: $j \geq 1$
Relation constants: $k \geq 1$
Factoids in Herbrand Base: *countably infinite*
Truth Assignments: *uncountably infinite*

*Upshot: Truth tables impossible.
Proofs are our only hope.*

Fitch System for Relational Logic

Negation Introduction	Universal Introduction
Negation Elimination	Universal Elimination
And Introduction	Existential Introduction
And Elimination	Existential Elimination
Or Introduction	Domain Closure
Or Elimination	
Assumption	
Implication Introduction	
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.

Fitch System for Term Logic

Negation Introduction	Universal Introduction
Negation Elimination	Universal Elimination
And Introduction	Existential Introduction
And Elimination	Existential Elimination
Or Introduction	Domain Closure
Or Elimination	Induction <i>New!!</i>
Assumption	
Implication Introduction	
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.

Overview

Induction

Induction is reasoning from the specific to the general.

If various instances of a schema are true and there are no counterexamples, we are tempted to conclude a universally quantified version of the schema.

$$\begin{array}{l} p(a) \Rightarrow q(a) \\ p(b) \Rightarrow q(b) \\ p(c) \Rightarrow q(c) \end{array} \quad \rightarrow \quad \forall x.(p(x) \Rightarrow q(x))$$

Induction is reasoning from the specific to the general. If we believe many instances of a schema and we know of no counterexamples, then we are often tempted to conclude that the schema is true in general.

Suppose we believe that a schema holds for every term in our language, e.g. $p(a) \Rightarrow q(a)$, $p(b) \Rightarrow q(b)$, and so forth. Then, we are entitled to conclude a universally quantified version of our schema e.g. $\forall x.(p(x) \Rightarrow q(x))$.

Lucky Guess

Definition

$$f(1) = 1$$

$$f(x + 1) = f(x) + 2 * x + 1$$

Data

1	1	=	1	=	1 ²
2	1+3	=	4	=	2 ²
3	1+3+5	=	9	=	3 ²
4	1+3+5+7	=	16	=	4 ²
5	1+3+5+7+9	=	25	=	5 ²

Conjecture

$$f(x) = x^2$$

In this case, the answer is correct. Lucky Guess.

Incomplete induction is induction where the set of instances is not complete. From a reasonable collection of instances, we sometimes leap to a universal conclusion even though we have not seen all instances. Consider, for example, the function shown here. $F(1)=1$, and $f(n+1)=f(n) + 2n + 1$. If we look at some values of this function, we notice a certain regularity, the value of f always seems to be the square of its input. From this sample, we are tempted to leap to the conclusion that $f(n)=n^2$. Lucky guess. In this case, the conclusion happens to be true; and we can prove it.

Not So Lucky Guess

Data:

$$2^{2^1} + 1 = 2^4 + 1 = 5$$

$$2^{2^2} + 1 = 2^4 + 1 = 17$$

$$2^{2^3} + 1 = 2^8 + 1 = 257$$

$$2^{2^4} + 1 = 2^{16} + 1 = 65537$$

“Theorem” by Fermat (1601-1665):

$$\text{prime}(2^{2^i} + 1)$$

Fact discovered (mercifully) after his death:

$$2^{2^5} + 1 = 4,294,967,297 = 641 * 6,700,417$$

Oops.

Here is another example. This one is due to the mathematician Fermat. He looked at values of the expression $2^{2^n} + 1$ for various values of n and noticed that they were all prime. So, he concluded the value of the expression was prime number. Unfortunately, this was not a lucky guess. His conjecture was ultimately disproved, in fact with the very next number in the sequence.

Domain Closure

$$\frac{\begin{array}{l} \phi[\sigma_1] \\ \dots \\ \phi[\sigma_n] \end{array} \quad \left. \vphantom{\begin{array}{l} \phi[\sigma_1] \\ \dots \\ \phi[\sigma_n] \end{array}} \right\} \begin{array}{l} \text{every} \\ \text{ground} \\ \text{term} \end{array}}{\forall v. \phi[v]}$$

Induction for finite languages is trivial. We simply use the Domain Closure rule of inference. For a language with object constants $\sigma_1, \dots, \sigma_n$, the rule is written as shown here. If we believe a schema is true for every instance, then we can infer a universally quantified version of that schema.

Finite Example

$$\begin{array}{l} p(a) \Rightarrow q(a) \\ p(b) \Rightarrow q(b) \\ p(c) \Rightarrow q(c) \\ p(d) \Rightarrow q(d) \\ \hline \forall x.(p(x) \Rightarrow q(x)) \end{array}$$

Suppose, for example, we have a language with just object constants and no function constants. Then we could write a domain closure rule as shown here. If a schema is true of a and b and c and d, then it is true of everything.

Infinite Case

What happens when we have infinitely many terms?

$$\begin{array}{l} p(a) \Rightarrow q(a) \\ p(s(a)) \Rightarrow q(s(a)) \\ p(s(s(a))) \Rightarrow q(s(s(a))) \\ p(s(s(s(a)))) \Rightarrow q(s(s(s(a)))) \\ \dots \\ \hline \forall x.(p(x) \Rightarrow q(x)) \end{array}$$

Unfortunately, this technique does not work when there are infinitely many ground terms. Suppose, for example, we have a language with infinitely many ground terms. We could write a domain closure rule analogous to the one we saw earlier. The rule would be sound in the sense that the conclusion is logically entailed by the premises of the rule. However, it would not help us produce a proof of the conclusion. To use a rule, we need to prove all of the rule's premises. Unfortunately, there are infinitely many premises. So, this rule cannot be used in generating a finite proof.

All is not lost. It is possible to write rules that cover all instances without enumerating them individually. The method depends on the structure of the language. The next sections describe how this can be done for languages with different structures.

Types of Induction

Linear Induction
induction on sequences

Tree Induction
induction on trees

Structural Induction
induction on complex structures

The techniques for complete induction vary with the structure of the language to which it is applied. We begin this lesson with a discussion of domain closure, a rule that applies when the Herbrand base of our language is finite. We then move on to linear induction, i.e. the special case in which the ground terms in the language form a linear sequence. After that, we look at tree induction, i.e. the special case in which the ground terms of the language form a tree. Finally, we look at structural induction, which applies to all languages.

Linear Induction

Linear Worlds

$$a \rightarrow s(a) \rightarrow s(s(a)) \rightarrow s(s(s(a))) \rightarrow \dots$$

Consider a language with a single object constant a and a single unary function constant s . There are infinitely many ground terms in this language, arranged in what resembles a straight line. Starting from the object constant, we move to a term in which we apply the function constant to that constant, then to a term in which we apply the function constant to that term, and so forth.

In this section, we concentrate on languages that have linear structure of this sort. Hereafter, we call these linear languages. In all cases, there is a single object constant and a single unary function constant. In talking about a linear language, we call the object constant the *base element* of the language, and we call the unary function constant the *successor function*.

Linear Worlds

Linear World:

$$a \rightarrow s(a) \rightarrow s(s(a)) \rightarrow s(s(s(a))) \rightarrow \dots$$

The object constant is called the *base element*, and the function constant is called the *successor function*.

Consider a language with a single object constant a and a single unary function constant s . There are infinitely many ground terms in this language, arranged in what resembles a straight line. Starting from the object constant, we move to a term in which we apply the function constant to that constant, then to a term in which we apply the function constant to that term, and so forth.

In this section, we concentrate on languages that have linear structure of this sort. Hereafter, we call these linear languages. In all cases, there is a single object constant and a single unary function constant. In talking about a linear language, we call the object constant the *base element* of the language, and we call the unary function constant the *successor function*.

Induction with Linear Worlds

Linear World:

$$a \rightarrow s(a) \rightarrow s(s(a)) \rightarrow s(s(s(a))) \rightarrow \dots$$

Linear Induction:

- (1) If we can show our *base element* has some property
and
(2) if we can show that, whenever an arbitrary element has that property, its *successor* has that property,
then we can conclude that *every* element has that property.

Consider a language with a single object constant a and a single unary function constant s . There are infinitely many ground terms in this language, arranged in what resembles a straight line. Starting from the object constant, we move to a term in which we apply the function constant to that constant, then to a term in which we apply the function constant to that term, and so forth.

In this section, we concentrate on languages that have linear structure of this sort. Hereafter, we call these linear languages. In all cases, there is a single object constant and a single unary function constant. In talking about a linear language, we call the object constant the base element of the language, and we call the unary function constant the successor function.

Dominoes



Dominoes Example

Object constant: a
Unary function constant: s
Unary relation constant: *falls*

Now, let's formalize this example and prove this result using induction. Let a represent the first domino. And let s represent the relationship between a domino and the next domino in line. Let's use the unary relation constant *falls* to express the fact that the domino given as argument eventually falls.

Dominoes Example

Object constant: a
Unary function constant: s
Unary relation constant: $falls$

Axioms:

$$falls(a) \\ \forall x.(falls(x) \Rightarrow falls(s(x)))$$

Here are our axioms. Suppose we know the first domino falls. We also know that whenever any domino falls, its successor falls.

Dominoes Example

Object constant: a
Unary function constant: s
Unary relation constant: $falls$

Axioms:

$$falls(a) \\ \forall x.(falls(x) \Rightarrow falls(s(x)))$$

Conclusion:

$$\forall x.falls(x)$$

Our goal is to prove that all of the dominoes fall.

Here is a one-step proof of our conclusion that all of the dominoes eventually. We start with the premise that the first domino in the line falls. We also have the premise that the dominoes are so placed so that each domino knocks over the next domino. These two sentences match the premises of the linear induction rule, and so we can derive the conclusion.





Linear Induction

$$\frac{\phi[a] \quad \forall x.(\phi[x] \Rightarrow \phi[s(x)])}{\forall x.\phi[x]}$$

Although there are infinitely many ground terms in any linear language, we can still generate finite proofs that are guaranteed to be correct. The trick is to use the structure of the terms in the language in expressing the premises of an inductive rule of inference known as **linear induction**. In general, if we know that a schema holds of our base element and if we know that, whenever the schema holds of an element, it also holds of the successor of that element, then we can conclude that the schema holds of all elements.

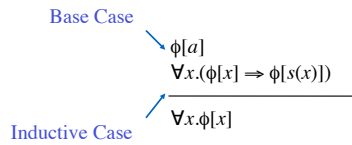
Linear Induction

Base Case

$$\frac{\phi[a] \quad \forall x.(\phi[x] \Rightarrow \phi[s(x)])}{\forall x.\phi[x]}$$

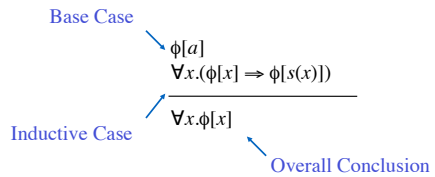
A bit of terminology before we go on. The first premise in this rule is called the base case of the induction, because it refers to the base element of the language.

Linear Induction



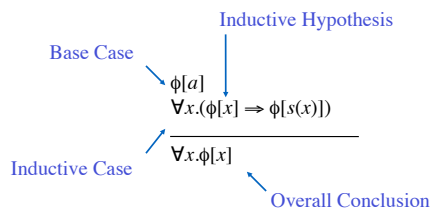
The second premise is called the inductive case.

Linear Induction



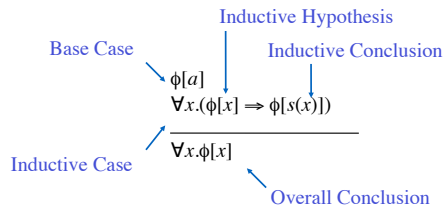
The conclusion of the rule as a whole is called the overall conclusion.

Linear Induction



The antecedent of the inductive case is called the inductive hypothesis. , and the consequent is called, not surprisingly, the inductive conclusion.

Linear Induction



The antecedent of the inductive case is called the inductive hypothesis. , and the consequent is called, not surprisingly, the inductive conclusion.

Whole Numbers

Object constant: 0

Unary function constant: s

$$0 \rightarrow s(0) \rightarrow s(s(0)) \rightarrow s(s(s(0))) \rightarrow \dots$$

Here is a more interesting example. Recall the formalization of Arithmetic introduced earlier in the course. Using the object constant 0 and the unary function constant s , we can represent each number n by applying the function constant to 0 exactly n times. We also have a single ternary relation constant $plus$ to represent the addition table.

The axioms shown here describe $plus$ in terms of 0 and s . The first sentence here says that adding 0 to any element produces that element. The second sentence states that adding the successor of a number to another number yields the successor of their sum.

Whole Numbers

Object constant: 0

Unary function constant: s

$$0 \rightarrow s(0) \rightarrow s(s(0)) \rightarrow s(s(s(0))) \rightarrow \dots$$

Unary relation constants: *even*, *odd*

Here is a more interesting example. Recall the formalization of Arithmetic introduced earlier in the course. Using the object constant 0 and the unary function constant s , we can represent each number n by applying the function constant to 0 exactly n times. We also have a single ternary relation constant $plus$ to represent the addition table.

The axioms shown here describe $plus$ in terms of 0 and s . The first sentence here says that adding 0 to any element produces that element. The second sentence states that adding the successor of a number to another number yields the successor of their sum.

Whole Numbers

Object constant: 0

Unary function constant: s

$$0 \rightarrow s(0) \rightarrow s(s(0)) \rightarrow s(s(s(0))) \rightarrow \dots$$

Unary relation constants: $even, odd$

Axioms:

$$\begin{aligned} &even(0) \\ &\forall x.(even(x) \Rightarrow odd(s(x))) \\ &\forall x.(odd(x) \Rightarrow even(s(x))) \end{aligned}$$

Here is a more interesting example. Recall the formalization of Arithmetic introduced earlier in the course. Using the object constant 0 and the unary function constant s , we can represent each number n by applying the function constant to 0 exactly n times. We also have a single ternary relation constant $plus$ to represent the addition table.

The axioms shown here describe $plus$ in terms of 0 and s . The first sentence here says that adding 0 to any element produces that element. The second sentence states that adding the successor of a number to another number yields the successor of their sum.

Whole Numbers

Object constant: 0

Unary function constant: s

$$0 \rightarrow s(0) \rightarrow s(s(0)) \rightarrow s(s(s(0))) \rightarrow \dots$$

Unary relation constants: $even, odd$

Axioms:

$$\begin{aligned} &even(0) \\ &\forall x.(even(x) \Rightarrow odd(s(x))) \\ &\forall x.(odd(x) \Rightarrow even(s(x))) \end{aligned}$$

Goal:

$$\forall x.(even(x) \vee odd(x))$$

Here is a more interesting example. Recall the formalization of Arithmetic introduced earlier in the course. Using the object constant 0 and the unary function constant s , we can represent each number n by applying the function constant to 0 exactly n times. We also have a single ternary relation constant $plus$ to represent the addition table.

The axioms shown here describe $plus$ in terms of 0 and s . The first sentence here says that adding 0 to any element produces that element. The second sentence states that adding the successor of a number to another number yields the successor of their sum.

Proof

- | | | |
|----|---|---------|
| 1 | $even(0)$ | Premise |
| 2. | $\forall x.(even(x) \Rightarrow odd(s(x)))$ | Premise |
| 3. | $\forall x.(odd(x) \Rightarrow even(s(x)))$ | Premise |

Finally, we use linear induction to derive our overall conclusion shown on line 19.

Proof		
1	$even(0)$	Premise
2.	$\forall x.(even(x) \Rightarrow odd(s(x)))$	Premise
3.	$\forall x.(odd(x) \Rightarrow even(s(x)))$	Premise
4.	$even(0) \vee odd(0)$	OI: 1

Finally, we use linear induction to derive our overall conclusion shown on line 19.

Proof		
1	$even(0)$	Premise
2.	$\forall x.(even(x) \Rightarrow odd(s(x)))$	Premise
3.	$\forall x.(odd(x) \Rightarrow even(s(x)))$	Premise
4.	$even(0) \vee odd(0)$	OI: 1
5.	$ even(c) \vee odd(c)$	Assumption

Finally, we use linear induction to derive our overall conclusion shown on line 19.

Proof		
1	$even(0)$	Premise
2.	$\forall x.(even(x) \Rightarrow odd(s(x)))$	Premise
3.	$\forall x.(odd(x) \Rightarrow even(s(x)))$	Premise
4.	$even(0) \vee odd(0)$	OI: 1
5.	$ even(c) \vee odd(c)$	Assumption
6.	$ even(c)$	Assumption

Finally, we use linear induction to derive our overall conclusion shown on line 19.

Proof		
1	$even(0)$	Premise
2.	$\forall x.(even(x) \Rightarrow odd(s(x)))$	Premise
3.	$\forall x.(odd(x) \Rightarrow even(s(x)))$	Premise
4.	$even(0) \vee odd(0)$	OI: 1
5.	$ even(c) \vee odd(c)$	Assumption
6.	$ even(c)$	Assumption
7.	$ even(c) \Rightarrow odd(s(c))$	UE: 2
8.	$ odd(s(c))$	IE: 7, 6

Finally, we use linear induction to derive our overall conclusion shown on line 19.

Proof		
1	$even(0)$	Premise
2.	$\forall x.(even(x) \Rightarrow odd(s(x)))$	Premise
3.	$\forall x.(odd(x) \Rightarrow even(s(x)))$	Premise
4.	$even(0) \vee odd(0)$	OI: 1
5.	$ even(c) \vee odd(c)$	Assumption
6.	$ even(c)$	Assumption
7.	$ even(c) \Rightarrow odd(s(c))$	UE: 2
8.	$ odd(s(c))$	IE: 7, 6
9.	$ even(s(c)) \vee odd(s(c))$	OI: 8
10.	$ even(c) \Rightarrow even(s(c)) \vee odd(s(c))$	II: 6, 9

Finally, we use linear induction to derive our overall conclusion shown on line 19.

Proof		
1	$even(0)$	Premise
2.	$\forall x.(even(x) \Rightarrow odd(s(x)))$	Premise
3.	$\forall x.(odd(x) \Rightarrow even(s(x)))$	Premise
4.	$even(0) \vee odd(0)$	OI: 1
5.	$ even(c) \vee odd(c)$	Assumption
	$ \dots$	
10.	$ even(c) \Rightarrow even(s(c)) \vee odd(s(c))$	II: 6, 9

Finally, we use linear induction to derive our overall conclusion shown on line 19.

Proof		
1	$even(0)$	Premise
2.	$\forall x.(even(x) \Rightarrow odd(s(x)))$	Premise
3.	$\forall x.(odd(x) \Rightarrow even(s(x)))$	Premise
4.	$even(0) \vee odd(0)$	OI: 1
5.	$ even(c) \vee odd(c)$	Assumption
	$ \dots$	
10.	$ even(c) \Rightarrow even(s(c)) \vee odd(s(c))$	II: 6, 9
11.	$ odd(c)$	Assumption

Finally, we use linear induction to derive our overall conclusion shown on line 19.

Proof		
1	$even(0)$	Premise
2.	$\forall x.(even(x) \Rightarrow odd(s(x)))$	Premise
3.	$\forall x.(odd(x) \Rightarrow even(s(x)))$	Premise
4.	$even(0) \vee odd(0)$	OI: 1
5.	$ even(c) \vee odd(c)$	Assumption
	$ \dots$	
10.	$ even(c) \Rightarrow even(s(c)) \vee odd(s(c))$	II: 6, 9
11.	$ odd(c)$	Assumption
12.	$ odd(c) \Rightarrow even(s(c))$	UI: 3
13.	$ even(s(c))$	IE: 12, 11
14.	$ even(s(c)) \vee odd(s(c))$	OI: 13
15.	$ odd(c) \Rightarrow even(s(c)) \vee odd(s(c))$	II: 11, 15

Finally, we use linear induction to derive our overall conclusion shown on line 19.

Proof		
1	$even(0)$	Premise
2.	$\forall x.(even(x) \Rightarrow odd(s(x)))$	Premise
3.	$\forall x.(odd(x) \Rightarrow even(s(x)))$	Premise
4.	$even(0) \vee odd(0)$	OI: 1
5.	$ even(c) \vee odd(c)$	BE: 4
	$ \dots$	
10.	$ even(c) \Rightarrow even(s(c)) \vee odd(s(c))$	II: 6,9
	$ \dots$	
15.	$ odd(c) \Rightarrow even(s(c)) \vee odd(s(c))$	II: 11,14

Proof		
1	$even(0)$	Premise
2.	$\forall x.(even(x) \Rightarrow odd(s(x)))$	Premise
3.	$\forall x.(odd(x) \Rightarrow even(s(x)))$	Premise
4.	$even(0) \vee odd(0)$	OI: 1
5.	$even(c) \vee odd(c)$	BE: 4
	...	
10.	$even(c) \Rightarrow even(s(c)) \vee odd(s(c))$	II: 6,9
	...	
15.	$odd(c) \Rightarrow even(s(c)) \vee odd(s(c))$	II: 11,14
16.	$even(s(c)) \vee odd(s(c))$	OE: 5, 10, 15

Proof		
1	$even(0)$	Premise
2.	$\forall x.(even(x) \Rightarrow odd(s(x)))$	Premise
3.	$\forall x.(odd(x) \Rightarrow even(s(x)))$	Premise
4.	$even(0) \vee odd(0)$	OI: 1
5.	$even(c) \vee odd(c)$	BE: 4
	...	
10.	$even(c) \Rightarrow even(s(c)) \vee odd(s(c))$	II: 6,9
	...	
15.	$odd(c) \Rightarrow even(s(c)) \vee odd(s(c))$	II: 11,14
16.	$even(s(c)) \vee odd(s(c))$	OE: 5, 10, 15
17.	$even(c) \vee odd(c) \Rightarrow even(s(c)) \vee odd(s(c))$	II: 5, 16

Proof		
1	$even(0)$	Premise
2.	$\forall x.(even(x) \Rightarrow odd(s(x)))$	Premise
3.	$\forall x.(odd(x) \Rightarrow even(s(x)))$	Premise
4.	$even(0) \vee odd(0)$	OI: 1
5.	$even(c) \vee odd(c)$	BE: 4
	...	
10.	$even(c) \Rightarrow even(s(c)) \vee odd(s(c))$	II: 6,9
	...	
15.	$odd(c) \Rightarrow even(s(c)) \vee odd(s(c))$	II: 11,14
16.	$even(s(c)) \vee odd(s(c))$	OE: 5, 10, 15
17.	$even(c) \vee odd(c) \Rightarrow even(s(c)) \vee odd(s(c))$	II: 5, 16
18.	$\forall x.(even(x) \vee odd(x) \Rightarrow even(s(x)) \vee odd(s(x)))$	UI: 17

Proof

1	$even(0)$	Premise
2.	$\forall x.(even(x) \Rightarrow odd(s(x)))$	Premise
3.	$\forall x.(odd(x) \Rightarrow even(s(x)))$	Premise
4.	$even(0) \vee odd(0)$	OI: 1
5.	$even(c) \vee odd(c)$	BE: 4
	...	
10.	$even(c) \Rightarrow even(s(c)) \vee odd(s(c))$	II: 6,9
	...	
15.	$odd(c) \Rightarrow even(s(c)) \vee odd(s(c))$	II: 11,14
16.	$even(s(c)) \vee odd(s(c))$	OE: 5, 10, 15
17.	$even(c) \vee odd(c) \Rightarrow even(s(c)) \vee odd(s(c))$	II: 5, 16
18.	$\forall x.(even(x) \vee odd(x) \Rightarrow even(s(x)) \vee odd(s(x)))$	UI: 17
19.	$\forall x.(even(x) \vee odd(x))$	Ind: 4, 18

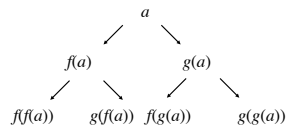
Finally, we use linear induction to derive our overall conclusion shown on line 19.

Tree Induction

Tree languages are a superset of linear languages. While, in linear languages, the terms in the language form a linear sequence, in tree languages the structure is more tree-like.

Tree Languages

Tree-Like World:



Languages like this are called *tree languages*.

Consider a language with an object constant a and two unary function constants f and g . Some of the terms in the language are shown here.

Tree Induction

$$\begin{array}{l} \phi[a] \\ \forall \mu. (\phi[\mu] \Rightarrow \phi[f(\mu)]) \\ \forall \mu. (\phi[\mu] \Rightarrow \phi[g(\mu)]) \\ \hline \forall v. \phi[v] \end{array}$$

As with linear languages, we can write an inductive rule of inference for tree languages. The tree induction rule of inference for the language just described is shown here. Suppose we know that a schema ϕ holds of a . Suppose we know that, whenever the schema holds of any element, it holds of the term formed by applying f to that element. And suppose we know that, whenever the schema holds of any element, it holds of the term formed by applying g to that element. Then, we can conclude that the schema holds of every element.

Canine Ancestry Example

Object constant: *rex*
Unary function constants: *f*, *g*
Unary relation constant: *p*

In order to see an example of tree induction in action, consider the ancestry tree for a particular dog. We use the object constant *rex* to refer to the dog; we use the unary function constant *f* map an arbitrary dog to its father; and we use *g* map a dog to its mother. Finally, we have a single unary relation constant *p* that is true of a dog if and only if it is purebred.

Canine Ancestry Example

Object constant: *rex*
Unary function constants: *f*, *g*
Unary relation constant: *p*

Axioms:

$$\begin{array}{l} p(\text{rex}) \\ \forall x. (p(x) \Leftrightarrow p(f(x)) \wedge p(g(x))) \end{array}$$

Now, we write down the fundamental rule of dog breeding - we say that a dog is purebred if and only if both its father and its mother are purebred. See below. (This is a bit oversimplified on several grounds. Properly, the father and mother should be of the same breed. Also, this formalization suggests that every dog has an ancestry tree that stretches back without end. However, let's ignore these imperfections for the purposes of our example.)

Canine Ancestry Example

Object constant: *rex*
Unary function constants: *f*, *g*
Unary relation constant: *p*

Axioms:

$$\forall x.(p(x) \Leftrightarrow p(f(x)) \wedge p(g(x)))$$

Goal:

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

Suppose now that we discover the fact that our dog rex is purebred. This gives us an additional axiom. Given this information, we know that every dog in Rex's ancestry tree must be purebred. We can prove this by a simple application of tree induction.

Canine Ancestry Proof

- | | | |
|----|---|---------|
| 1. | $p(\text{rex})$ | Premise |
| 2. | $\forall x.(p(x) \Leftrightarrow p(f(x)) \wedge p(g(x)))$ | Premise |

Here is a proof of our conclusion. We start with the premise that Rex is purebred. We also have our constraint on purebred animals as a premise.

Canine Ancestry Proof

- | | | |
|----|---|---------|
| 1. | $p(\text{rex})$ | Premise |
| 2. | $\forall x.(p(x) \Leftrightarrow p(f(x)) \wedge p(g(x)))$ | Premise |
| 3. | $p(c) \Leftrightarrow p(f(c)) \wedge p(g(c))$ | UE: 2 |
| 4. | $p(c) \Rightarrow p(f(c)) \wedge p(g(c))$ | BE: 3 |

We use Universal Elimination to instantiate the second premise, and then we use Biconditional Elimination on the biconditional in line 3 to produce the implication on line 4.

Canine Ancestry Proof

- | | | |
|----|---|------------|
| 1. | $p(\text{rex})$ | Premise |
| 2. | $\forall x.(p(x) \leftrightarrow p(f(x)) \wedge p(g(x)))$ | Premise |
| 3. | $p(c) \leftrightarrow p(f(c)) \wedge p(g(c))$ | UE: 2 |
| 4. | $p(c) \Rightarrow p(f(c)) \wedge p(g(c))$ | BE: 3 |
| 5. | $\begin{array}{ l} p(c) \end{array}$ | Assumption |

On line 5, we start a subproof with the assumption the x is purebred.

Canine Ancestry Proof

- | | | |
|----|---|------------|
| 1. | $p(\text{rex})$ | Premise |
| 2. | $\forall x.(p(x) \leftrightarrow p(f(x)) \wedge p(g(x)))$ | Premise |
| 3. | $p(c) \leftrightarrow p(f(c)) \wedge p(g(c))$ | UE: 2 |
| 4. | $p(c) \Rightarrow p(f(c)) \wedge p(g(c))$ | BE: 3 |
| 5. | $\begin{array}{ l} p(c) \end{array}$ | Assumption |
| 6. | $\begin{array}{ l} p(f(c)) \wedge p(g(c)) \end{array}$ | IE: 4, 5 |

We use Implication Elimination to derive the conjunction on line 6.

Canine Ancestry Proof

- | | | |
|----|---|------------|
| 1. | $p(\text{rex})$ | Premise |
| 2. | $\forall x.(p(x) \leftrightarrow p(f(x)) \wedge p(g(x)))$ | Premise |
| 3. | $p(c) \leftrightarrow p(f(c)) \wedge p(g(c))$ | UE: 2 |
| 4. | $p(c) \Rightarrow p(f(c)) \wedge p(g(c))$ | BE: 3 |
| 5. | $\begin{array}{ l} p(c) \end{array}$ | Assumption |
| 6. | $\begin{array}{ l} p(f(c)) \wedge p(g(c)) \end{array}$ | IE: 4, 5 |
| 7. | $\begin{array}{ l} p(f(c)) \end{array}$ | AE: 6 |

Then we use And Elimination to pick out the first conjunct.

Canine Ancestry Proof

1.	$p(\text{rex})$	Premise
2.	$\forall x.(p(x) \leftrightarrow p(f(x)) \wedge p(g(x)))$	Premise
3.	$p(c) \leftrightarrow p(f(c)) \wedge p(g(c))$	UE: 2
4.	$p(c) \Rightarrow p(f(c)) \wedge p(g(c))$	BE: 3
5.	$p(c)$	Assumption
6.	$p(f(c)) \wedge p(g(c))$	IE: 4, 5
7.	$p(f(c))$	AE: 6
8.	$p(c) \Rightarrow p(f(c))$	II: 5, 7
9.	$\forall x.(p(x) \Rightarrow p(f(x)))$	UI: 8

We then use Implication Introduction to discharge our assumption, and we Universal Introduction to produce the inductive case for f.

Canine Ancestry Proof

1.	$p(\text{rex})$	Premise
2.	$\forall x.(p(x) \leftrightarrow p(f(x)) \wedge p(g(x)))$	Premise
3.	$p(c) \leftrightarrow p(f(c)) \wedge p(g(c))$	UE: 2
4.	$p(c) \Rightarrow p(f(c)) \wedge p(g(c))$	BE: 3
...		
9.	$\forall x.(p(x) \Rightarrow p(f(x)))$	UI: 8
...		
14.	$\forall x.(p(x) \Rightarrow p(g(x)))$	UI: 13

We then repeat this process to produce an analogous result for g on line 14.

Canine Ancestry Proof

1.	$p(\text{rex})$	Premise
2.	$\forall x.(p(x) \leftrightarrow p(f(x)) \wedge p(g(x)))$	Premise
3.	$p(c) \leftrightarrow p(f(c)) \wedge p(g(c))$	UE: 2
4.	$p(c) \Rightarrow p(f(c)) \wedge p(g(c))$	BE: 3
...		
9.	$\forall x.(p(x) \Rightarrow p(f(x)))$	UI: 8
...		
14.	$\forall x.(p(x) \Rightarrow p(g(x)))$	UI: 13
15.	$\forall x.p(x)$	Ind: 1, 9, 14

Finally, we use the tree induction rule on the sentences on lines 1, 9, and 14 and thereby derive the desired overall conclusion.

Linear Induction

Linear Language:

$$a \rightarrow s(a) \rightarrow s(s(a)) \rightarrow s(s(s(a))) \rightarrow \dots$$

Induction:

$$\frac{\begin{array}{l} \phi[a] \\ \forall \mu. (\phi[\mu] \Rightarrow \phi[s(\mu)]) \end{array}}{\forall v. \phi[v]}$$

Multiple Object Constants

Linear Language:

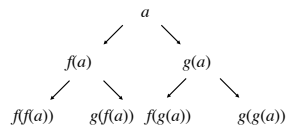
$$\begin{array}{l} a \rightarrow s(a) \rightarrow s(s(a)) \rightarrow s(s(s(a))) \rightarrow \dots \\ b \rightarrow s(b) \rightarrow s(s(b)) \rightarrow s(s(s(b))) \rightarrow \dots \end{array}$$

Induction:

$$\frac{\begin{array}{l} \phi[a] \\ \phi[b] \\ \forall \mu. (\phi[\mu] \Rightarrow \phi[s(\mu)]) \end{array}}{\forall v. \phi[v]}$$

Tree Induction

Tree Language:



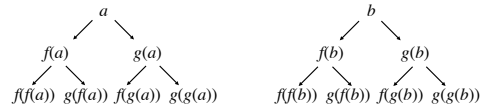
Induction:

$$\frac{\begin{array}{l} \phi[a] \\ \forall \mu. (\phi[\mu] \Rightarrow \phi[f(\mu)]) \\ \forall \mu. (\phi[\mu] \Rightarrow \phi[g(\mu)]) \end{array}}{\forall v. \phi[v]}$$

Consider a language with an object constant a and two unary function constants f and g . Some of the terms in the language are shown here.

Multiple Object and Function Constants

Tree Language:



Induction:

$$\begin{array}{l}
 \phi[a] \\
 \phi[b] \\
 \forall \mu. (\phi[\mu] \Rightarrow \phi[f(\mu)]) \\
 \forall \mu. (\phi[\mu] \Rightarrow \phi[g(\mu)]) \\
 \hline
 \forall v. \phi[v]
 \end{array}$$

Structural Induction

Structural induction is the most general form of induction. In structural induction, we can have multiple object constants, multiple function constants, and, unlike our other forms of induction, we can have function constants with multiple arguments.

n -ary Function Constant

Object constant: a

Binary function constant: f

Sample Terms

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

Consider a language with two object constants a and b and a single binary function constant c . Here we see a list of some of the terms in the language. There is no graphical rendering in this case, as the structure is more complicated than a line or a tree.

Structural Induction

Object constant: a
Binary function constant: f

$$\frac{\begin{array}{l} \phi[a] \\ \forall \lambda. \forall \mu. (\phi[\lambda] \wedge \phi[\mu] \Rightarrow \phi[f(\lambda, \mu)]) \end{array}}{\forall v. \phi[v]}$$

The structural induction rule for this language is shown here. If we know that ϕ holds of our base element and if we know that, if ϕ holds of any two elements, it holds of their combination, then we can conclude ϕ holds of everything.

Structural Induction

Object constants: a, b
Binary function constant: f

$$\frac{\begin{array}{l} \phi[a] \\ \phi[b] \\ \forall \lambda. \forall \mu. (\phi[\lambda] \wedge \phi[\mu] \Rightarrow \phi[f(\lambda, \mu)]) \end{array}}{\forall v. \phi[v]}$$

Structural Induction

Object constant: a
Binary function constants: f, g

$$\frac{\begin{array}{l} \phi[a] \\ \forall \lambda. \forall \mu. (\phi[\lambda] \wedge \phi[\mu] \Rightarrow \phi[f(\lambda, \mu)]) \\ \forall \lambda. \forall \mu. (\phi[\lambda] \wedge \phi[\mu] \Rightarrow \phi[g(\lambda, \mu)]) \end{array}}{\forall v. \phi[v]}$$

Structural Induction

Object constants: a, b

Binary function constants: f, g

$$\begin{array}{l}
 \phi[a] \\
 \phi[b] \\
 \forall \lambda. \forall \mu. (\phi[\lambda] \wedge \phi[\mu] \Rightarrow \phi[f(\lambda, \mu)]) \\
 \forall \lambda. \forall \mu. (\phi[\lambda] \wedge \phi[\mu] \Rightarrow \phi[g(\lambda, \mu)]) \\
 \hline
 \forall v. \phi[v]
 \end{array}$$

Multiple object constants, multiple function constants, multiple conditions.

Objects and Base Cases

Object constants: a, b

Binary function constants: f, g

$$\begin{array}{l}
 \phi[a] \\
 \phi[b] \\
 \forall \lambda. \forall \mu. (\phi[\lambda] \wedge \phi[\mu] \Rightarrow \phi[f(\lambda, \mu)]) \\
 \forall \lambda. \forall \mu. (\phi[\lambda] \wedge \phi[\mu] \Rightarrow \phi[g(\lambda, \mu)]) \\
 \hline
 \forall v. \phi[v]
 \end{array}$$

Multiple object constants, multiple function constants, multiple conditions.

Functions and Inductive Cases

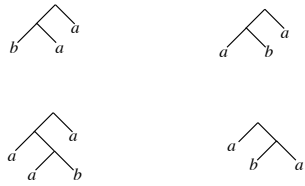
Object constants: a, b

Binary function constants: f, g

$$\begin{array}{l}
 \phi[a] \\
 \phi[b] \\
 \forall \lambda. \forall \mu. (\phi[\lambda] \wedge \phi[\mu] \Rightarrow \phi[f(\lambda, \mu)]) \\
 \forall \lambda. \forall \mu. (\phi[\lambda] \wedge \phi[\mu] \Rightarrow \phi[g(\lambda, \mu)]) \\
 \hline
 \forall v. \phi[v]
 \end{array}$$

Multiple object constants, multiple function constants, multiple conditions.

Trees

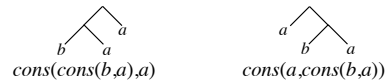


Trees have many properties - e.g. symmetry, uniformity. Also many relationships - e.g. congruence.

Tree Vocabulary

Object constants: a, b

Unary function constants: $cons$



Unary relation constants: *symmetric, uniform*

Binary relation constant: *subtree, congruent*

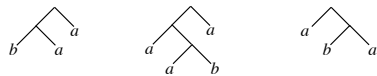
Congruence

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

$$\begin{aligned} & \text{congruent}(a, a) \\ & \text{congruent}(a, b) \\ & \text{congruent}(b, a) \\ & \text{congruent}(b, b) \end{aligned}$$

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:

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

Properties

Congruence is an equivalence relation.

Reflexivity

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

Symmetry

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

Transitivity

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

Problem

Given:

$$\begin{aligned} & \text{congruent}(a, a) \\ & \text{congruent}(a, b) \\ & \text{congruent}(b, a) \\ & \text{congruent}(b, b) \end{aligned}$$
$$\forall u. \forall v. \forall x. \forall y. (\text{congruent}(\text{cons}(u, v), \text{cons}(x, y)) \Leftrightarrow \text{congruent}(u, x) \wedge \text{congruent}(v, y))$$

Prove:

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

Problem Restated using $\equiv$

Given:

$$\begin{aligned}a &\equiv a \\ a &\equiv b \\ b &\equiv a \\ b &\equiv b\end{aligned}$$

$$\forall u. \forall v. \forall x. \forall y. (cons(u, v) \equiv cons(x, y) \Leftrightarrow u \equiv x \wedge v \equiv y)$$

Prove:

$$\forall x. x \equiv x$$

Induction Strategy

Our goal is to prove that congruence is reflexive.

$$\forall x. x \equiv x$$

Base Cases:

$$\begin{aligned}a &\equiv a \\ b &\equiv b\end{aligned}$$

Inductive Case:

$$\forall x. \forall y. (x \equiv x \wedge y \equiv y \Rightarrow cons(x, y) \equiv cons(x, y))$$

Proof of Reflexivity

- | | |
|--|---------|
| 1. $a \equiv a$ | Premise |
| 2. $a \equiv b$ | Premise |
| 3. $b \equiv a$ | Premise |
| 4. $b \equiv b$ | Premise |
| 5. $\forall u. \forall v. \forall x. \forall y. (cons(u, v) \equiv cons(x, y) \Leftrightarrow u \equiv x \wedge v \equiv y)$ | Premise |

Proof of Reflexivity

- | | |
|--|-----------|
| 1. $a \equiv a$ | Premise |
| 2. $a \equiv b$ | Premise |
| 3. $b \equiv a$ | Premise |
| 4. $b \equiv b$ | Premise |
| 5. $\forall u. \forall v. \forall x. \forall y. (cons(u, v) \equiv cons(x, y) \Leftrightarrow u \equiv x \wedge v \equiv y)$ | Premise |
| 6. $cons(c, d) \equiv cons(c, d) \Leftrightarrow c \equiv c \wedge d \equiv d$ | 4 x UE: 5 |
| 7. $c \equiv c \wedge d \equiv d \Rightarrow cons(c, d) \equiv cons(c, d)$ | BE: 6 |

Proof of Reflexivity

- | | |
|--|-----------|
| 1. $a \equiv a$ | Premise |
| 2. $a \equiv b$ | Premise |
| 3. $b \equiv a$ | Premise |
| 4. $b \equiv b$ | Premise |
| 5. $\forall u. \forall v. \forall x. \forall y. (cons(u, v) \equiv cons(x, y) \Leftrightarrow u \equiv x \wedge v \equiv y)$ | Premise |
| 6. $cons(c, d) \equiv cons(c, d) \Leftrightarrow c \equiv c \wedge d \equiv d$ | 4 x UE: 5 |
| 7. $c \equiv c \wedge d \equiv d \Rightarrow cons(c, d) \equiv cons(c, d)$ | BE: 6 |
| 8. $\forall x. \forall y. (x \equiv x \wedge y \equiv y \Rightarrow cons(x, y) \equiv cons(x, y))$ | 2 x UI: 7 |

Proof of Reflexivity

- | | |
|--|--------------|
| 1. $a \equiv a$ | Premise |
| 2. $a \equiv b$ | Premise |
| 3. $b \equiv a$ | Premise |
| 4. $b \equiv b$ | Premise |
| 5. $\forall u. \forall v. \forall x. \forall y. (cons(u, v) \equiv cons(x, y) \Leftrightarrow u \equiv x \wedge v \equiv y)$ | Premise |
| 6. $cons(c, d) \equiv cons(c, d) \Leftrightarrow c \equiv c \wedge d \equiv d$ | 4 x UE: 5 |
| 7. $c \equiv c \wedge d \equiv d \Rightarrow cons(c, d) \equiv cons(c, d)$ | BE: 6 |
| 8. $\forall x. \forall y. (x \equiv x \wedge y \equiv y \Rightarrow cons(x, y) \equiv cons(x, y))$ | 2 x UI: 7 |
| 9. $\forall x. (x \equiv x)$ | Ind: 1, 4, 8 |

