

# PARTIVc: INFERENCE RULES FOR FIRST-ORDER LOGIC

Version 2.11: 2004/05/14 14:25:13.514 GMT-5

John Greiner  
Ian Barland

This work is produced by The Connexions Project and licensed under the  
Creative Commons Attribution License \*

## Abstract

How to reason with first-order formulas.

## partIVc: Inference rules for first-order logic

### 1 Inference with quantifiers

Proving first-order sentences with inference rules is not substantially different than for propositional ones. We simply need to introduce new inference rules for dealing with the new syntactic forms – relations and quantifiers.

To understand relations, we turn to their interpretation<sup>1</sup>. An interpretation provides axioms that we can use, just like those in propositional logic. For example, we have the axioms for WaterWorld as modeled in first-order logic<sup>2</sup>.

What is the most natural way to prove an existential sentence, like there exists a prime number greater than 5? That's easy, you just mention such a number, like 11, and show that it is indeed prime and greater than 5. In other words, to prove  $\exists x.\phi$ , we'll choose some **witness**, and show that  $\phi$  holds, if we use that witness in the place of  $x$ . We use the notation  $\phi[x/t]$  to mean such a **substitution** of a term  $t$  for every occurrence of variable  $x$  within  $\phi$ .

ASIDE: Technically, we have to be careful. If there are multiple quantifiers binding  $x$ , we only want to rename the appropriate ones. Also, we don't want any free variable occurrences in  $t$  to be **captured** by a bound variable in  $\phi$ . For simplicity, we'll continue to assume that the bound variables of  $\phi$  are  $\alpha$ -renamed, so that they are distinctly named from each other and from the variables in  $t$ .

How do you find a witness? That's the difficult part. You, the person creating the proof, must grab a suitable example out of thin air, based on your knowledge of what you want to prove about it. In our previous example, we used our knowledge about prime numbers and about the greater-than relation to pick a witness that would work. In essence, we figured

---

\*<http://creativecommons.org/licenses/by/1.0>

<sup>1</sup><http://cnx.rice.edu/content/m10726/latest/>

<sup>2</sup><http://cnx.rice.edu/content/new0/latest/>

out what facts needed to be true about the witness for the formula to hold, and used that to guide our choice of witness. But, this can easily be more difficult, as when proving that there exists a prime greater than 6971 of the form  $4x - 1$ .

ASIDE: Another is 796751.

Another approach is trial-and-error. Pick a value as the witness, and attempt to continue with the proof. If you succeed, you're done. If not, pick another candidate.

Can we extend that idea to proving a universal sentence? One witness is certainly not enough. We'd need to work with *lots* of witnesses, in fact, every single member of our domain. That's not very practical, especially with infinitely large domains. We need to show that no matter what domain element you choose, the formula holds.

If we choose an **arbitrary** member of the domain, and show that the sentence holds for it, that is sufficient. But, what do we mean by "arbitrary"? In short, it means that we have no control over what element is picked, or equivalently, that the proof must hold regardless of what element is picked. More precisely, a variable is **arbitrary unless**:

- A variable is not arbitrary if it is free in a premise.
- A variable is not arbitrary if it is at any time involved in a  $\exists$ Elim step – either as the introduced witness  $c$ , or anywhere else in the formula.

The usual way to introduce arbitrary variables is during  $\forall$ Elim (w/o later using it in  $\exists$ Elim). The formal inference rule for introduction of universal quantification will use these cases as restrictions.

### 1.1 Formal inference rules and proofs

As with equivalences, we will use our original propositional inference rules<sup>3</sup> (.ps)<sup>4</sup> and add new ones for reasoning about quantified formulas<sup>5</sup> (.pdf)<sup>6</sup> (.ps)<sup>7</sup>.

Recall the syllogisms from a previous lecture. The general form of a syllogism is

1.  $\forall x. (P(x) \rightarrow Q(x))$  [major premise]
2.  $P(c)$  [minor premise]
3.  $Q(c)$  [conclusion]

In our system, we don't have syllogism as a separate rule of inference, but it's easy to see how to translate any syllogism into our system: (for specific relations  $P$  and  $Q$ , and a specific constant  $c$ ).

1	$\forall x. (P(x) \rightarrow Q(x))$	premise
2	$P(c)$	premise
3	$(P(c) \rightarrow Q(c))$	$\forall$ Elim, by line 1, with $x=c$
4	$Q(c)$	$\rightarrow$ Elim, by lines 2,3, with $\phi=Pc$ , and $\psi=Qc$

Eliminating a quantifier via  $\forall$ Elim and  $\exists$ Elim is often merely an intermediate step, where the quantifier will be reintroduced later. This moves the quantification from being explicit

<sup>3</sup><http://cnx.rice.edu/content/m10529/latest/>

<sup>4</sup><http://www.teachLogic.org/Base/Printables/inference-rules.ps>

<sup>5</sup><http://cnx.rice.edu/content/m11046/latest/>

<sup>6</sup><http://www.teachLogic.org/Base/Printables/first-order-inference-rules.pdf>

<sup>7</sup><http://www.teachLogic.org/Base/Printables/first-order-inference-rules.ps>

to implicit, so that we can use other inference rules on the body of the formula. When this is done, it is very important to pay attention to the restrictions on  $\forall$ Intro, so that we don't accidentally "prove" anything too strong.

### Example 1:

$\exists x, ., \forall y, ., \phi \vdash \forall y. \exists x. \phi$ , for the particular case of  $\phi = R(x, y)$  (other cases all similar).

1	$\exists x. \forall y. R(x, y)$	premise
2	$\forall y. R(p, y)$	$\exists$ Elim, by line 1
3	$R(p, q)$	$\forall$ Elim, by line 2
4	$\exists x. R(x, q)$	$\exists$ Intro, by line 3
5	$\forall y. \exists x. R(x, y)$	$\forall$ Intro, by line 4

Note that the other direction doesn't hold.

Also note, we cannot instead conclude in line 4 that  $\forall x. R(x, q)$  by  $\forall$ Intro, since variable  $p$  was introduced by  $\exists$ Elim in line 2.

The  $\forall$ Intro principle is actually very familiar. For instance, after having shown  $\neg(a \wedge b) \vdash (\neg a \vee \neg b)$ , we then claimed this was really true for *arbitrary* propositions instead of just  $a, b$ . (We actually went a bit further, generalizing individual propositions to entire (arbitrary) WFFs  $\phi, \psi$ . This could only be done because in any particular interpretation, a formula  $\phi$  will either be true or false, so replacing it by a proposition still preserves the important part of the proof-of-equivalence.)

The  $\forall$ Intro is also used in many informal proofs. Consider: "If a number  $n$  is prime, then ...". This translates to " $(\text{prime}(n) \rightarrow \dots)$ ", where  $n$  is arbitrary. We are entirely used to thinking of this as " $\forall n. (\text{prime}(n) \rightarrow \dots)$ " even though " $n$ " was introduced as if it were a particular number.

## 2 Proofs and programming

We previously saw<sup>8</sup> that the inference rules of propositional logic are closely related to the process of type checking. The same holds here. For example, in many programming languages, we can write a sorting function that works on *any* type of data. It takes two arguments, a comparison function for the type and a collection (array, list, ...) of data of that type. The type of the sorting function can then be described as "for all types  $T$ , given a function of type  $(T \text{ and } T) \rightarrow T$ , and data of type (collection  $T$ ), it returns data of type (collection  $T$ )". This **polymorphic** type uses universal quantification.

ASIDE: *poly*- from Gk. "much" or "many". *morph* - "form", "beauty", or "outward appearance"

Note that the details about substitutions and capture noted here arise in any kind of program that manipulates expressions with bound variables. That includes not only automated theorem provers, but compilers. To avoid such issues, many systems essentially rename all variables by using pointers or some similar system of each variable referring to the corresponding quantifier.

<sup>8</sup><http://cnx.rice.edu/content/m10718/latest/#programming>