Predicate Logic

Robert Keller

5 March 2007

Predicate logic is the most versatile family of logic. It builds atop proposition logic, in that there are certain sub-expressions identifiable as proposition. However, it is much richer, in that it enables discussing individuals in a domain, whereas proposition logic is only able to deal with propositions are indecomposable.

It is useful to extend the grammar of proposition logic to first include *terms*, which represent individuals in a *domain* or *universe of discourse*. Terms are constants, variables, or the result of applying a function symbol to terms, figuratively:

$T \rightarrow C \mid V \mid F(T, T, \ldots, T)$	*terms*
$C \rightarrow c \mid C'$	*constant symbols*
$V \rightarrow x \mid V'$	*variable symbols*
$F \rightarrow f \mid F'$	*function symbols*

Here we are using, the prime device to permit an unlimited number of the various symbols, e.g. c, c', c'', … for constant symbols. We won't hesitate to use a, b, c for constant symbols, in lieu of the prime symbols for readability. Similarly, we use f, g, h, … for function symbols, x, y, z, … for variables, etc.

Examples of Terms:

c	*constant*
x	*variable*
f(c, x)	*function application*
g(f(c, x), d)	*function application*

Atomic formulas are ones that represent true or false, given specific individuals. They play the role of propositions in the proposition calculus.

$A \rightarrow P(T, T, \ldots, T) \mid \bot$	*atomic formulas*
$P \rightarrow p \mid P'$	*predicate symbols*

Examples of Atomic Formulas:

P(c)	*predicate value*
P(x)	*predicate value*
P(g(f(c, x), d))	*predicate value*

Formulas can be atomic, formulas connected by the propositional connectives, or quantified formulas:

$$F \rightarrow A \mid (F \wedge F) \mid (F \text{ `}\rightarrow\text{' } F) \mid \forall V\ F \qquad \textit{formulas}$$

$\forall$ is called the *universal quantifier* ("for all"). It is always followed by a variable and a sub-formula.

The *existential quantifier* $\exists$ ("for some") abbreviates, much as $\varphi \vee \psi$ abbreviates $\neg(\neg\varphi \wedge \neg\psi)$:

$\exists x\ \varphi$ abbreviates $\neg \forall x\ (\neg\varphi)$

where x is a variable and φ a formula.

Semantics

Recall that semantics is the endeavor of assigning *meanings* to formulas. Meanings of formulas are expressed in terms of *interpretations*, which give meaning to individual constant, function, and predicate symbols. In order to define "interpretation", we must first define "structure".

A *structure* is a collection of items:

- A domain D (*always* non-empty)
- Some functions on the domain (arity to be specified)
- Some predicates on the domain (arity also to be specified)

An example of structure might be the *natural numbers*

- Domain: {0, 1, 2, 3, …}
- Functions ' (successor, add 1), + (addition)
- Predicates = (equality), <

All structures are assumed to have an equality predicate.

An *interpretation* for a structure is a mapping from symbols into elements of the structure:

- Each constant symbol maps to a specific domain element, e.g. **0** the constant symbol maps to 0 the number.
- Each function symbol maps to a specific function on the domain:
  - ' (prime) maps to the unary function that adds 1 to its argument
  - + maps to the binary function that adds two numbers.
- Each predicate symbol maps to a specific predicate (or relation) on the domain:
  - = maps to the equality predicate

- < maps to the less-than predicate

Recall that an n-ary predicate is just a set of n-tuples. An n-ary function is a set of n+1-tuples, with certain additional properties.

A term is called *closed* if it contains no variables. An interpretation I of a closed term t *induces* a value I(t) that is a domain element. Recursively:

- A constant induces the corresponding value as defined in the interpretation.
- A function symbol applied to terms induces the value determined by applying the corresponding function to the induced values of the arguments.

An atomic formula is closed if it contains no variables. An interpretation I of a closed atomic formula induces a value that is 1 (true) or 0 (false). This value is determined by applying the corresponding predicate to the induced values of the arguments.
Example: With the natural numbers interpretation, here are some terms, and their induced values.

Closed Term t	Induced Value I(t)
0	0
0’’’	3
0’’’ + 0’’’’’	8
(0’’’ + 0’’’’’)’’	10

Here are some closed atomic formulae over the same interpretation:

Closed Atomic Formula A	Induced Value I(A)
0’ = 0	0 (false)
0’’’ < 0’’’’	1 (true)
0 + 0’ = 0’ + 0	1

The induced value of a closed atomic formula plays a role similar to *valuation* in proposition logic. Truth values are induced for closed formulae connected with propositional connectives in the same way as for proposition logic, e.g.

Closed Formula F	Induced Value I(F)
0’ = 0 ∧ 0 = 0	0 (false)
0’’’ < 0’’’’ ∧ 0’ < 0’’	1 (true)

An interpretation *satisfies* a closed formula iff its induced value is 1.

For formulas that have quantifiers, the induced truth values take into consideration all values in the domain of the interpretation. One way to handle this is to introduce a *constant symbol for each domain value* in D. For example, for the natural numbers, we’d have an infinite set of new constant symbols. The induced truth value of a closed formula having only variable x is

$$I(\forall x\ F) = \wedge\{I(F[d/x]) \mid d \in D\}$$

where F[d/x], read "F with d for x" means the *substitution* for x of the constant *symbol* corresponding to domain element d. The conjunction over a set of values means that all values in the set must be 1 in order for the result to be 1. Otherwise the result is 0.

The Purposes of Variables

Variables in predicate formulas serve several purposes:

- A variable establishes the parts of "variability" in the presence of quantification.
- A variable can be used to identify as a single entity individuals that occur in different parts of a formula.

As an example of the first case, in $\forall x\ P(x) \vee \exists y\ \neg Q(y)$ there are two parts of variability, regarding the predicate symbols P and Q, and they are distinguished by their own variables. For the second case, consider $\forall x\ (P(x) \vee \neg Q(x))$. Here both predicate symbols refer to the same individual. Similarly, for a two-place predicate, say R, R(x, x) is quite different from R(x, y). The first case says that the individual represented by x is related to itself, whereas the second says that x is related to y. Here x and y could represent the same or different variables. $\forall x \forall y\ R(x, y)$ says that any two individuals are related, *including* each individual being related to itself.

In a formula with no quantifiers, all variables are *free*. When a quantifier is added, that variable becomes *bound* (the opposite of free). Binding relates *instances* of a variable to the specific quantifier. For example, in both formulas $\forall x\ P(x)$ and $\exists x\ \neg Q(x)$ variable x is bound. In $\forall x\ P(x) \vee \exists x\ \neg Q(x)$, the instances of x do *not* refer to the same individual, because they are bound separately in different sub-formulas. Note that a binding can be added even if a variable is already bound, for example: $\forall x\ \exists x\ P(x)$. In this case, it is really the inner binding that matters. The outer instance of x is not identified with any other x in the formula. It might as well not be there. Still, this formula is legitimate.

One way to keep bindings and instances straight is to draw lines under the formula showing which refer to the same individual, as in

$$\begin{array}{l} \quad\quad\quad\quad\ \ \_\_\_\_\_\quad\quad\quad\ \ \_\_\_\_\_ \\ \quad\quad\quad\quad\ \ |\quad\quad|\quad\quad\quad|\quad\quad\ \ | \\ \forall x\ (P(x) \wedge \exists y\ R(x, y)) \vee \exists x\ \forall y\ \neg S(x, y) \\ \ \ |\_\_\_|\_\_\_\_\_|\quad\quad\quad\ \ |\_\_\_\_\_| \end{array}$$

Here there are two different uses of each of x and y.

Induced values are first defined for formulas with no free variables. Then the definition for a formula with free variables is *as if* the formula had *implicit* $\forall$ quantifiers over each free variable, the so-called *universal closure* of the formula.

When there are several quantified variables, the evaluation of the induced value proceeds recursively, first fixing the outermost bound variable, then varying the next level variable, etc.

When the existential quantifier is present, the induced value is effectively a disjunction:

$$I(\exists x\ F) = \vee\{I(F[d/x]) \mid d \in D\}$$

Examples

Consider a structure with domain {0, 1, 2}, a binary function + specified by x + y = (x + y) mod 3, an equality predicate =, and the predicate **even** which is true for even numbers {0, 2}.

Determine the induced truth values of the following formulae:

- $\forall x\ (even(x) \vee even(x+1)$
- $\forall x\ (even(x) \rightarrow \neg even(x+1)$
- $\forall x\ \exists y\ (x + y) = 0$
- $\exists x\ \forall y\ (x + y) = y$

Note that the same formula could apply to a different interpretation, e.g. the natural numbers, but then the induced values might be different.

Validity

When a formula φ induces the value 1 with respect to an interpretation I, it is said to be *valid for* that interpretation.

$$I \models \varphi$$

A formula is simply *valid* if it is valid for *all* interpretations that make sense, in that the interpretation provides functions, predicates, and constants that agree with the arities in the formula.

$$\models \varphi$$

A formula can be shown *not* valid simply by giving a *counterexample*, i.e. an interpretation in which it is false.

A formula is *satisfiable* iff there is an interpretation for which is valid. Otherwise it is *unsatisfiable*. Thus, as with proposition logic, a formula is valid iff its negation is unsatisfiable.

Example: Consider two similar formulas:

a. $\exists x\,(d(x) \rightarrow \forall y\, d(y))$

b. $(\exists x\, d(x)) \rightarrow (\forall y\, d(y))$

Show that a. is valid but b. is not.

Refutation Tree Method

This method extends the corresponding method for proposition logic to predicate logic, insofar as that is possible. Be forewarned that *there is no algorithm that will test for the satisfiability of an arbitrary predicate logic formula*. Although this method will never give a wrong answer, there are instances where it will fail to terminate.

All of the rules that we used for proposition logic are still in effect. But there are some additional rules for quantifiers.

$\neg\forall$ rule	Replace a formula $\neg\forall x\, \varphi$ with $\exists x\, \neg\varphi$.	$\neg(\forall \nu)\ \varphi$ ✓ $(\exists \nu)\ \neg\varphi$
$\neg\exists$ rule	Replace a formula $\neg\exists x\, \varphi$ with $\forall x\, \neg\varphi$.	$\neg(\exists \nu)\ \varphi$ ✓ $(\forall \nu)\ \neg\varphi$
$\exists$ rule	Replace a formula $\exists x\, \varphi$ with $\varphi[\mathbf{c}/x]$, the result of substituting **c** for each *free* occurrence of x, where **c** is a *new* constant symbol, not otherwise used. The formula being replaced is checked and not used again.	$(\exists x)\ \varphi$ ✓ $\varphi[c/x]$
$\forall$ rule	Add on each path below, a formula $\varphi[t/x]$ where t is *any* term. The formula is *not checked*, and *can* be used again.	$(\forall \nu)\ \varphi$ $\varphi[t/x]$

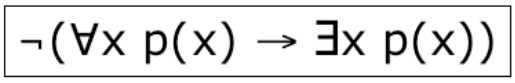
Note that the $\forall$ rule is not algorithmic, in that it there is substantial freedom in how it is used. Use of this rule is part of the "art" of showing a formula unsatisfiable using the refutation tree method.

As in the propositional version, a path from the root is *closed* if it contains a formula and its negation. Closed paths are marked by placing $\perp$ at the end of the path. Once a path becomes closed, it is not further extended by replacements or splits.

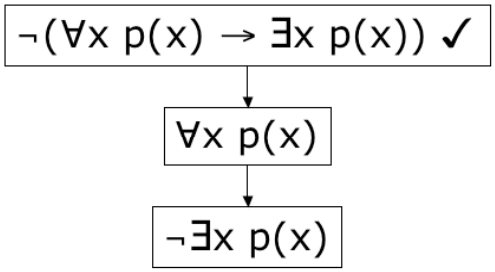
Main Result: If the formula at the root is unsatisfiable then a state of expansion can be reached wherein all paths are closed.

Example: Determine whether or not $(\forall x\ p(x)) \rightarrow (\exists x\ q(x))$ is valid using the refutation tree method.

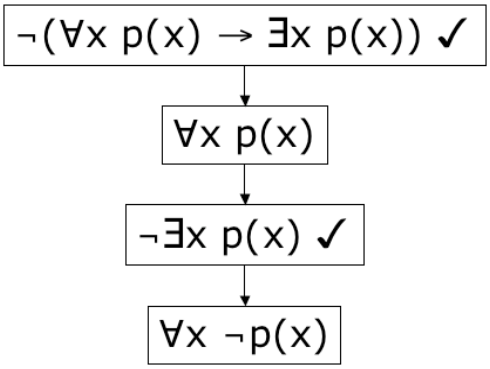
We start the tree with the *negation* of the formula of interest:



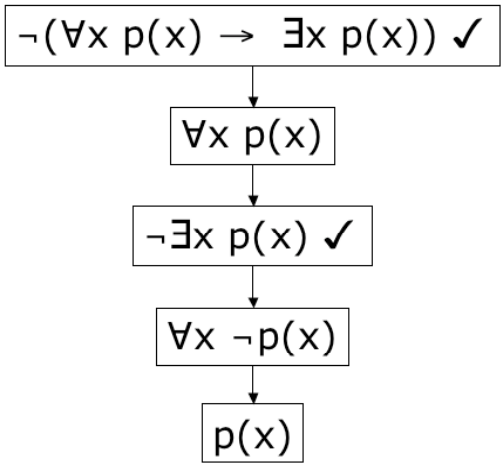
Use the ¬→ rule from proposition logic, to stack:



Now use the ¬∃ rule:

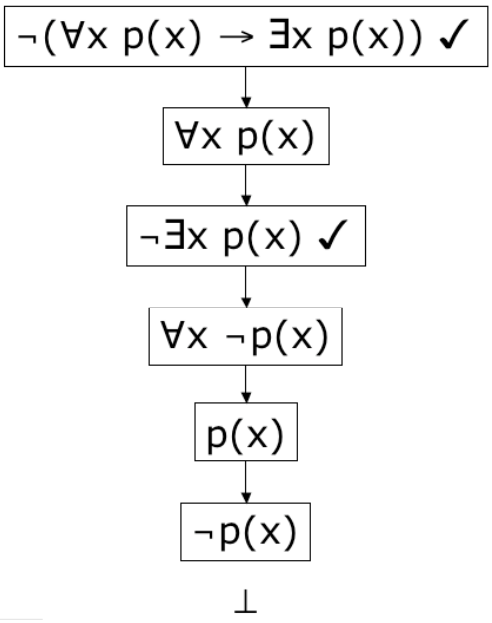


Now use the ∀ rule, with x as the term substituted for x:



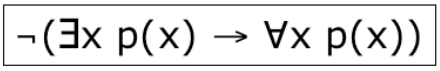
Note that the substituted x is not particularly linked to the x in the formula from which it was derived. It was just a convenient choice. Note also that the parent formula was not checked.

Now use the ∀ rule again, this time on ∀x ¬p(x). Here we substitute the *same* x as in the previous use of the rule, because we want to get a formula and its negation:

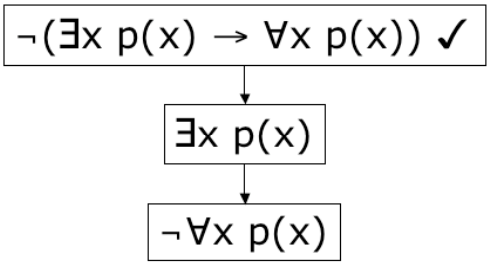


The one and only path closes, so the original formula (∀x p(x)) → (∃x p(x)) is valid, because its negation is unsatisfiable.

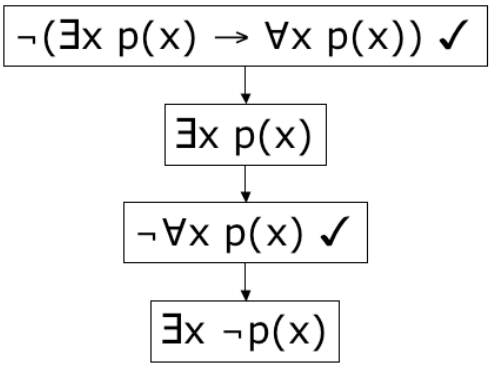
Example: Determine whether or not (∃x p(x)) → (∀x p(x)) is valid using the refutation tree method. The root of the tree is the negated formula:



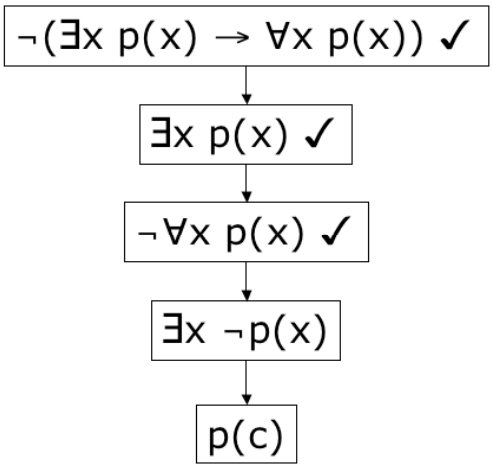
Use the propositional ¬→ rule:



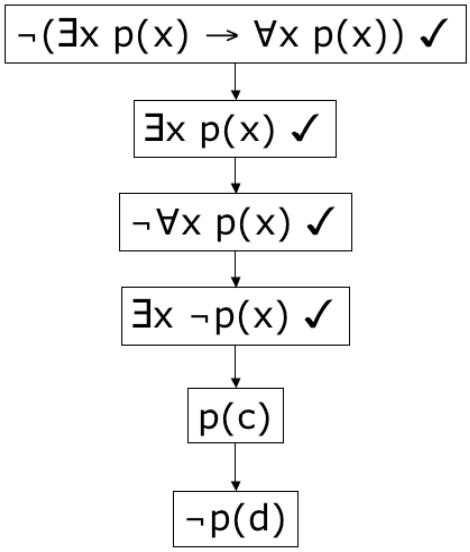
Use the ¬∀ rule:



Use the ∃, introducing constant **c**:



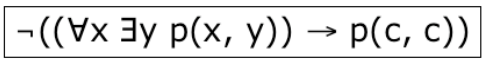
Use the ∃, introducing constant **d**:



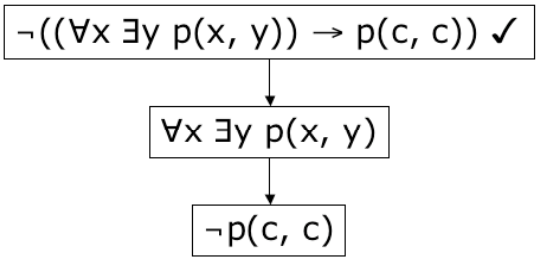
Notice that the one and only path cannot be extended further. Yet it is not closed, so the negated formula is satisfiable, and thus the original formula is not valid.

A counterexample interpretation I can be found from the tree. The domain of I is {c, d}, and there is a single predicate p for which p(c) is true, but p(d) is not.

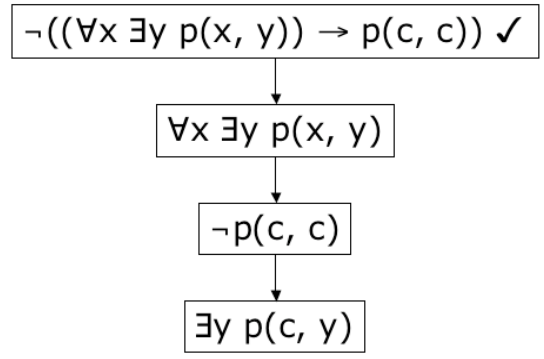
Example: Attempt to determine whether or not (∀x∃y p(x, y)) → p(c, c) is valid using the refutation tree method. The root of the tree is the negated formula:



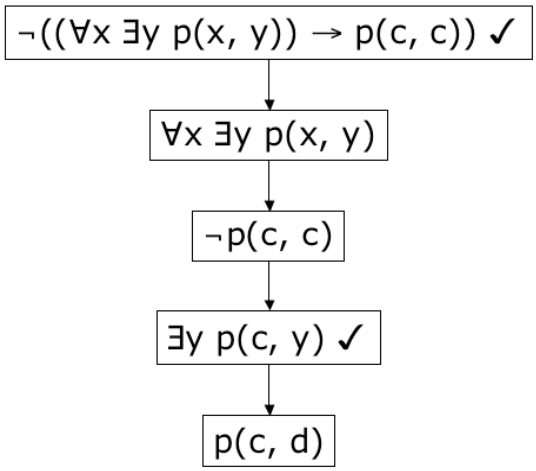
Apply the ¬ rule:



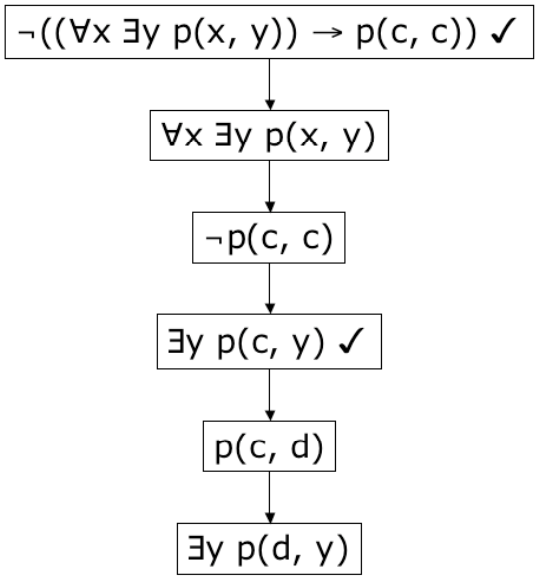
Use the ∀ rule, substituting **c** for x (the ∀ formula does not get checked off):



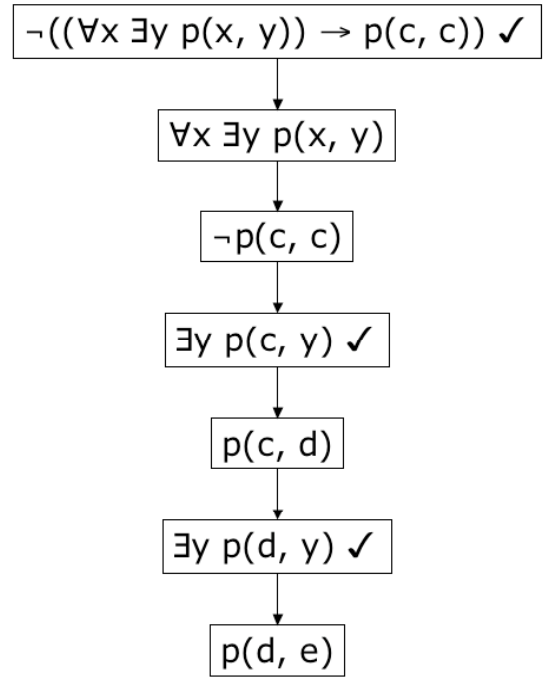
Use the ∃ rule, introducing new constant **d** for y (the ∃ formula *does* get checked off):



Now we can use the ∀ rule again, substituting **d** for x:



Use the ∃ rule, introducing new constant **e** for y:



By now you can get the idea that this can go on forever, neither closing a path nor running out of new formulas to generate.

In this example, the original formula was not valid. There is an implied counterexample: one in which the domain is infinite {c, d, e, … } with a predicate p such that p(c, d), p(d, e), p(e, f), … but not p(c, c). However, this counterexample cannot be derived explicitly from a finite open path.

We have now shown by example the three possible outcomes of the refutation tree method for predicate logic:

- The tree construction terminates with all paths closed, meaning the root is unsatisfiable.
- The tree construction terminates with an open path, meaning the root is satisfiable.
- The tree construction does not terminate. The root is satisfiable. However, there is no algorithm for detecting non-termination in general.

If the root is derived as the negation of a formula, that formula is valid iff the root is not satisfiable. In some sense, this means that validity is “easier” to check than satisfiability, at least by this method. If the original formula is valid, the tree construction will necessarily terminate. This is not the case if its negation is satisfiable.

Proof Methods

We now introduce natural deduction proof rules for predicate logic. The main rules to be added involve the quantifiers. As before, we want an elimination rule and an introduction rule for each quantifier. Unlike the propositional case, however, there are some syntactic restrictions on how the rules are used.

The first rule for the ∀ quantifier is:

∀-Elimination:

$$\frac{\forall x\ \varphi}{\varphi[t/x]}\ \forall E$$

provided that t is any term that is *free for x in* φ.

The phrase "t is free for x in φ" means that no variable in t (including t itself, if it is a variable) becomes bound as a result of the substitution in φ.

Note that the restriction on t does not rule out t *being* x.

Also notice that φ[t/x] *means* the replacement of all *free* occurrences of x with t, not the replacement of all occurrences.

Examples:

$$\frac{\forall x\ p(x)}{p(x)}\ \forall E$$ Permitted (x for x in p(x))

$$\frac{\forall x \exists y\ p(x, y)}{\exists y\ p(y, y)}\ \forall E$$ Not permitted (y for x in ∃y p(y, x)); y is not free for x in ∃y p(x, y)

$$\frac{\forall x\ p(x)}{p(f(x))}\ \forall E$$ Permitted (f(x) for x in p(x))

$$\frac{\forall x \exists y\ p(x, y)}{\exists y\ p(f(y), y)}\ \forall E$$ Not permitted (f(y) for x in ∃y p(x, y)); y is not free for x in ∃y p(x, y)

$$\frac{\forall x\ p(x, y)}{p(y, y)}\ \forall E$$ Permitted (y for x in p(x, y))

$$\frac{\forall x\ p(x, y)}{p(f(y), y)}\ \forall E$$ Permitted (f(y) for x in p(x, y))

The idea of ∀-Elimination is that if we have derived a statement about *every* x in the domain, we should be able to substitute for x any specific term, since such a term just

represents a domain value. The only precaution is that we cannot make substitutions that contain variables that would become bound as a result.

The first not-permitted example above shows why. Although for every x there may be a y such that p(x, y), substituting y would require it to mean the same thing as another variable that already has a meaning. (Consider that p(x, y) could mean "x is the father of y".)

For reference, here is van Dalen's recursive definition of "t is free for x in φ" (paraphrased):

- φ is atomic [i.e. has just a single predicate and no quantifiers], *or*
- φ is one of {ψ ∧ ξ, ψ → ξ, ¬ψ, ψ ∨ ξ}, where t is free for x in both ψ and ξ, *or*
- φ is one of {∀yψ, ∃yψ}, x is not y, y is not in t and t is free for x in ψ.

He then goes on to prove:

"t is free for x in φ"
iff
the variables of t are not bound by any quantifier in φ[t/x].

The latter seems like it would have been a fine definition to me, but the definition given is computational.

The second rule for the ∀ quantifier is:

∀- Introduction:

$$\frac{\varphi}{\forall x \varphi} \quad \forall I$$

provided that x is not free in any hypothesis on which φ depends.

The proviso here means that x truly has an *unrestricted* value. One way to get such an x is to introduce it as an unrestricted assumption. In mathematics, we say something like:

"Let x be an arbitrary value. … [*derivation of* φ] .. . Since x was arbitrary, ∀xφ."

Examples:

$$\frac{p(x)}{\forall x\, p(x)} \quad \forall I$$

where x is not free in a hypothesis on which p(x) depends

$$\frac{\exists y\, p(x, y)}{\forall x \exists y\, p(x, y)} \quad \forall I$$

where x is not free in a hypothesis on which p(x, y) depends

Quantifier Proof Examples

Example: ∀x p(x) |– ∀y p(y)

Tabular form:

Step	Formula	Justification
1	∀x p(x)	Premise
2	p(y)	1, ∀-Elimination
3	∀y p(y)	2, ∀-Introduction

Tree form:

$$\dfrac{\dfrac{\forall x\, p(x)}{p(y)}\,\forall E}{\forall y\, p(y)}\,\forall I$$

Note that for the ∀I, the proviso is satisfied: y is not free in the hypothesis on which p(y) depends. For contrast, we present an incorrect derivation:

Bad Example: ∀x p(x, y) |– ∀y p(y, y)

Tabular form:

Step	Formula	Justification
1	∀x p(x, y)	Premise
2	p(y, y)	1, ∀-Elimination
3	∀y p(y, y)	2, ∀-Introduction

Above, step 2 is allowable, but step 3 is not, as y is free in hypothesis 1.

Example: ∀x∀y p(x, y) |– ∀y p(y, y)

Tabular form:

Step	Formula	Justification
1	∀x ∀y p(x, y)	Premise
2	∀y p(x, z)	1, ∀-Elimination
3	p(z, z)	2, ∀-Elimination
4	∀y p(y)	2, ∀-Introduction

Note that we substituted variable z for x in step 2. Substituting y would not be allowed, as y is not free for x in ∀y p(x, y). Also, z is not free in the hypothesis on which p(z, z) depends: ∀x ∀y p(x, y).

In many of these examples, having an intuitive understanding of the meaning of the formulas can go a long way in helping decide whether a step violates the constraints.

A standard way of ensuring that the proviso for ∀I is followed is to do a sub-proof (e.g. open a box) in which a totally new variable is introduced at the start. Then when a desired conclusion involving that variable free is reached, the box is closed and the variable is ∀ quantified outside.

Here is how the first proof above might look using this scheme:

Step	**Formula**	**Justification**
1	∀x p(x)	Premise
2	x_0	New variable
3	$p(x_0)$	1, ∀-Elimination
4	∀y p(y)	2-3, ∀-Introduction

Below is another example:

Derive $(\forall x)(p(x) \rightarrow q(x)),\ (\forall x)\ p(x) \vdash (\forall x)\ q(x)$:

1.	$(\forall x)(p(x) \rightarrow q(x))$	Premise
2.	$(\forall x)\ p(x)$	Premise
3.	x_0	
4.	$p(x_0) \rightarrow q(x_0)$	∀E 1
5.	$p(x_0)$	∀E 2
6.	$q(x_0)$	→E 4, 5
7.	$(\forall x)\ q(x)$	∀I 3-6

Here is the English-language equivalent of the above proof:

> "Assume $(\forall x)(p(x) \rightarrow q(x))$ and $(\forall x)\ p(x)$.
>
> Let x_0 be an arbitrary element.
>
> From the the first assumption $p(x_0) \rightarrow q(x_0)$, and from the second $p(x_0)$, hence also $q(x_0)$ by *modus ponens*.
>
> Since x_0 was chosen arbitrarily, $q(x_0)$ gives us $(\forall x)\ q(x)$."

Here is a slightly-tricky example:

Derive $(\forall x)\ (\forall y)\ p(x, y) \vdash (\forall y)\ (\forall x)\ p(x, y)$

(The temptation is to use x_0 in the outer box, but that fails.)

1.	$(\forall x)\ (\forall y)\ p(x, y)$	Premise
2.	y_0	
3.	x_0	
4.	$(\forall y)\ p(x_0, y)$	∀E 1
5.	$p(x_0, y_0)$	∀E 4
6.	$(\forall x)\ p(x, y_0)$	∀I 3-5
7.	$(\forall y)\ (\forall x)\ p(x, y)$	∀I 2-6

Now we give rules for the ∃ quantifier.

The **∃-Introduction rule** is:

$$\frac{\varphi[t/x]}{\exists x\ \varphi}\ \exists I$$

where t is free for x in φ

In mathematics, this corresponds to a proof of the form:

> "… φ[t/x]. So we have exhibited an x for which φ is true. Thus ∃xφ."

Example:

Derive $(\forall x)p(x) \vdash (\exists x)\ p(x)$

1.	$(\forall x)\ p(x)$	Premise
2.	$p(x)$	∀E 1
3.	$(\exists x)\ p(x)$	∃I 2

Note that, as with the ∨-Introduction rule, the ∃-Introduction rule *loses* information. So it will often *not* be the choice for the last step of a proof.

Finally, we have the **∃-Elimination rule**, which is analogous to the ∨-Elimination rule:

$$\frac{\exists x\ \varphi \qquad \begin{array}{c}[\varphi]\\ .\\ .\\ .\\ \psi\end{array}}{\psi}\ \exists E$$

provided that x is not free in ψ or in a hypothesis of the sub-derivation other than φ.

As with ∀I it is helpful to introduce a new variable for the x (actually it behaves like a constant) for *an* x such that φ. The mathematical proof form is:

"… ∃x φ. Let x_0 be an x such that φ. . . . ψ. Therefore ψ.

This is clearer with the tabular presentation:

Derive $(\forall x)(p(x) \rightarrow q(x)),\ (\exists x)\ p(x) \vdash (\exists x)\ q(x)$:

1.		$(\forall x)(p(x) \rightarrow q(x))$	Premise
2.		$(\exists x)\ p(x)$	Premise
3.	x_0	$p(x_0)$	Assumption
4.		$p(x_0) \rightarrow q(x_0)$	∀E 1
5.		$q(x_0)$	→E 3, 4
6.		$(\exists x)\ q(x)$	∃I 5
7.		$(\exists x)\ q(x)$	∃E 3-6

- In the ∃E rule, φ is identified with p(x), while ψ is identified with $(\exists x)\ q(x)$.
- Try not to be confused by the fact that ∃ is in the conclusion; The *original* x in 2 *was* eliminated!

Bad Example:

Derive $(\forall x)(p(x) \rightarrow q(x)),\ (\exists x)\ p(x) \vdash (\exists x)\ q(x)$:

1.		$(\forall x)(p(x) \rightarrow q(x))$	Premise
2.		$(\exists x)\ p(x)$	Premise
3.	x_0	$p(x_0)$	Assumption
4.		$p(x_0) \rightarrow q(x_0)$	∀E 1
5.		$q(x_0)$	→E 3, 4
6.		$q(x_0)$	∃E 3-5
7.		$(\exists x)\ q(x)$	∃I 6

- Formulas containing x_0 cannot be carried outside the box.
- The box for ∃E has two purposes:
  - Restricting the scope of the introduced variable.
  - Restricting the scope of the assumption.