Chapter 10

First-Order Logic

10.1 Overview

First-Order Logic is the calculus one usually has in mind when using the word ‘‘logic’’. It is expressive enough for all of mathematics, except for those concepts that rely on a notion of construction or computation. However, dealing with more advanced concepts is often somewhat awkward and researchers often design specialized logics for that reason.

Our account of first-order logic will be similar to the one of propositional logic. We will present

- The *syntax*, or the formal language of first-order logic, that is symbols, formulas, sub-formulas, formation trees, substitution, etc.
- The *semantics* of first-order logic
- *Proof systems* for first-order logic, such as the axioms, rules, and proof strategies of the first-order tableau method and refinement logic
- The *meta-mathematics* of first-order logic, which established the relation between the semantics and a proof system

In many ways, the account of first-order logic is a straightforward extension of propositional logic. One must, however, be aware that there are subtle differences.

10.2 Syntax

The syntax of first-order logic is essentially an extension of propositional logic by quantification $\forall$ and $\exists$. Propositional variables are replaced by *n-ary predicate symbols* (P, Q, R) which may be instantiated with either *variables* (x, y, z, ...) or *parameters* (a, b, ...). Here is a summary of the most important concepts.

(1) *Atomic formulas* are expressions of the form $Pc_1..c_n$ where P is an n-ary predicate symbol and the c_i are variables or parameters.

Note that many accounts of first-order logic use *terms* built from variables and *function symbols* instead of parameters. This makes the formal details a bit more complex.

(2) *Formulas* are built from atomic formulas using logical connectives and quantifiers.

Every atomic formula is a formula.

If A and B are formulas and x is a variable then (A), $\neg A$, $A \wedge B$, $A \vee B$, $A \Rightarrow B$, $(\forall x)A$, and $(\exists x)A$ are formulas.

(3) *Pure formulas* are formulas without parameters.

(4) The *degree* $d(A)$ of a formula A is the number of logical connectives and quantifiers in A.

(5) The *scope* of a quantifier is the smallest formula that follows the quantifier.

In $(\forall x)Px \vee Qx$ the scope of $(\forall x)$ is just Px, while Qx is outside the scope of the quantifier. To include Qx in the scope of $(\forall x)$ one has to add parentheses: $(\forall x)(Px \vee Qx)$.

Note that the conventions about the scope of quantifiers differ in the literature.

(6) *Free and bound variables* are defined similarly to Second-Order Propositional Logic

A variable x occurs bound in A if it occurs in the scope of a quantifier. Any other occurrence of x in A is free.

(7) *Closed formulas* (or sentences) are formulas without free variables.

This is the default from now on.

(8) *Substitution*: $A|_a^x$ (or $A[a/x]$) is the result of replacing every free occurrence of the variable x in A by the parameter a.

The technical definition is similar to the one for P^2. However, since the term being substituted for x does not contain variables, capture cannot occur.

(9) *Subformulas* are defined similar to propositional logic.

The only modification is that for any parameter a the formula $A|_a^x$ is an immediate subformula of $(\forall x)A$ and $(\exists x)A$.

(10) The *formation tree* of a formula F is a representation of all subformulas of A in tree format.

That is, the root of the tree is F.

The sucessor of a formula of the form $\neg A$ is A.

The successors of $A \wedge B$, $A \vee B$, $A \Rightarrow B$ are A and B.

The successors of $(\forall x)A$ and $(\exists x)A$ are $A|^x_{a_i}$ for all parameters a_i. Note that quantifiers usually have infinitely many successors.

Atomic formulas have no successors.

10.3 Semantics

The semantics of first-order logic, like the one of propositional logic and P^2, is based on a concept of valuations. In propositional logic, it was sufficient to assign values to all propositional variables and then extend the evaluation from atoms to formulas in a canonical fashion. In P^2, the semantics of quantified formulas was defined in terms of the values of all immediate subformulas: $v[(\forall p)A] = (v|^p_f)[A] \wedge_{\mathbb{B}} (v|^p_t)[A]$.

In first-order logic, we will proceed in the same way. However, since we don't have propositional variables anymore, we have to explain the meaning of atomic formulas first.

The standard approach is to interpret parameters by elements of some *universe* $\mathcal{U}$ and n-ary predicates by subsets of $\mathcal{U}^n$. A closed formula $Pa_1..a_n$ then expresses the fact that the interpretations $k_i \in \mathcal{U}$ of the a_i, taken together as n-tuple $(k_1,..,k_n)$, form an element of the interpretation of P.

Smullyan's approach is similar to the above idea but avoids set theory altogether. Instead, he introduces *$\mathcal{U}$-formulas*, where the elements of the universe $\mathcal{U}$ are used as parameters and defines *first-order valuations* as canonical extensions of boolean valuations on the set $\boldsymbol{E}^{\mathcal{U}}$ of all closed $\mathcal{U}$-formulas. The semantics of arbitrary formulas is then defined by a mapping φ from the set of parameters into $\mathcal{U}$.

Definition 10.3.1 *A first-order valuation v of $E^{\mathcal{U}}$ is an assignment of truth values to elements of $E^{\mathcal{U}}$ such that*

(1) *v is a boolean valuation of $E^{\mathcal{U}}$, i.e.*

$v[\neg A] = t$ *iff* $v[A] = f$

$v[A \wedge B] = t$ *iff* $v[A] = t$ *and* $v[B] = t$

$v[A \vee B] = t$ *iff* $v[A] = t$ *or* $v[B] = t$

$v[A \Rightarrow B] = t$ *iff* $v[A] = f$ *or* $v[B] = t$

(2) $v[(\forall x)A] = t$ *iff* $v[A|_k^x] = t$ *for every* $k \in \mathcal{U}$

$v[(\exists x)A] = t$ *iff* $v[A|_k^x] = t$ *for at least one* $k \in \mathcal{U}$

All valuations can be defined as canonical extensions of *atomic valuations*, i.e. assignments of truth values to the atomic formulas in $E^{\mathcal{U}}$. A *valuation tree* for a formula A is the formation tree of A together with a consistent assignment of truth values to all the nodes in that tree.

Note that since formation trees are usually infinite, one cannot expect to *compute* the truth value of a formula A solely on the basis of a given atomic valuation.

As in propositional logic, the semantics of formulas can also be described via via truth sets.

Definition 10.3.2 *A first-order truth set S (w.r.t. $\mathcal{U}$) is a subset of of $E^{\mathcal{U}}$ such that*

(1) *S statisfies the requirements on propositional truth sets, i.e.*

$A \in S$ *iff* $\neg A \notin S$

$A \wedge B \in S$ *iff* $A \in S$ *and* $B \in S$

$A \vee B \in S$ *iff* $A \in S$ *or* $B \in S$

$A \Rightarrow B \in S$ *iff* $A \notin S$ *or* $B \in S$

(2) $(\forall x)A \in S$ *iff* $A|_k^x \in S$ *for every* $k \in \mathcal{U}$

$(\exists x)A \in S$ *iff* $A|_k^x \in S$ *for at least one* $k \in \mathcal{U}$

It is easy (though tedious) to show that truth sets correspond to valuations in the sense that every first-order truth set is exactly the set of all formulas that are true under a fixed first-order valuation.

The definition of first-order valuations can be extended to sentences with parameters as follows. Let φ be a mapping from the set of parameters to $\mathcal{U}$. For a formula A define A^{φ} to be the result of replacing every parameter a_i in A by $\varphi(a_i)$. We say that A is *true under φ and v* if $v[A^{\varphi}] = t$.

The standard semantics of first-order formulas can be linked to the above as follows. Let E define the set of all closed formulas. An *interpretation of E* is a triple $I = (\mathcal{U}, \varphi, \iota)$, where $\mathcal{U}$ is an arbitrary

set, φ is a mapping from the set of parameters to $\mathcal{U}$, and ι is a function that maps each n-ary predicate symbol P to a set $I(P) \subseteq \mathcal{U}^n$ (or an n-ary relation over $\mathcal{U}$).

An atomic sentence $Pa_1..a_n$ is *true under* I if $(\varphi(a_1), ..\varphi(a_n)) \in \iota(P)$. In this manner, every interpretation induces an atomic valuation v_0 (together with φ), defined by $v_0[Pa_1..a_n^\varphi] = t$ iff $(\varphi(a_1), ..\varphi(a_n)) \in \iota(P)$, and vice versa ($\iota(P) = \{\varphi(a_1), ..\varphi(a_n)) \mid v_0[Pa_1..a_n^\varphi] = t\}$). From now on we will use whatever notion is more convenient.

A formula A is called *satisfiable* if it is true under at least one interpretation I (i.e. under at least one universe $\mathcal{U}$, one mapping φ, and one interpretation of the predicate symbols). I is also called a *model* of A. A is *valid* if A is true under every interpretation. These notions can be extended to sets of formula sin a canonical fashion.

It should be noted that there is a fine distinction between boolean valuations and first-order valuations. Boolean valuations can only analyze the propositional structure of formulas. They cannot evaluate quantified formulas and therefore have to treat them like propositional variables. In contrast to that first-order valuations can analyze the internals of quantified formulas and extract information that is unaccessible to boolean valuations.

For instance, a boolean valuations would interpret the logical structure of the formula $(\forall x)(Px \wedge Qx) \Rightarrow (\forall x)Px$ as $PQ \Rightarrow P$, which is obviously not a tautology. In contrast to that, every first-order valuation would go into the details of $(\forall x)(Px \wedge Qx)$ and $(\forall x)Px$ and evaluate to true. Thus the formula is valid, but not a tautology.

For the same reason, the formula $(\forall x)(Px \wedge Qx) \wedge (\exists x)(\neg Px)$ is truth-functionall satisfiable but not first-order satisfiable, since there is no first-order valuation (with a non-empty universe) that can make it true.

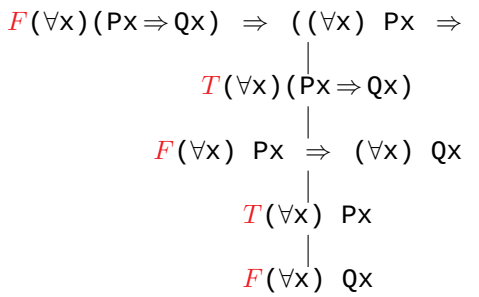
First-order valuations provide a more specific analysis than boolean valuations can give. They agree on quantifier-free formulas, however (Exercise!), and in that sense first-order logic is a canonical extension of propositional logic.

Chapter 11

First-Order Proof Systems

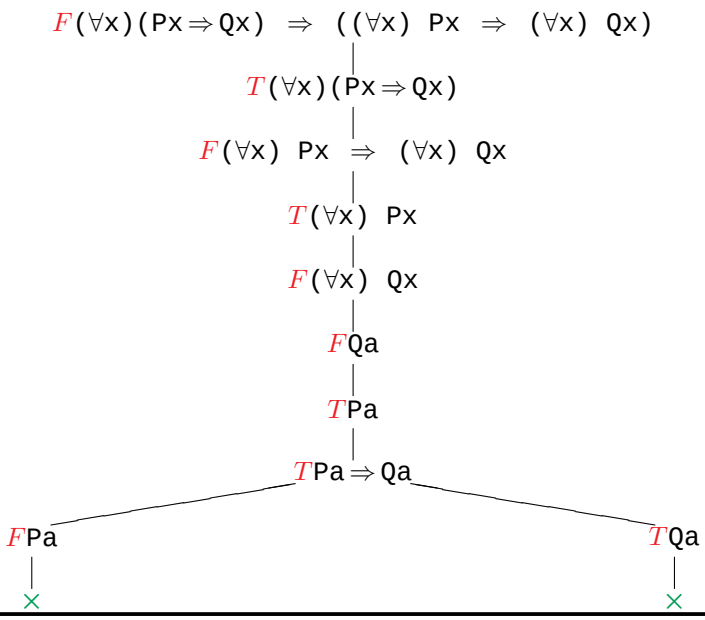
11.1 First-Order Tableaux

Since the evaluation of quantified formulas usually requires the evaluation of the formula for all possible elements of the universe, truth tables are unsuited for proving first-order formulas correct. Universes are usually infinite and even in a finite universe, the search space would quickly explode. The extension of the tableaux method to first-order logic, on the other hand, is quite straightforward. Let us consider an example.



Up to this point we have proceeded as in propositional logic. Now we have to start decomposing quantifiers. The formula $(\forall x)Qx$ is false if Qx can be made false for at least one element k of the universe. Since the elements of the universe do not belong to the syntax of the formulas, we substitute x by a parameter a instead.

In the following step we decompose $T(\forall x)Px$. We know that $(\forall x)Px$ is true if Px is true for all elements of the universe. This means we can substitute any parameter for x and we choose a again, since this is useful for completing the proof. The remaining proof is straightforward and we get



Q: *Why did we decompose F(∀x)Qx before T(∀x)Px in the proof?*

The parameter a that we substituted for x was supposed to indicate that Qx can be made false by some yet unknown element of the universe. Since we do not know this element, a should be a *new parameter* -- this way we make sure that we don't make any further assumptions about a by accidentally linking it to a parameter that was introduced earlier in the proof.

If we were to decompose T(∀x)Px *before* F(∀x)Qx then we would not be able to use a as parameter for Q, since it has already been used for P and is not unknown anymore. If we decompose F(∀x)Qx first, then a is still new. Choosing the same a for P is a decision we make afterwards.

In informal mathematics, quantifiers are handled in exactly the same way. When proving $(\forall x)(Px \wedge Qx) \Rightarrow (\forall x)Qx$ we assume $(\forall x)(Px \wedge Qx)$ and then try to show $(\forall x)Qx$. For this purpose we assume a to be arbitrary, but fixed, and try to prove Qa. Since we know $(\forall x)(Px \wedge Qx)$, we also know that $Pa \wedge Qa$ holds for the arbitrary a that we just chose and conclude that Qa is in fact the case. Note that it was crucial to have the a before instantiating $(\forall x)(Px \wedge Qx)$.

11.2 Extension of the unified notation

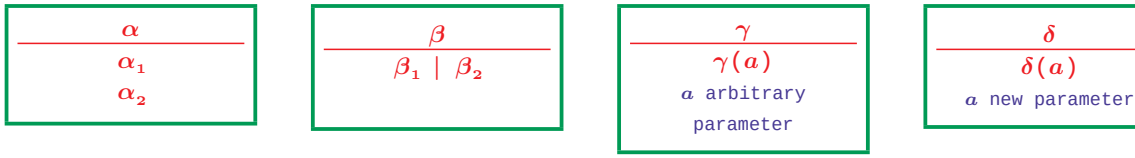
The above example shows that there are two different ways to handle quantifiers in tableaux proofs.

In the first case, we have formulas of the form T(∀x)A and, by duality, F(∃x)A, which we call formulas of type γ of *universal type*. γ-formulas are decomposed into TA[a/x] (and FA[a/x], respectively),

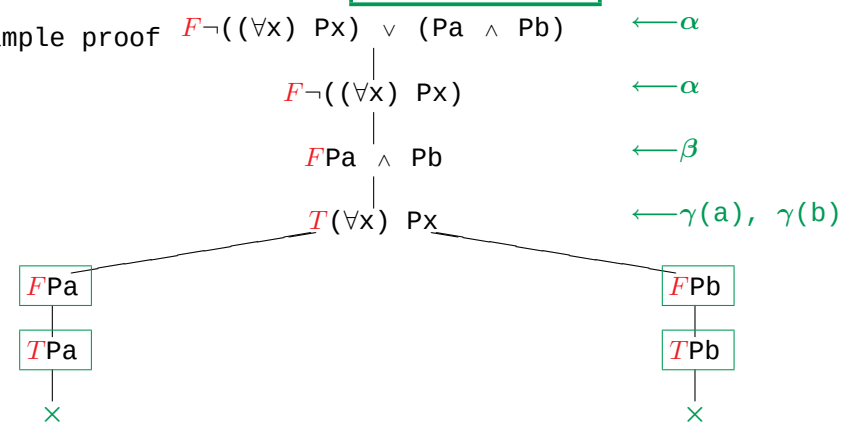
where a is an arbitrary parameter. These formulas are often denoted by $\gamma(a)$.

In the other case, we have formulas of the form $F(\forall x)A$ and, by duality, $T(\exists x)A$, which we call formulas of type δ of *existential type*. δ-formulas are decomposed into $FA[a/x]$ (and $TA[a/x]$, respectively), where a is a new parameter. These formulas are often denoted by $\delta(a)$ and the requirement that a must be new is usually called the *proviso* of the rule.

Altogether we have now four types of inference rules.[1]



Here is another example proof



Note that in this proof, the γ-formula $T(\forall x)$ Px had to be instantiated twice to complete the proof. In general, formulas of universal type may be used arbitrarily often in a proof and therefore validity in first-order logic is not decidable.[2]

11.3 A liberalized δ rule

The proviso of the δ rule, which requires a to be a new parameter, is quite restrictive and makes formal proofs more complicated than they have to be. Actually, the proviso is more restrictive than it has to be. It is possible to liberalize the δ rule by replacing it by the following requirement:

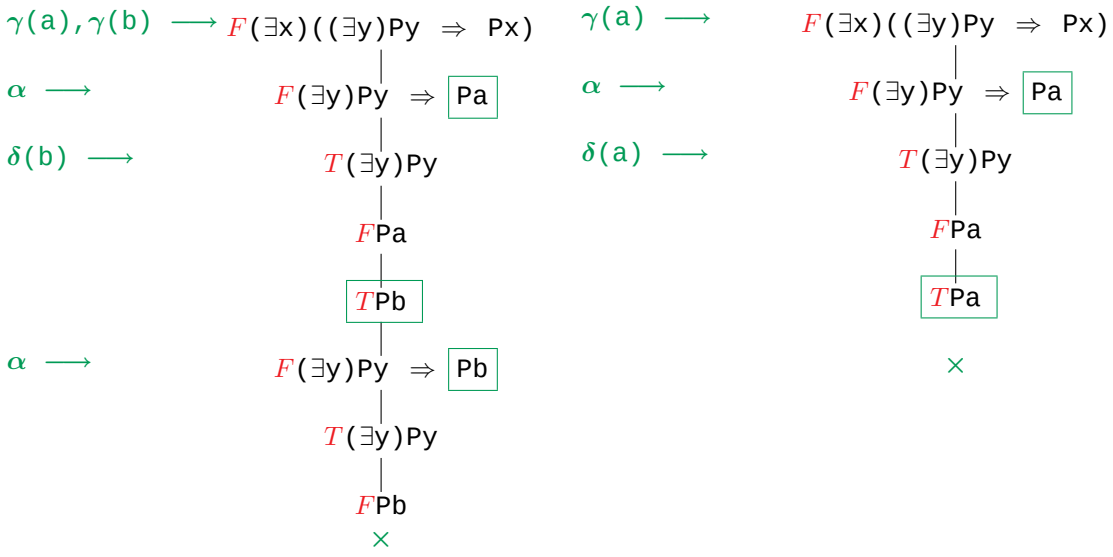
> provided a is a new parameter
> or a was not previously introduced on the same path by a δ rule, does not occur in δ, and no parameter in δ was previously generated by a δ rule

[1] In calculi that use terms instead of parameters, the γ-rule allows a to be an arbitrary term (representing some object) whereas in the δ rule a must be a new variable, representing the fact that the element of the universe is unknow.

[2] This argument only appeals to the intuition. The actual proof of the undecidability of first-order logic is more complex, since one has to show that there is no other way to determine that a formula is not valid.

In other words, if a does already occur in the proof then we may use it in a δ rule if it was generated by some γ rule. The rationale is that this γ rule could also be applied later and use the parameter a at that point ... after the δ rule has introduced it. Thus the fact that the γ rule appears earlier in the proof should not affect the parameters that the δ rule is permitted to use.

The following example shows the advantages of using a liberalized δ rule. In the proof on the left, the (original) the δ rule, which can only be applied after the first application of the γ rule, cannot use the parameter a because it already occurs in the proof. It has to use a new parameter b instead and we have to apply the γ rule again to get the formula FPb. Using the liberalized δ rule instead makes the proof on the right much shorter.



11.4 First-Order Gentzen Systems

Like analytic tableaux, Gentzen systems can easily be adapted to first-order logic. In propositional logic Gentzen systems were isomorphic to block tableaux, which in turn were just a different way of writing down tableau proofs -- keeping track of a set S of all formulas in the tree that still could be decomposed yet unused form. Thus we get the proof rules of first-order Gentzen systems for sequents with multiple conclusions by reformulating the tableau rules as block tableau rules and then converting these into the notation of Gentzen systems. The following table shows the block tableau rules on the left and the gentzen proof rules on the right.

T	F	L	R
γ $S,\ \mathsf{T}(\forall xA)$ $S,\ \mathsf{T}(A[t/x])$	δ $S,\ \mathsf{F}(\forall xA)$ $S,\ \mathsf{F}(A[a/x])$	$\forall\mathsf{L}$ $H,\ \forall xA \vdash G$ $H,\ A[t/x] \vdash G$	$\exists\mathsf{L}$ $H \vdash G,\ \forall xA$ $H \vdash G,\ A[a/x]$
δ $S,\ \mathsf{T}(\exists xA)$ $S,\ \mathsf{T}(A[a/x])$	γ $S,\ \mathsf{F}(\exists xA)$ $S,\ \mathsf{F}(A[t/x])$	$\forall\mathsf{R}$ $H,\ \exists xA \vdash G$ $H,\ A[a/x] \vdash G$	$\exists\mathsf{R}$ $H \vdash G,\ \exists xA$ $H \vdash G,\ A[t/x]$
The parameter t substituted for x can be chosen arbitrarily while a must be new			

Note that the same proviso applies to sequent proof rules as to tableau rules. Gentzen's original paper and most presentations of Gentzen systems use terms built from variables and function symbols instead of parameters. In that case a γ rule may substitute an arbitrary term t for x while the δ rule must choose a new (!) variable a.

11.5 First-Order Refinement Logic

The rules of first-order refinement logic can be extracted from those for multi-conclusioned sequents by dropping the extra conclusions, adopting a list notation, and adding a description of the evidence constructed by each rule. This leads to the following set of proof rules

Elimination (left)	Introduction (right)
$\mathsf{allL}\ i\ t$ $H,\ f{:}\forall xA,\ \Delta \vdash G$ $\mathsf{ev} = g[f(t)/pf]$ $H,\ f{:}\forall xA,\ pf{:}A[t/x],\ \Delta \vdash G$ $\mathsf{ev} = g[pf]$	$H \vdash \forall xA$ $\mathsf{ev} = \mathsf{fun}\ a \to pf[a]$ allR $H \vdash A[a/x]$ $\mathsf{ev} = pf[a]$
$\mathsf{exL}\ i$ $H,\ z{:}\exists xA,\ \Delta \vdash G$ $\mathsf{ev} = \mathsf{let}\ z{=}(a,pf)\ \mathsf{in}\ g[a,pf]$ $H,\ pf{:}A[a/x],\ \Delta \vdash G$ $\mathsf{ev} = g[a,pf]$	$H \vdash \exists xA$ $\mathsf{ev} = (t,pf)$ $\mathsf{exR}\ t$ $H \vdash A[t/x]$ $\mathsf{ev} = pf$
t can be an arbitrary term while a must be a new variable	

Since refinement logic is based on a term language we have rephrased the proviso accordingly.[3] Note that the rule allL explicitly re-introduces

[3] Terms are defined inductively: every variable x is a term and if f is an n-ary function symbol and $t_1, .., t_n$ are terms then $f(t_1, .., t_n)$ is a term. Constants are 0-ary function symbols applied to an empty list of terms and are written without the parentheses. Note that in most applications certain function symbols use a special (infix or other) syntax.

the assumption $\forall x A$ in the subgoal. The reason for this is that univerally quantified formulas in the assumptions may have to be instantiated several time in order to complete the proof. Any proof of $((\forall x) Px) \Rightarrow (Pa \wedge Pb)$ must instantiate the variable x with both a and b. If allL would drop the assumption $\forall x A$, then some proof attempts would not succeed as they cannot show both Pa and Pb. Here is a proof of that statement in refinement logic.

```
⊢ ((∀x)Px) ⇒ (Pa ∧ Pb)          by impR
  (∀x)Px ⊢ Pa ∧ Pb              by allL 1 a
   (∀x)Px, Pa ⊢ Pa ∧ Pb         by allL 1 b
    (∀x)Px, Pa, Pb ⊢ Pa ∧ Pb    by andR
[1]  (∀x)Px, Pa, Pb ⊢ Pa         by axiom 1
[2]  (∀x)Px, Pa, Pb ⊢ Pb         by axiom 2
```

Note that there is a different RL proof for `((∀x)Px) ⇒ (Pa ∧ Pb)` that uses `allL` only once in each branch of the proof and would succeed even if `allL` were to drop the univerally quantified formula. But the above example shows that `allL` would at least be irreversible. The fact that keeping the univerally quantified formulas is crucial for completeness is related to the use of a term structure instead of parameters. The formula `(((∀x)(P(x)⇒P(f(x)))) ∧ P(a)) ⇒ P(f(f(a)))` can only be proven if we instantiate `x` with `a` and then with `f(a)`. In this case both instances must occur in the same branch, since we need the first instance to prove `P(f(a))` from `P(a)` and the second to get `P(f(f(a)))` from the `P(f(a))` that we just proved.

The construction of evidence can be explained as follows:

`allR` In order to prove $\forall x A$ we have to prove $A[a/x]$ for an arbitrary (new) variable a. This will give us some proof evidence pf which will very likely depend on a. Since we must be able to do so without actually knowing a we will have constructed a function that given an arbitrary a will construct the evidence $pf[a]$. This function is sufficient evidence for the fact that A is true for all (arbitrary) x.

`allL` To prove a goal G by decomposing the assumption $\forall x A$ we introduce the assumption $A[t/x]$ for some term t that is given as parameter of the rule and prove G on that basis. This will give us some proof evidence g which may depend on the evidence pf for the truth of $A[t/x]$. Now if f is the function that describes the evidence for the assumption $\forall x A$, then $f(t)$ is evidence for $A[t/x]$ and can thus replace pf in the evidence $g[pf]$ for G.

`exR` In order to prove $\exists x A$ we have to prove $A[t/x]$ for a given term t. If pf is evidence for $A[t/x]$ then the combining this evidence with the information which term we used into a pair (t, pf) is evidence for the existence fact that A is true for some Element x.

`exL` To prove a goal G by decomposing the assumption $\exists x A$ we introduce the assumption $A[a/x]$ for some arbitrary (new) variable a (to indicate that we don't know for which specific Element a the assumption $A[a/x]$ holds) and prove G on that basis. This will give us some proof evidence g which may depend on the evidence pf for the truth of $A[t/x]$ and on the variable a. Now if z is a placeholder for the pair that describes the evidence for the assumption $\exists x A$, then z can be written as (a, pf) and a and pf can be used to construct $g[a, pf]$

Note that the above description links universal quantifiers to implication and existential quantifiers to conjunction and does not view them as generalized conjunction or generalized disjunction as it is commonly done. The reason is that we focus on the construction of evidence for truth instead of a possibly infinite (or even nondenumerable combination of formulas.