First-Order Logic is the calculus one usually has in mind when using the word "logic". It is widely used in mathematics and computer science because 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.

In our account of propositional logic we already made use of first-order logic to describe metamathematical concepts such as tautologies, soundness, or completeness. For instance, the definition of a formula X being a tautology, which requires the boolean value of X under an arbitrary assignment of values to the variables of X to be true, is expressed as $\forall$v:Var(X)$\to\mathbb{B}$. (bval(X,v)=t, and the goal of tableaux proofs (find a falsifying assignment) as $\exists$v:Var(X)$\to\mathbb{B}$. (bval(X,v)=f. Using formulas to represent metamathematical concepts makes the descriptions more precise and provides a foundation for the implementation of logical calculi on a computer.

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, subformulas, 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, first-order logic is a straightforward extension of propositional logic. One must, however, be aware that there are subtle differences.

13.1 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.

Definition 13.1 (Syntax of first-order logic)

(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.*

(2) *The formulas of first-order logic are recursively defined as follows*
   (a) Every atomic formula is a formula.
   (b) If A is a formula then so is $\neg A$.
   (c) If A and B are formulas then so are $(A \Rightarrow B)$, $(A \wedge B)$, and $(A \vee B)$.
   (d) If B is a formula and x a variable then $(\forall x)B$ and $(\exists x)B$ are formulas.

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

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

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

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 but the fundamental concepts remain the same.

Note that conventions about the scope of quantifiers differ in the literature and that many accounts of logic provide preference rules that permit dropping parentheses. As this usually leads to confusion we require that parentheses should always be used to avoid ambiguities. Outer parentheses and parentheses around quantified formulas may be omitted.

Thus 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: $(\forall x)(Px \vee Qx)$.

Definition 13.2 (Free and bound variables / Substitution)

(1) *A variable x occurs bound in A if it occurs in the scope of a quantifier $(\forall x)$ or $(\exists x)$. Any other occurrence of x in A is free.*

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

(3) *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.*[1]

(4) *Subformulas are defined similar to propositional logic. The only modification is that for any parameter a the formula $B|_a^x$ is an immediate subformula of $(\forall x)B$ and $(\exists x)B$.*

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

- *(a) The root of the tree is F.*
- *(b) The sucessor of a formula of the form $\neg A$ is A.*
- *(c) The successors of $A \wedge B$, $A \vee B$, $A \Rightarrow B$ are A and B.*
- *(d) The successors of $(\forall x)B$ and $(\exists x)B$ are $A|_{a_i}^x$ for all parameters a_i.*
  Note that quantifiers usually have infinitely many successors.
- *(e) Atomic formulas have no successors.*

From now on, closed formulas are the default when we use the word "formula".

13.2 Evidence Semantics

As in the propositional case, the meaning of formulas will be defined through evidence. A formula X is considered *valid* if we are able to provide evidence that justifies the validity of X. Evidence will described in the form of terms that can be used in a computational fashion. In a sense, evidence terms form a primitive functional programming language that can be executed on a computer.

[1] Since we only allow variables to be substituted by terms, issues like capture cannot occur

- Evidence for atomic formulas

  A predicate symbol P is a placeholder for some arbitrary unknown predicate and an atomic formula $A = Pc_1..c_n$ is a placeholder for some unknown proposition. As such, it cannot have a fixed evidence and the type of its evidences $[A]$ remains unspecified.[2]

- The evidence terms for $A \Rightarrow B$, $A \wedge B$, $A \vee B$, $\neg A$ are formal justifications for implications, conjunctions, disjunctions, and negations as in the case of propositional logic. They are summarized in the upper part of table **??**.

- Evidence for $(\forall x)B$

  To know $(\forall x)B$ we must know how to generate some evidence b for B for every possible instance of the variable x. Thus the evidence for $(\forall x)B$ must be a *function* f that generates the evidence $b : [B]$ for every possible value a that we may provide as input.

  To some extent the evidence for $(\forall x)B$ is similar to the one for $A \Rightarrow B$ as in both cases the type of evidences is a function type. There are, however, some important differences. While in the case of implication the input type is some evidence type $[A]$ the input for evidences for $(\forall x)B$ is taken from some (unspecified) *universe* of objects, which we denote by $\mathbb{U}$. Furthermore, since the formula B may contain x as free variable, there is some dependency between the *value of the input* $a : \mathbb{U}$ and the *type of the output* of the evidence function.

  Consider, for instance, the formula $(\forall x)(Pa \Rightarrow Px)$. Here the subformula $B = (Pa \Rightarrow Px)$ has evidence if we instantiate x with a and no evidence otherwise. So the evidence type for B clearly depends on the object we choose as input for the evidence function $f : [(\forall x)B]$.

  As a consequence, the evidence type of $(\forall x)B$ cannot be a simple function type like $\mathbb{U}{\to}[B]$ but has to be a function type that captures the dependency between input value and output type. In programming, such types are called *dependent function types*.

  Adopting programming notation we write $[(\forall x)B] = x : \mathbb{U}{\to}[B]$ [3]

  Let us look at two simple concrete examples.

  - $(\forall x)(Px \Rightarrow Px)$: The evidence must be a function $f : (x{:}\mathbb{U}{\to}([Px]{\to}[Px]))$ that takes some object x and produces evidence for $Px \Rightarrow Px$. The latter is a function that takes some evidence $p : [Px]$ and produces evidence for Px. Using identity function $\lambda p.\, p$ as solution for that the overall evidence has the form $\lambda x.\, (\lambda p.\, p)$.
  - $((\forall x)Px) \Rightarrow Pa$: The evidence must be a function that takes as input some evidence $f : [(\forall x)Px]$ and produces evidence for Pa. Since f has the type $x : \mathbb{U}{\to}[Px]$, applying f to the parameter a will result in an element of $[Pa]$, which we can use as the desired evidence. Thus the overall evidence will have the form $\lambda f.\, fa$.

- Evidence for $(\exists x)B$

  To know $(\exists x)B$ we must be able to generate evidence for B for at least one possible instance of the variable x. As in the case of the universal quantifier the formula B may contain x as free variable, which means that there is some dependency between element that instantiates x and the type of the evidence for B that we need to provide. Therefore we cannot simply

[2] This may change when we study advanced logics in which certain predicate symbols like the equality symbol have a predefined meaning. Atomic formulas like `0=0` will then have atomic evidences.

[3] Dependent function types are often written as $x : S{\to}T[x]$, where $T[x]$ expresses that the output type T may contain x as free variable. $x : S{\to}T[x]$ is the the type of functions that on input $s : S$ return a value of a type $T[s/x]$.

proposition A	evidence type $[A]$	evidence term	evidence decomposition
$A \Rightarrow B$	$[A] \rightarrow [B]$	$\lambda a.b$	$f(a)$
$A \wedge B$	$[A] \times [B]$	(a, b)	p_1, p_2
$A \vee B$	$[A] + [B]$	$\texttt{inl}(a), \texttt{inr}(b)$	$\texttt{case } e \texttt{ of inl}(a) \rightarrow s \mid \texttt{inr}(b) \rightarrow t$
$\neg A$	$[A] \rightarrow \{\}$	$\lambda a.b$	$f(a)$
$(\forall x)B$	$x : \mathbb{U} \rightarrow [B]$	$\lambda a.b$	$f(a)$
$(\exists x)B$	$x : \mathbb{U} \times [B]$	(a, b)	p_1, p_2

Table 1: Evidence terms for first-order logic

provide some element of $[B]$ as evidence for $(\exists x)B$ but we also have to provide the specific element $a : \mathbb{U}$ for which we claim B to be valid.

Therefore the evidence for $(\exists x)B$ must give us the concrete witness a *and* the evidence $b : B[a/x]$, that is a pair (a, b) where $a : \mathbb{U}$ and $b : [B[a/x]]$. Thus the type of evidences $[(\exists x)B]$ is a *dependent product type*.

We write $[(\exists x)B] = x : \mathbb{U} \times [B]$.

Again let us look at a few concrete examples

- $Pa \Rightarrow ((\exists x)Px)$: The evidence must be a function that takes some evidence $p : [Pa]$ and produces a pair (x, p') where $x : \mathbb{U}$ and p' is evidence for Px. The obvious choice is $x = a$ and $p' = p$ and the the overall evidence is $\lambda p.\,(a, p)$.
- $((\exists x)Px) \Rightarrow ((\exists y)Py)$: The evidence must be a function that takes as input an element $z : (x : \mathbb{U} \times [Px])$, i.e. a pair $z\ (a, p)$, where $a : \mathbb{U}$ and p is evidence for Pa, and produces a pair (x, p') where $x : \mathbb{U}$ and p' is evidence for Px. The obvious choice is again $x = a$ and $p' = p$ and the the overall evidence is $\lambda z.\,(z_1, z_2)$. Since (z_1, z_2) is identical to z we can simplify the evidence to $\lambda z.\,z$. [4]

Table **??** summarizes the evidence terms and types for the first-order quantifiers. Note that the evidence terms associated with universal quantifiers are the same as those associated with implication and that he evidence terms associated with existential quantifiers are the same as those associated with conjunction. The difference is only in the evidence types – quantifiers require dependent types while "ordinary"function and product types are sufficient for the propositional connectives.

We conclude this section by posing a few problems that should be elaborated in groups. For each of the formulas below the group should find the evidence that validates the formula or explain why evidence cannot be constructed.

(1) $((\forall x)(Px \wedge Qx)) \Rightarrow ((\forall x)Px \wedge (\forall x)Qx)$:

The evidence must be a function that takes as input a function $f : (x{:}\mathbb{U} \rightarrow [Px] \times [Qx])$ and produces a pair of functions $(g, h) : (x{:}\mathbb{U} \rightarrow [Px]) \times (x{:}\mathbb{U} \rightarrow [Qx])$. To construct g and g we take an input $x{:}\mathbb{U}$ and generate the evidence for $[Px]$ by applying f to x and selecting the first and second component. Thus the overall evidence is $\lambda f.\,(\lambda x.(f\,x)_1, \lambda x.(f\,x)_2)$

(2) $((\forall x)Px \wedge (\forall x)Qx) \Rightarrow ((\forall x)(Px \wedge Qx))$: The evidence is $\lambda z.\,(\lambda x.\,(z_1\,x, z_2\,x))$

[4] One could also attempt to argue semantically, claiming that $(\exists x)Px$ and $(\exists y)Py$ must express the same, and provide $\lambda x.x$ as overall evidence. But the justification for this claim is the argument that we just provided.

(3) $((\forall x)Px \vee (\forall x)Qx) \Rightarrow ((\forall x)(Px \vee Qx))$:

The evidence is $\lambda z.\,(\lambda x.\,(\texttt{case}\ z\ \texttt{of}\ \texttt{inl}(f) \rightarrow \texttt{inl}(f\,x)\ \mid\ \texttt{inr}(g) \rightarrow \texttt{inr}(g\,x)\,))$

(4) $((\forall x)(Px \vee Qx)) \Rightarrow ((\forall x)Px \vee (\forall x)Qx)$: This formula is not valid.

A possible counterexample is $P\,x$ being $\texttt{odd}(x)$ and $Q\,x$ being $\texttt{even}(x)$.

(5) $((\exists x)(Px \wedge Qx)) \Rightarrow ((\exists x)Px \wedge (\exists x)Qx)$: The evidence is $\lambda z.\,(\,(z_1, z_{21}), (z_1, z_{22})\,)$

(6) $((\exists x)Px \wedge (\exists x)Qx) \Rightarrow ((\exists x)(Px \wedge Qx))$: This formula is not valid.

A possible counterexample is $P\,x$ being $\texttt{odd}(x)$ and $Q\,x$ being $\texttt{even}(x)$.

(7) $((\exists x)Px \vee (\exists x)Qx) \Rightarrow ((\exists x)(Px \vee Qx))$:

The evidence is $\lambda z.\,(\texttt{case}\ z\ \texttt{of}\ \texttt{inl}(x) \rightarrow (x_1,\texttt{inl}(x_2))\ \mid\ \texttt{inr}(y) \rightarrow (y_1,\texttt{inr}(y_2))\,)$

(8) $((\exists x)(Px \vee Qx)) \Rightarrow ((\exists x)Px \vee (\exists x)Qx)$:

The evidence is $\lambda z.\,(\texttt{case}\ z_2\ \texttt{of}\ \texttt{inl}(x) \rightarrow \texttt{inl}(z_1, x)\ \mid\ \texttt{inr}(y) \rightarrow \texttt{inr}(z_1, y)\,)$

(9) $(\exists x)(Px \Rightarrow (\forall y)Py)$:

The evidence must be a pair (a, f) where $a{:}\mathbb{U}$ and f is a function that takes evidence p for Pa as input and generates a function $g : (x{:}\mathbb{U}{\rightarrow}[Px])$. There is no way to construct the generic "evidence function" g solely on the basis of the evidence for a specific Pa.

(10) $(\forall x)((\forall y)Py \Rightarrow Px)$: The evidence is $\lambda x.\,(\lambda f.\,f\,x)$

(11) $(\exists x)((\exists y)Py \Rightarrow Px)$:

The evidence must be a pair (a, f) where $a{:}\mathbb{U}$ and f is a function that takes a pair (b, p) : $(y{:}\mathbb{U}{\times}([Py])$ as input and generates evidence for Pa. The only way to construct f is to make sure that a matches its input b – then the evidence for Pa would be the second input component. There is no way to do this without knowing the input.

(12) $\neg((\exists x)Px) \Rightarrow ((\forall x)(((\exists y)Py) \Rightarrow Px))$: The evidence is $\lambda f.\,(\lambda x.\,(\lambda z.\,\texttt{any}(f\,z)))$

(13) $((\exists x)Px) \Rightarrow ((\forall x)(Px \Rightarrow Qx) \Rightarrow ((\exists y)Qy))$: The evidence is $\lambda z.\,(\lambda f.\,(z_1, (f\,z_1)\,z_2))$

(14) $\neg((\exists x)Px) \Rightarrow ((\forall x)\neg(Px))$:

The evidence must be a function that takes as input a function $f : (x{:}\mathbb{U}{\times}[Px]){\rightarrow}\{\}$ and produces a function $g : (x{:}\mathbb{U}{\rightarrow}([Px]{\rightarrow}\{\}))$. To construct g we take an input $x{:}\mathbb{U}$ and evidence $p : [Px]$ and create the element of $\{\}$ by applying f to the pair (x, p). Thus the overall evidence is $\lambda f.\,(\lambda x.\,(\lambda p.\,f(x, p)))$

(15) $((\forall x)\neg(Px)) \Rightarrow \neg((\exists x)Px)$: The evidence is $\lambda f.\,(\lambda z.\,(f z_1)z_2)$

(16) $((\exists x)Px) \Rightarrow \neg((\forall x)\neg(Px))$: The evidence is $\lambda z.\,(\lambda f.\,(f z_1)z_2)$

(17) $\neg((\forall x)Px) \Rightarrow ((\exists x)\neg(Px))$:

The evidence must be a function that takes as input a function $f : (x{:}\mathbb{U}{\rightarrow}[Px]){\rightarrow}\{\}$ and produces a pair $z : (x{:}\mathbb{U}{\times}([Px]{\rightarrow}\{\}))$. To construct g we have to find an element $a{:}\mathbb{U}$ and evidence p for Pa. There is no uniform way to construct these two evidences solely from f.

13.3 First-Order refinement proofs and evidence construction

Similar to evidence semantics, the proof calculus of refinement logic can be extended from propositional to first-order logic. Most rules follow immediately from an intuitive understanding of the logical operators. As in our discussion of evidence we investigate each connective separately.

- Refinement rules for atomic formulas

  An atomic formula $A = Pc_1..c_n$ cannot be decomposed into smaller components because it is a placeholder for an unknown proposition. But if A is also one of the assumptions in the hypotheses, we can prove A it using the `axiom` rule.

  $$H, a{:}A, H' \vdash A \quad \texttt{ev} = a \quad \texttt{by axiom}$$

- Refinement rules for $A \Rightarrow B$, $A \wedge B$, $A \vee B$, $\neg A$ are the refinement rules for implications, conjunctions, disjunctions, and negations that we already know from propositional logic. They are summarized in the upper part of table **??**.

- Refinement rules for $(\forall x)B$

  To prove a universally quantified formula $(\forall x)B$ we have to be able to prove B for every possible instance of the variable x. The only way to do this formally is to come up with a generic proof for B that does not depend on how x is being instantiated. For this purpose we substitute x by a *new* parameter a' that does not occur elsewhere in the sequent and prove $B[a'/x]$. The fact that a' is new makes sure that nothing is known about a' in the proof of $B[a'/x]$, which means that the proof will in fact apply to every possible instance of x.[5]

  $$\begin{array}{ll} H \vdash (\forall x)B & \texttt{by allR} \\ \quad H \vdash B[a'/x] & \end{array}$$

  To describe the evidence constructed by this rule we assume that we have some evidence b for the the subgoal sequent $H \vdash B[a'/x]$. This means that we must have found a generic way to generate the evidence b for $B[a'/x]$ for arbitrary a'. In other words, there is a function that on input a' computes b and this function must be the evidence for $(\forall x)B$. Using the λ-notation for evidence terms introduced previously, we can write the rule as

  $$\begin{array}{lll} H \vdash (\forall x)B & \texttt{ev} = \lambda a'.\, b & \texttt{by allR} \\ \quad H \vdash B[a'/x] & \texttt{ev} = b & \end{array}$$

  In addition to refining a universal quantifier in the conclusion of a sequent we also need rules for decomposing quantifiers in assumptions. To prove a goal C by decomposing the assumption $(\forall x)B$ we introduce the assumption $B[a/x]$ for some parameter a and prove C on that basis.

  For the construction of evidence let us assume that f is a label for $(\forall x)B$ in the main goal, that b is a label for $B[a/x]$ in the subgoal, and that c the evidence for C. The latter means that there is a function that computes evidence c for C from arbitrary evidences b for $B[a/x]$. Since $f(a)$ is a specific evidence for $B[a/x]$ we can apply this function, i.e. $\lambda b.c$, to $f(a)$, evaluate it and get the desired evidence for C in the main goal. If we integrate evidence construction into the rule we get

[5] In our description of the formal rules we use the stroke only to emphasize the fact that a' is new, which is a side condition on the parameters that may actually be chosen when the rule is applied. Apart from that condition, any parameter may be chosen.

$H, f{:}(\forall x)B, H' \vdash C$ ev = $c[f(a)/b]$ by allL a
$H, f{:}(\forall x)B, b{:}B[a/x], H' \vdash C$ ev = c

Note that the parameter a has to be provided when executing the rule since the parameter that instantiates the formula $(\forall x)B$ is not determined by the context of the rule. The user applying the rule has to choose it.

Let us look at two simple concrete examples.

– $(\forall x)(Px \Rightarrow Px)$: The refinement proof is straightforward.

$\vdash (\forall x)(Px \Rightarrow Px)$ ev = $\lambda a.\,(\lambda p.p)$ by allR
1 $\vdash Pa \Rightarrow Pa$ ev = $\lambda p.p$ by impliesR
1.1 $p{:}Pa \vdash Pa$ ev = p by axiom

– $((\forall x)Px) \Rightarrow Pa$: The obvious choice for instantiating x is the parameter a.

$\vdash ((\forall x)Px) \Rightarrow Pa$ ev = $\lambda f.\,f(a)$ by impliesR
1 $f{:}(\forall x)Px \vdash Pa$ ev = $p[f(a)/p]$ by allL a
1.1 $f{:}(\forall x)Px, p{:}Pa \vdash Pa$ ev = p by axiom

In both cases the constructed evidence is the same as we had in section **??**.

Note that the rule allL explicitly re-introduces the assumption $(\forall x)B$ in the subgoal. The reason for this is that univerally quantified formulas in the assumptions may have to be instantiated several times 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 $(\forall x)B$, 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.

$\vdash ((\forall x)Px) \Rightarrow (Pa \wedge Pb)$ ev = $\lambda f.(f(a), f(b))$ by impliesR
1 $f{:}(\forall x)Px \vdash Pa \wedge Pb$ ev = $(f(a), f(b))$ by allL a
1.1 $f{:}(\forall x)Px, p_a{:}Pa \vdash Pa \wedge Pb$ ev = $(p_a, f(b))$ by allL b
1.1.1 $f{:}(\forall x)Px, p_a{:}Pa, p_b{:}Pb \vdash Pa \wedge Pb$ ev = (p_a, p_b) by andR
1.1.1.1 $f{:}(\forall x)Px, p_a{:}Pa, p_b{:}Pb \vdash Pa$ ev = p_a by axiom
1.1.1.2 $f{:}(\forall x)Px, p_a{:}Pa, p_b{:}Pb \vdash Pb$ ev = p_b by axiom

- Refinement rules for $(\exists x)B$

To prove an existentially quantified formula $(\exists x)B$ we have to be able to prove $B[a/x]$ for some parameter a. If b is the evidence for $B[a/x]$ then combining this evidence with the parameter a is evidence for the existence of an x such that B holds.

$H \vdash (\exists x)B$ ev = (a, b) by exR a
$H \vdash B[a/x]$ ev = b

Again, the parameter a has to be provided when executing the rule.

To prove a goal C by decomposing the assumption $(\exists x)B$ we introduce the assumption $B[a'/x]$ for a new parameter a' and prove C on that basis. As in the case of allR requiring a' to be new makes sure that nothing is known about a' when we use $B[a'/x]$ to prove C except that a' does exist. For the construction of evidence let us assume that z is a label for $(\exists x)B$ in the main goal, b is the evidence for the assumption $B[a/x]$ in the subgoal, and c the evidence for C. Then z is assumed to be the same as the evidence pair (a', b), which means that replacing a by z_1 and b by z_2 in c gives us the evidence for the main goal.

$H, z{:}(\exists x)B, H' \vdash C$ ev = $c[z_1, z_2/a', b]$ by exL
$H, b{:}B[a'/x], H' \vdash C$ ev = c

left		right	
impliesL	$H, f{:}A \Rightarrow B, H' \vdash C$ ev = $c[f(a)/b]$ $H, f{:}A \Rightarrow B, H' \vdash A$ ev = a $H, b{:}B, H' \vdash C$ ev = c	$H \vdash A \Rightarrow B$ ev = $\lambda a.b$ $H, a{:}A \vdash B$ ev = b	impliesR
andL	$H, x{:}A \wedge B, H' \vdash C$ ev = $c[x_1, x_2/a, b]$ $H, a{:}A, b{:}B, H' \vdash C$ ev = c	$H \vdash A \wedge B$ ev = (a, b) $H \vdash A$ ev = a $H \vdash B$ ev = b	andR
orL	$H, x{:}A \vee B, H' \vdash C$ ev = case x of inl$(a) \to c_1$ | inr$(b) \to c_2$ $H, a{:}A, H' \vdash C$ ev = c_1 $H, b{:}B, H' \vdash C$ ev = c_2	$H \vdash A \vee B$ ev = inl(a) $H \vdash A$ ev = a $H \vdash A \vee B$ ev = inr(b) $H \vdash B$ ev = b	orR1 orR2
notL	$H, f{:}\neg A, H' \vdash C$ ev = any$(f(a))$ $H, f{:}\neg A, H' \vdash A$ ev = a	$H \vdash \neg A$ ev = $\lambda a.b$ $H, a{:}A \vdash$ f ev = b	notR
		$H, a{:}A, H' \vdash A$ ev = a	axiom
allL a	$H, f{:}(\forall x)B, H' \vdash C$ ev = $c[f(a)/b]$ $H, f{:}(\forall x)B, b{:}B[a/x], H' \vdash C$ ev = c	$H \vdash (\forall x)B$ ev = $\lambda a'.b$ $H \vdash B[a'/x]$ ev = b	allR
exL	$H, z{:}(\exists x)B, H' \vdash C$ ev = $c[z_1, z_2/a', b]$ $H, b{:}B[a'/x], H' \vdash C$ ev = c	$H \vdash (\exists x)B$ ev = (a, b) $H \vdash B[a/x]$ ev = b	exR a
a can be an arbitrary parameter while a' must be new			

Table 2: Rules of the first-order refinement calculus

Again let us look at a few concrete examples

- $Pa \Rightarrow ((\exists x)Px)$: Again a simple and straightforward proof

  $\vdash Pa \Rightarrow ((\exists x)Px)$ ev = $\lambda p.(a, p)$ by impliesR
  1 $p{:}Pa \vdash (\exists x)Px$ ev = (a, p) by exR a
  1.1 $p{:}Pa \vdash Pa$ ev = p by axiom

- $((\exists x)Px) \Rightarrow ((\exists y)Py)$:

  $\vdash ((\exists x)Px) \Rightarrow ((\exists y)Py)$ ev = $\lambda z.(z_1, z_2)$ by impliesR
  1 $z{:}(\exists x)Px \vdash (\exists y)Py$ ev = (z_1, z_2) by exL
  1.1 $p{:}Pa \vdash (\exists y)Py$ ev = (a, p) by exR a
  1.1.1 $p{:}Pa \vdash Pa$ ev = p by axiom

  Note that (z_1, z_2) is idential to z so the evidence constructed is the same as in section **??**. Note also that the order of rule applications is important. Had we used the rule exR a first we wouldn't be able to complete the proof.

  $\vdash ((\exists x)Px) \Rightarrow ((\exists y)Py)$ by impliesR
  1 $z{:}(\exists x)Px \vdash (\exists y)Py$ by exR a
  1.1 $z{:}(\exists x)Px \vdash Pa$ by ???

Table **??** summarizes all the rules of the refinement calculus for first order logic.

Exercises

As an exercise the following problems should be investigated in groups. For each of the formulas below the group should find a refinement proof and construct the evidence from the proof. In cases where no proof can be found, try to explain why the proof has to get stuck.

(1) $((\forall x)(Px \wedge Qx)) \Rightarrow ((\forall x)Px \wedge (\forall x)Qx)$:

$\vdash\ ((\forall x)(Px \wedge Qx)) \Rightarrow ((\forall x)Px \wedge (\forall x)Qx)$	by impliesR
1 $(\forall x)(Px \wedge Qx) \vdash ((\forall x)Px \wedge (\forall x)Qx)$	by andR
1.1 $(\forall x)(Px \wedge Qx) \vdash (\forall x)Px$	by allR
1.1.1 $(\forall x)(Px \wedge Qx) \vdash Pa$	by allL a
1.1.1.1 $(\forall x)(Px \wedge Qx), Pa \wedge Qa \vdash Pa$	by andL
1.1.1.1.1 $(\forall x)(Px \wedge Qx), Pa, Qa \vdash Pa$	by axiom
1.2 $(\forall x)(Px \wedge Qx) \vdash (\forall x)Qx$	by allR
1.2.1 $(\forall x)(Px \wedge Qx) \vdash Qa$	by allL a
1.2.1.1 $(\forall x)(Px \wedge Qx), Pa \wedge Qa \vdash Qa$	by andL
1.2.1.1.1 $(\forall x)(Px \wedge Qx), Pa, Qa \vdash Qa$	by axiom

The evidence extracted from this proof is $\lambda f.\,(\lambda x.(f\ x)_1, \lambda x.(f\ x)_2)$

(2) $((\forall x)Px \wedge (\forall x)Qx) \Rightarrow ((\forall x)(Px \wedge Qx))$:

$\vdash\ ((\forall x)Px \wedge (\forall x)Qx) \Rightarrow ((\forall x)(Px \wedge Qx))$	by impliesR
1 $(\forall x)Px \wedge (\forall x)Qx \vdash (\forall x)(Px \wedge Qx)$	by andL
1.1 $(\forall x)Px, (\forall x)Qx \vdash (\forall x)(Px \wedge Qx)$	by allR
1.1.1 $(\forall x)Px, (\forall x)Qx \vdash Pa \wedge Qa$	by andR
1.1.1.1 $(\forall x)Px, (\forall x)Qx \vdash Pa$	by allL a
1.1.1.1.1 $(\forall x)Px, (\forall x)Qx, Pa \vdash Pa$	by axiom
1.1.1.2 $(\forall x)Px, (\forall x)Qx \vdash Qa$	by allL a
1.1.1.2.1 $(\forall x)Px, (\forall x)Qx, Qa \vdash Qa$	by axiom

(3) $((\forall x)Px \vee (\forall x)Qx) \Rightarrow ((\forall x)(Px \vee Qx))$:

$\vdash\ ((\forall x)Px \vee (\forall x)Qx) \Rightarrow ((\forall x)(Px \vee Qx))$	by impliesR
1 $(\forall x)Px \vee (\forall x)Qx \vdash (\forall x)(Px \vee Qx)$	by orL
1.1 $(\forall x)Px \vee (\forall x)Qx \vdash Pa \vee Qa$	by allR
1.1.1 $(\forall x)Px \vdash Pa \vee Qa$	by allL a
1.1.1.1 $Pa \vdash Pa \vee Qa$	by orR1
1.1.1.1.1 $Pa \vdash Pa$	by axiom
1.1.2 $(\forall x)Qx \vdash Pa \vee Qa$	by
1.1.2.1 $Qa \vdash Pa \vee Qa$	by orR2
1.1.2.1.1 $Qa \vdash Qa$	by axiom

(4) $((\forall x)(Px \vee Qx)) \Rightarrow ((\forall x)Px \vee (\forall x)Qx)$: A proof attempt will get stuck

$\vdash\ ((\forall x)(Px \vee Qx)) \Rightarrow ((\forall x)Px \vee (\forall x)Qx)$	by impliesR
1 $(\forall x)(Px \vee Qx) \vdash (\forall x)Px \vee (\forall x)Qx$	by ???

At this point we have to prove either $(\forall x)Px$ or $(\forall x)Qx$ but there is no way to prove that.

(5) $((\exists x)(Px \wedge Qx)) \Rightarrow ((\exists x)Px \wedge (\exists x)Qx)$:

$\vdash\ ((\exists x)(Px \wedge Qx)) \Rightarrow ((\exists x)Px \wedge (\exists x)Qx)$	by impliesR
1 $(\exists x)(Px \wedge Qx) \vdash (\exists x)Px \wedge (\exists x)Qx)$	by exL
1.1 $Pa \wedge Qa \vdash (\exists x)Px \wedge (\exists x)Qx$	by andL
1.1.1 $Pa, Qa \vdash (\exists x)Px \wedge (\exists x)Qx$	by andR
1.1.1.1 $Pa, Qa \vdash (\exists x)Px$	by exR a
1.1.1.1.1 $Pa, Qa \vdash Pa$	by axiom
1.1.1.2 $Pa, Qa \vdash (\exists x)Qx$	by exR a
1.1.1.2.1 $Pa, Qa \vdash Qa$	by axiom

(6) $((\exists x)Px \wedge (\exists x)Qx) \Rightarrow ((\exists x)(Px \wedge Qx))$: Here is a proof attempt

$\vdash\ ((\exists x)Px \wedge (\exists x)Qx) \Rightarrow ((\exists x)(Px \wedge Qx))$	by impliesR
1 $\quad (\exists x)Px \wedge (\exists x)Qx \vdash (\exists x)(Px \wedge Qx)$	by andL
1.1 $\quad (\exists x)Px, (\exists x)Qx \vdash (\exists x)(Px \wedge Qx)$	by exL
1.1.1 $\quad Pa, (\exists x)Qx \vdash (\exists x)(Px \wedge Qx)$	by exL
1.1.1 $\quad Pa, Qb \vdash (\exists x)(Px \wedge Qx)$	by ???

The proof gets stuck because in the second application of exL we will have to use a *new* parameter instead of using a again.

(7) $((\exists x)Px \vee (\exists x)Qx) \Rightarrow ((\exists x)(Px \vee Qx))$:

$\vdash\ ((\exists x)Px \vee (\exists x)Qx) \Rightarrow ((\exists x)(Px \vee Qx))$	by impliesR
1 $\quad (\exists x)Px \vee (\exists x)Qx \vdash (\exists x)(Px \vee Qx)$	by orL
1.1 $\quad (\exists x)Px \vdash (\exists x)(Px \vee Qx)$	by exL
1.1.1 $\quad Pa \vdash (\exists x)(Px \vee Qx)$	by exR a
1.1.1.1 $\quad Pa \vdash Pa \vee Qa$	by orR1
1.1.1.1.1 $\quad Pa \vdash Pa$	by axiom
1.2 $\quad (\exists x)Qx \vdash (\exists x)(Px \vee Qx)$	by exL
1.2.1 $\quad Qa \vdash (\exists x)(Px \vee Qx)$	by exR a
1.2.1.1 $\quad Qa \vdash Pa \vee Qa$	by orR2
1.2.1.1.1 $\quad Qa \vdash Qa$	by axiom

(8) $((\exists x)(Px \vee Qx)) \Rightarrow ((\exists x)Px \vee (\exists x)Qx)$:

$\vdash\ ((\exists x)(Px \vee Qx)) \Rightarrow ((\exists x)Px \vee (\exists x)Qx)$	by impliesR
1 $\quad (\exists x)(Px \vee Qx) \vdash (\exists x)Px \vee (\exists x)Qx$	by exL
1.1 $\quad Pa \vee Qa \vdash (\exists x)Px \vee (\exists x)Qx$	by orL
1.1.1 $\quad Pa \vdash (\exists x)Px \vee (\exists x)Qx$	by orR1
1.1.1.1 $\quad Pa \vdash (\exists x)Px$	by exR a
1.1.1.1.1 $\quad Pa \vdash Pa$	by axiom
1.1.2 $\quad Pa \vdash (\exists x)Px \vee (\exists x)Qx$	by orR1
1.1.2.1 $\quad Pa \vdash (\exists x)Px$	by exR a
1.1.2.1.1 $\quad Pa \vdash Pa$	by axiom

(9) $(\exists x)(Px \Rightarrow (\forall y)Py)$: Here is a proof attempt

$\vdash\ (\exists x)(Px \Rightarrow (\forall y)Py)$	by exR a
1 $\quad \vdash Pa \Rightarrow (\forall y)Py$	by impliesR
1.1 $\quad Pa \vdash (\forall y)Py$	by allR
1.1.1 $\quad Pa \vdash Pb$	by ???

The proof gets stuck because in the application of allR we will have to use a *new* parameter instead of using a again.

(10) $(\forall x)((\forall y)Py \Rightarrow Px)$:

$\vdash\ (\forall x)((\forall y)Py \Rightarrow Px)$	by allR
1 $\quad \vdash (\forall y)Py \Rightarrow Pa$	by impliesR
1.1 $\quad (\forall y)Py \vdash Pa$	by allL a
1.1.1 $\quad Pa \vdash Pa$	by axiom

(11) $(\exists x)((\exists y)Py \Rightarrow Px)$: Here is a proof attempt

$\vdash\ (\exists x)((\exists y)Py \Rightarrow Px)$	by exR a
1 $\quad \vdash (\exists y)Py \Rightarrow Pa$	by impliesR
1.1 $\quad (\exists y)Py \vdash Pa$	by exL
1.1.1 $\quad Pb \vdash Pa$	by ???

The proof gets stuck because in the application of exL we will have to use a *new* parameter instead of using a again.

(12) $\neg((\exists x)Px) \Rightarrow ((\forall x)(((\exists y)Py) \Rightarrow Px))$:

$\vdash\ \neg((\exists x)Px) \Rightarrow ((\forall x)(((\exists y)Py) \Rightarrow Px))$	by impliesR
1 $\neg((\exists x)Px) \vdash (\forall x)(((\exists y)Py) \Rightarrow Px)$	by allR
1.1 $\neg((\exists x)Px) \vdash ((\exists y)Py) \Rightarrow Pa$	by impliesR
1.1.1 $\neg((\exists x)Px), (\exists y)Py \vdash Pa$	by notL
1.1.1.1 $\neg((\exists x)Px), (\exists y)Py \vdash (\exists x)Px$	by axiom

(13) $((\exists x)Px) \Rightarrow ((\forall x)(Px \Rightarrow Qx) \Rightarrow ((\exists y)Qy))$:

$\vdash\ ((\exists x)Px) \Rightarrow ((\forall x)(Px \Rightarrow Qx) \Rightarrow ((\exists y)Qy))$	by impliesR
1 $(\exists x)Px \vdash (\forall x)(Px \Rightarrow Qx) \Rightarrow ((\exists y)Qy)$	by impliesR
1.1 $(\exists x)Px, (\forall x)(Px \Rightarrow Qx) \vdash (\exists y)Qy$	by exL
1.1.1 $Pa, (\forall x)(Px \Rightarrow Qx) \vdash (\exists y)Qy$	by allL a
1.1.1.1 $Pa, Pa \Rightarrow Qa \vdash (\exists y)Qy$	by exR a
1.1.1.1.1 $Pa, Pa \Rightarrow Qa \vdash Qa$	by impliesL
1.1.1.1.1.1 $Pa, Pa \Rightarrow Qa \vdash Pa$	by axiom
1.1.1.1.1.2 $Pa, Pa \Rightarrow Qa, Qa \vdash Qa$	by axiom

(14) $\neg((\exists x)Px) \Rightarrow ((\forall x)\neg(Px))$:

$\vdash\ \neg((\exists x)Px) \Rightarrow ((\forall x)\neg(Px))$	by impliesR
1 $\neg((\exists x)Px) \vdash (\forall x)\neg(Px)$	by allR
1.1 $\neg((\exists x)Px) \vdash \neg(Pa)$	by notR
1.1.1 $\neg((\exists x)Px), Pa \vdash$ f	by notL
1.1.1.1 $\neg((\exists x)Px), Pa \vdash (\exists x)Px$	by exR a
1.1.1.1.1 $\neg((\exists x)Px), Pa \vdash Pa$	by axiom

(15) $((\forall x)\neg(Px)) \Rightarrow \neg((\exists x)Px)$:

$\vdash\ ((\forall x)\neg(Px)) \Rightarrow \neg((\exists x)Px)$	by impliesR
1 $(\forall x)\neg(Px)) \vdash \neg((\exists x)Px)$	by notR
1.1 $(\forall x)\neg(Px)), (\exists x)Px \vdash$ f	by exL
1.1.1 $(\forall x)\neg(Px)), Pa \vdash$ f	by allL a
1.1.1.1 $\neg(Pa), Pa \vdash$ f	by notL
1.1.1.1.1 $\neg(Pa), Pa \vdash Pa$	by axiom

(16) $((\exists x)Px) \Rightarrow \neg((\forall x)\neg(Px))$:

$\vdash\ ((\exists x)Px) \Rightarrow \neg((\forall x)\neg(Px)$	by impliesR
1 $(\exists x)Px \vdash \neg((\forall x)\neg(Px)$	by exL
1.1 $Pa \vdash \neg((\forall x)\neg(Px)$	by notR
1.1.1 $Pa, (\forall x)\neg(Px) \vdash$ f	by allL a
1.1.1.1 $Pa, \neg(Pa) \vdash$ f	by notL
1.1.1.1.1 $Pa, \neg(Pa) \vdash Pa$	by axiom

(17) $\neg((\forall x)Px) \Rightarrow ((\exists x)\neg(Px))$: Here is a proof attempt

$\vdash\ \neg((\forall x)Px) \Rightarrow ((\exists x)\neg(Px)))$	by impliesR
1 $\vdash\ \neg((\forall x)Px) \vdash ((\exists x)\neg(Px)))$	by ???

At this point we're stuck. If we apply notL we will lose the conclusion $((\exists x)\neg(Px)))$ and have to prove $(\forall x)Px$, which clearly won't work. But there are no other proof rule that can be applied here.