The boolean semantics of first-order logic, like the one of propositional logic, 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 by showing how to calculate the value of a composed formula from values of the subformulas. In first-order logic, our starting point has to be atomic formulas instead of propositional variables and we have to explain how to calculate the value of quantified formulas that may have infinitely many subformulas.

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 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 14.1 (first-order valuations)

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

$v[(\exists x)B]$ = *t iff* $v[B|^x_k]$ = *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 14.2 *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) $(\forall x)B \in S$ *iff* $B|^x_k \in S$ *for every* $k \in \mathcal{U}$

$(\exists x)B \in S$ *iff* $B|^x_k \in S$ *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-functionally 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.

14.1 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.

$$\begin{tikzcd}
F(\forall x)(Px \Rightarrow Qx) \Rightarrow ((\forall x)Px \Rightarrow (\forall x)Qx) \arrow[d, dash] \\
T(\forall x)(Px \Rightarrow Qx) \arrow[d, dash] \\
F(\forall x)Px \Rightarrow (\forall x)Qx \arrow[d, dash] \\
T(\forall x)Px \arrow[d, dash] \\
F(\forall x)Qx
\end{tikzcd}$$

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

$$\begin{tikzcd}
 & F(\forall x)(Px \Rightarrow Qx) \Rightarrow ((\forall x)Px \Rightarrow (\forall x)Qx) \arrow[d, dash] & \\
 & T(\forall x)(Px \Rightarrow Qx) \arrow[d, dash] & \\
 & F(\forall x)Px \Rightarrow (\forall x)Qx \arrow[d, dash] & \\
 & T(\forall x)Px \arrow[d, dash] & \\
 & F(\forall x)Qx \arrow[d, dash] & \\
 & FQa \arrow[d, dash] & \\
 & TPa \arrow[d, dash] & \\
 & TPa \Rightarrow Qa \arrow[dl, dash] \arrow[dr, dash] & \\
FPa \arrow[d, dash] & & TQa \arrow[d, dash] \\
\times & & \times
\end{tikzcd}$$

Q: $\boxed{\textit{Why did we decompose } F(\forall x)Qx \textit{ before } T(\forall x)Px \textit{ in the proof?}}$

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 $T(\forall x)Px$ before $F(\forall x)Qx$ 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 $F(\forall x)Qx$ 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 $(\forall x)(Px \wedge Qx)$.

14.2 Extension of the unified notation

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

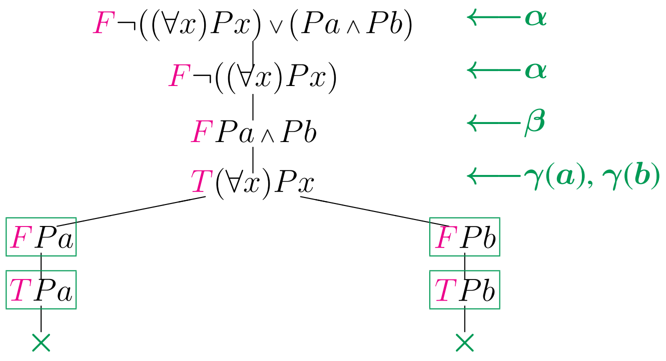
In the first case, we have formulas of the form $T(\forall x)A$ and, by duality, $F(\exists x)A$, which we call formulas of type γ of *universal type*. γ-formulas are decomposed into $TB[a/x]$ (and $FB[a/x]$, respectively), where a is an arbitrary parameter. These formulas are often denoted by $\gamma(a)$.

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 $FB[a/x]$ (and $TB[a/x]$, 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]

$$\boxed{\begin{array}{c}\alpha\\ \hline \alpha_1\\ \alpha_2\end{array}} \qquad \boxed{\begin{array}{c}\beta\\ \hline \beta_1 \mid \beta_2\end{array}} \qquad \boxed{\begin{array}{c}\gamma\\ \hline \gamma(a)\\ a \text{ arbitrary parameter}\end{array}} \qquad \boxed{\begin{array}{c}\delta\\ \hline \delta(a)\\ a \text{ new parameter}\end{array}}$$

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]

14.3 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:

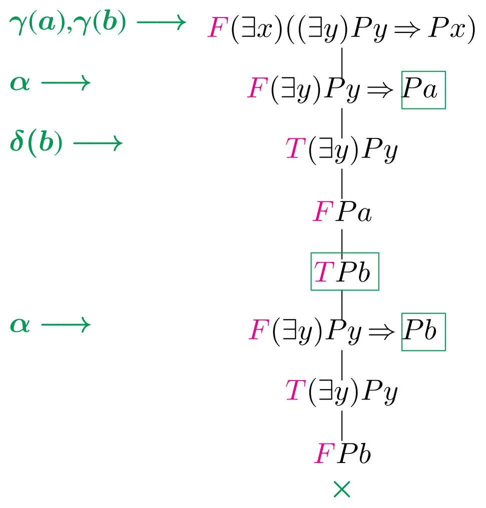
> provided a is a new parameter
> or a was not previously introduced on the same path by a δ rule, does not occur in δ,
> and no parameter in δ was previously generated by a δ rule

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.

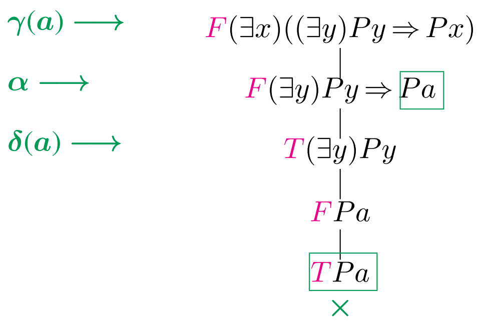
[1]In calculi that use terms instead of parameters, the γ-rule allows a to be an arbitrary term (representing some object) whereas in the δ rule a must be a new variable, representing the fact that the element of the universe is unknow.

[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.

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.



$\gamma(a),\gamma(b) \longrightarrow$ $F(\exists x)((\exists y)Py \Rightarrow Px)$
$\alpha \longrightarrow$ $F(\exists y)Py \Rightarrow \boxed{Pa}$
$\delta(b) \longrightarrow$ $T(\exists y)Py$
FPa
$\boxed{TPb}$
$\alpha \longrightarrow$ $F(\exists y)Py \Rightarrow \boxed{Pb}$
$T(\exists y)Py$
FPb
$\times$



$\gamma(a) \longrightarrow$ $F(\exists x)((\exists y)Py \Rightarrow Px)$
$\alpha \longrightarrow$ $F(\exists y)Py \Rightarrow \boxed{Pa}$
$\delta(a) \longrightarrow$ $T(\exists y)Py$
FPa
$\boxed{TPa}$
$\times$

14.4 Transforming tableaux proofs into Refinement logic proofs

In propositional logic we could use booleanization to relate the truth table semantics of tableaux to the evidence semantics of refinement logic. This technique can be extended to first-order logic if the domains under consideration are finite. In this case a universally quantified formula $(\forall x)B$ corresponds to the formula $B[a_1/x] \wedge ... \wedge B[a_n/x]$ and a universally quantified formula $(\exists x)B$ to $B[a_1/x] \vee ... \vee B[a_n/x]$, where $a_1, .., a_n$ are (representatives of) all the elements of the universe, and the decidability of atomica formulas would propagate to all formulas of first-order logic.

Beyond finite domains all bets are off. We cannot guarantee the decidability of formulas anymore and evidence will be increasingly hard or impossible to construct. To what extent the booleanization of evidence can be extended to first-order formulas in these cases stillneeds to be researched.

If the decidability of formulas can be guaranteed, we can translate tableaux proofs into refinement logic proofs using the same method as in propositional logic. We convert tableaux into block tableaux, convert the block tableaux rules into rules that generate proofs with only one F-formula, and perform a syntax translation that separates the T-formulas from the F-formulas by putting a between them and then drops the signs. This leads to decomposition rules that are mostly identical to the rules of the propositional refinement calculus, except for the rules `orR1`,, `orR2`, `impliesL`, `notL`, and – as the example of $(\exists x)((\exists y)Py \Rightarrow Px)$ in Section 14.3 shows – `exR`.[3] For these rules we provide refinement logic proof fragments that simulate their behavior. The proof fragments for `orR1`*, `orR2`*, `impliesL`*, and `notL`* have already been discussed in our account of propositional logic. The simulation of the rule `exR`* is given below.

`exR`*:		$H \vdash (\exists x)B$	by `Use_decidability_of` $(\exists x)B$
	`1`	$H,\ ((\exists x)B) \vee \neg((\exists x)B) \vdash (\exists x)B$	by `orL`
	`1.1`	$H,\ (\exists x)B \vdash (\exists x)B$	by `axiom`
	`1.2`	$H,\ \neg((\exists x)B) \vdash (\exists x)B$	by `exR` a
	`1.2.1`	$H,\ \neg((\exists x)B) \vdash B[a/x]$	

	T	F	
α	$S, TA \wedge B$ S, TA, TB	$S, FA \wedge B$ S, FA S, FB	β
β	$S, TA \vee B$ S, TA S, TB	$S, FA \vee B$ $S, FA\,, FB$	α
β	$S, TA \Rightarrow B$ S, FA S, TB	$S, FA \Rightarrow B$ S, TA, FB	α
α	$S, T\neg A$ S, FA	$S, F\neg A$ S, TA	α
$*$	S, TA, FA		
δ	$S, T(\exists x)B$ $S, TB[a'/x]$	$S, F(\exists x)B$ $S, FB[a/x]$	γ
γ	$S, T(\forall x)B$ $S, TB[a/x]$	$S, F(\forall x)B$ $S, FB[a'/x]$	δ

$\mapsto$

	left	right	
andL	$S, A \wedge B \vdash C$ $S, A, B \vdash C$	$S \vdash A \wedge B$ $S \vdash A$ $S \vdash B$	andR
orL	$S, A \vee B \vdash C$ $S, A \vdash C$ $S, B \vdash C$	$S \vdash A \vee B$ $S, \neg A \vdash \mathrm{B}$ $S \vdash A \vee B$ $S, \neg B \vdash A$	orR1* orR2*
impliesL*	$S, A \Rightarrow B \vdash C$ $S, \neg C \vdash A$ $S, B \vdash C$	$S \vdash A \Rightarrow B$ $S, A \vdash B$	impliesR
notL*	$S, \neg A \vdash C$ $S, \neg C \vdash A$	$S \vdash \neg A$ $S, A \vdash \mathsf{f}$	notR
axiom	$S, A \vdash A$		
exL	$S, (\exists x)B \vdash C$ $S, B[a'/x] \vdash C$	$S \vdash (\exists x)B$ $S, \neg(\exists x)B \vdash B[a/x]$	exR* a
allL a	$S, (\forall x)B \vdash C$ $S, B[a/x] \vdash C$	$S \vdash (\forall x)B$ $S \vdash B[a'/x]$	allR

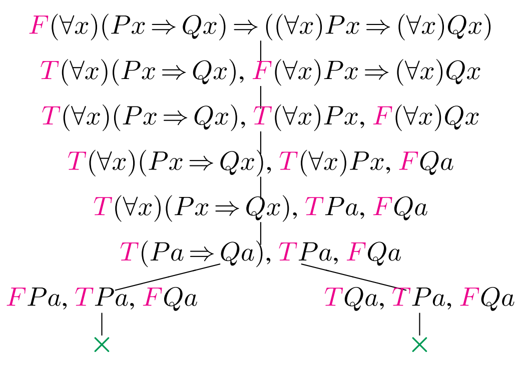
Table 1: Translation of tableaux rules into refinement rules

All other rules do not introduce additional F-formulas and immediately translat into the corresponding rules of refinement logic. Table 1 summarizes the translation of the tableaux rules into refinement rules.

We conclude this section with a few examples.

Example 14.3 (Translation of tableaux roofs into refinement logic)

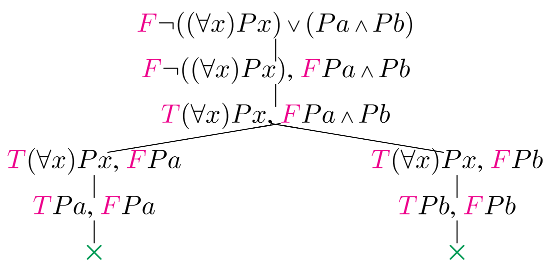
The block tableau for $(\forall x)(Px \Rightarrow Qx) \Rightarrow ((\forall x)Px \Rightarrow (\forall x)Qx)$ translates immediately without requiring decidabilities


$F(\forall x)(Px \Rightarrow Qx) \Rightarrow ((\forall x)Px \Rightarrow (\forall x)Qx)$
$T(\forall x)(Px \Rightarrow Qx), F(\forall x)Px \Rightarrow (\forall x)Qx$
$T(\forall x)(Px \Rightarrow Qx), T(\forall x)Px, F(\forall x)Qx$
$T(\forall x)(Px \Rightarrow Qx), T(\forall x)Px, FQa$
$T(\forall x)(Px \Rightarrow Qx), TPa, FQa$
$T(Pa \Rightarrow Qa), TPa, FQa$
FPa, TPa, FQa
TQa, TPa, FQa
$\times$
$\times$

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

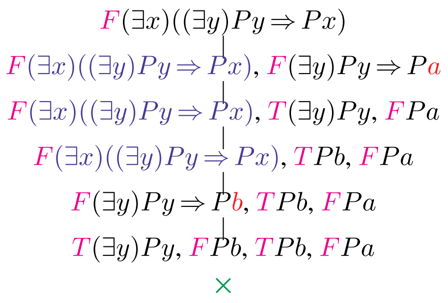
[3]In principle, all signed formulas may be decomposed multiple times in a tableau proof, since the tableaux calculus does not explicitly exclude that. But this option has a significant effect only in the case of the γ-rules, where it enables us to instantiate the γ-subformulas several times with different parameters. As a result the $T\forall$-rule will create multiple T-formulas, which is captured in the allL rule for refinement logic, and the $F\exists$-rule will create multiple F-formulas, which is not permitted for exR.

The block tableau for $\neg((\forall x)Px) \vee (Pa \wedge Pb)$ and its translation are shown below. The translation needs to preserve the disjunct eliminated by the application of orR2. We use Decide A as abbreviation for the proof fragment consisting of an application of the rule `Use_decidability_of` A followed immediately by `orL`.


$F\neg((\forall x)Px) \vee (Pa \wedge Pb)$
$F\neg((\forall x)Px)$, $F Pa \wedge Pb$
$T(\forall x)Px$, $F Pa \wedge Pb$
$T(\forall x)Px$, $F Pa$
$T(\forall x)Px$, $F Pb$
$T Pa$, $F Pa$
$T Pb$, $F Pb$
$\times$
$\times$

$\vdash\ \neg((\forall x)Px) \vee (Pa \wedge Pb)$	`by Decide` $(\forall x)Px$
`1` $(\forall x)Px\ \vdash\ \neg((\forall x)Px) \vee (Pa \wedge Pb)$	`by orR2`
`1.1` $(\forall x)Px\ \vdash\ Pa \wedge Pb$	`by andR`
`1.1.1` $(\forall x)Px\ \vdash\ Pa$	`by allL` a
`1.1.1.1` $(\forall x)Px, Pa\ \vdash\ Pa$	`by axiom`
`1.1.2` $(\forall x)Px\ \vdash\ Pb$	`by allL` b
`1.1.2.1` $(\forall x)Px, Pb\ \vdash\ Pb$	`by axiom`
`2` $\neg((\forall x)Px)\ \vdash\ \neg((\forall x)Px) \vee (Pa \wedge Pb)$	`by orR1`
`2.1` $\neg((\forall x)Px)\ \vdash\ \neg((\forall x)Px)$	`by axiom`

The block tableau for $(\exists x)((\exists y)Py \Rightarrow Px)$ and its translation are shown below. The translation needs to preserve the conclusion which is eliminated by the application of exR.


$F(\exists x)((\exists y)Py \Rightarrow Px)$
$F(\exists x)((\exists y)Py \Rightarrow Px)$, $F(\exists y)Py \Rightarrow Pa$
$F(\exists x)((\exists y)Py \Rightarrow Px)$, $T(\exists y)Py$, $F Pa$
$F(\exists x)((\exists y)Py \Rightarrow Px)$, $T Pb$, $F Pa$
$F(\exists y)Py \Rightarrow Pb$, $T Pb$, $F Pa$
$T(\exists y)Py$, $F Pb$, $T Pb$, $F Pa$
$\times$

$\vdash\ (\exists x)((\exists y)Py \Rightarrow Px)$	`by Decide` $(\exists x)((\exists y)Py \Rightarrow Px)$
`1` $(\exists x)((\exists y)Py \Rightarrow Px)\ \vdash\ (\exists x)((\exists y)Py \Rightarrow Px)$	`by axiom`
`2` $\neg(\exists x)((\exists y)Py \Rightarrow Px)\ \vdash\ (\exists x)((\exists y)Py \Rightarrow Px)$	`by exR` a
$\neg(\exists x)((\exists y)Py \Rightarrow Px)\ \vdash\ (\exists y)Py \Rightarrow Pa$	`by impliesR`
$\neg(\exists x)((\exists y)Py \Rightarrow Px), (\exists y)Py\ \vdash\ Pa$	`by exL`
$\neg(\exists x)((\exists y)Py \Rightarrow Px), Pb\ \vdash\ Pa$	`by notL`
$\neg(\dots), Pb\ \vdash\ (\exists x)((\exists y)Py \Rightarrow Px)$	`by exR` b
$\neg(\dots), Pb\ \vdash\ (\exists y)Py \Rightarrow Pb$	`by impliesR`
$\neg(\dots), Pb, (\exists y)Py\ \vdash\ Pb$	`by axiom`

$\square$